Introducing Use Cases in a small Organization: An Experience and Lessons Learned[1]

Dolors Costal, Xavier Franch, Nuria Rodríguez, Luis Delgado, Carme Jiménez

Universitat Politècnica de Catalunya (UPC), Barcelona (Spain)
{dolors|franch}@essi.upc.edu
{nuria.rodriguez-camara|luis.delgado|carme.jimenez}@upc.edu

Abstract. In this paper we report the adoption of use cases by a small organization in a university setting. Use cases were first introduced in the middle of a huge project and adopted thereafter for later projects. The paper mostly focuses in the first experience, whose most interesting characteristics were the large size of the resulting specification, the fact that it took place once the project had started (for documentation purposes instead of driving the development) and the limitation that resources allocated were not as much as required. We present the lessons learned from this experience.

1 Introduction

Use cases [1, 2] are widely accepted artefacts for describing the functional behaviour of a system. As a particular model of the specification, use cases are supposed to be defined in advance to the system itself (or more accurately, from an iterative point of view, a functionality should not be implemented before the use case that describes it).

We report an experience of adoption of use cases in a small organization. Use cases were introduced in the context of a particular project where the assumptions above failed. The undertaken project was the specification, for documentation purposes, of a large system, and it was developed in an organization that never before had used scenarios or use cases. The specification started more than one year after a first prototype of the system was deployed. Due to scheduling and budget constraints, resources were not allocated as required. We show which concrete problems appeared, and how they were tackled. We finally give some lessons learned.

The project ran at the PRISMA organization, a UPC unit focused on delivering software for the university. PRISMA, with 23 employees, is a typical example of small organization. Its first mission was to produce the PRISMA information system for providing an integrated management of academic data. A quality team was constituted (the authors of this paper). It was composed of two groups, one coming from the PRISMA organization itself, and the other by some researchers coming from the Software Engineering for Information Systems research group (GESSI) at the UPC. The collaboration embraced several activities, and building some UML artefacts [3] was one of them.

[1] This work has been partly supported by the *Ministerio de Ciencia y Tecnologia* and FEDER under project TIN2010-19130-C02-01.

2 Departing Situation and Decisions Made

Developing a UML documentation was not scheduled from the beginning of the quality assessment activity. The project manager took the decision of using UML for four purposes: (1) to validate systematically the suitability of processes and data, (2) to create a high-level documentation for future maintenance of the system, (3) to provide a view understandable for other prospective users, and (4) to obtain a "UML-label", considered a strategic goal by PRISMA managers for disseminating and trying to sell the product to other universities.

The use case specification was a milestone of this UML documentation. Due to the size of the project, it was foreseen to be a challenging activity. The use case specification started once a first prototype of the system was already running in some pilot schools. At that time, the PRISMA software engineering group was developing new functionalities and supporting the pilot schools. Due to the small size of the organization, the use case developer role was played by software developers. They had less time than required to write the use cases and they did not perceive use cases specification as important as implementation.

Due to these conditions, we (i.e., the quality team) took the following decisions:

- We prioritized the use cases because the manpower allocated to this task was not enough to develop all of them in detail. We considered as prioritisation units the modules that were already defined as PRISMA main functional areas.
- We wrote a very prescriptive use case template. We analysed some classical, well-established proposals (remarkably [4, 5, 6]), and developed a style sheet.
- We defined a methodology to develop the use cases, with special emphasis in stepwise refinement and linkage with the requirements management part.
- We designed tool support for the methodology. This tool support put together the previously selected Visure Requirements tool, which was used to write the use case diagrams, with Lotus Notes for writing the individual use cases.
- We designed a procedure for requirements change management. The procedure covered both the organizational side (e.g., who is in charge of doing what) and the technical side (e.g., how to use the tools to ensure consistency).

In this paper, we will focus on the first two issues.

3 Use Case Prioritization

The first decision we took was to divide the use case specification activity into two phases. The first phase, within project development yet, consisted in documenting some designated modules, whilst the second one aimed at finishing the specification during the first months of the project maintenance phase.

During the first phase, the prioritization criteria we used were: significance of the addressed functional areas and, within them, those functionalities that were more stable. Both "significance" and "stability" are fuzzy concepts, therefore we undertook a joint analysis with the development manager. Four modules were designated as the most significant because of their highly strategic nature and their large size: *Students and Files*, *Registration*, *Evaluations* and *Economic Management*. Between them, the

first two were pointed out as the most stable. We were in charge of developing the specification for them, interacting with the developers as questions arose. This kind of smooth interaction turned out to be very important for overcoming the barrier among developers and the quality team (see Section 5).

Once the running system became more stable and developers more available, the second phase started. Then, the rest of functional areas (i.e. *University Programs Management*, *Undergraduate Final Projects*, *Teaching Management* and *Information Management*) were documented in parallel by the developers under our direction.

4 Use Case Template

We proposed a template to establish the items that have to be included in a use case description. Together with the template we had the purpose of providing a clear definition of its key concepts, such as primary actor, stakeholder, etc., to facilitate its use by developers and create a common understanding in the organization. Altogether was encoded in a conceptual model with a glossary, which in fact can be considered a kind of ontology for our use cases.

In Fig. 1 we present a simple use case example from the PRISMA project. This use case has several extensions though only one is included below. The template is mostly based on Cockburn's fully dressed use case format [4] with the exception of the last two items: use case data definitions and auxiliary definitions.

Use case data definitions simplify the description and maintenance of use case scenarios where some long list of related data is referred to, by allowing the definition of composite data. In the PRISMA project, the data was synthesised basically from the data base definition and the user interface, which at some times exhibit some discrepancies; building the use cases was a good opportunity for detecting and solving them. Enclosing data definition in a separated data definition section facilitates the quick reference to this set of elements and increases understandability and modifiability of the use case while reducing its size. In addition, it facilitates maintenance because, if there is any change in personal data elements, a single definition has to be modified. In a similar way, auxiliary definitions simplify the description of use case steps where a complex calculation is performed or a complex term is used. For instance, in the first category falls the computation of the final mark of a student, a weighted sum taking into account marks, credits, type of course, etc. These two components are not part of the use case itself, they are separated and therefore may be referred from several use cases. References to them are underlined.

Use case understandability and independence from GUI were considered of special relevance for the purposes of the project. We considered many recommendations for use case development, most of them extracted from [4], to fulfil these goals. We found especially useful those related with lexical and syntactic conventions.

Use Case Name: Modify the data of a student.
Primary Actor: School.
Scope: Academic Record Management System.
Stakeholders and interests:
School: Wants to have available up-to-date data from its students.
Student: Wants the School have their data up-to-date.
Wants no one else gain access to their data.
Precondition: None.
Minimal guarantees:
When a student data is updated, his/her student identifier is not updated in such a way that another student with the same student identifier exists in the system
Success guarantees: Student's data has been updated.
Trigger: The School has a change request form filled by a Student.
Main success scenario:
1. The School indicates to the System the search identifier of the student whose data has to be updated.
2. The System shows to the School the Student personal data and Student access data.
3. The School indicates to the System the updates of these data of the Student.
4. The System registers the updated data of the student.

Extension: Wrong student identifier update.
Condition: The School has indicated an update of the Student identifier and there already exists another Student with the indicated new value.
Step: 4
Scenario:
1. The System communicates to the School that the Student identifier cannot be updated because there already exists another Student with the indicated new Student identifier.
2. Go to step 3 from main success scenario.

Use Case Data:
Student personal data: DNI, entry date, surname, name, birth date, country of birth, nationality, family address, address during semester, username, password, email.
Access data: access year, access semester, title, access date, access type, access mark.

Auxiliary definitions:
Search identifier of the student: DNI, passport or name and surname.
Student identifier: DNI or passport

Fig. 1. Example of use case

5 Lessons Learned

We summarise in this section some lessons learned that may be considered the scientific outcome of our experience and that we summarize in this section. We have focused on lessons about: 1) criteria that affect the general appearance of use cases; 2) project management with use cases; 3) technical aspects of use case definition.

Lessons on use case design criteria:

Lesson 1. Do not get rid completely of the existing implementation.

- It is necessary to take into account the general structure of the solution (not the user interface).

- Define packages in the use case model which correspond to main system areas. This makes easier the involvement of the development people.
- Use navigational maps to identify includes and alternative courses.
- Use forms to check whether all relevant data is included in the specification.

Lesson 2. Give preference to robustness over accuracy for dealing with small variations of business processes.

- Useful in cases such as ours, in which different units of an organization have essentially the same business processes but differ slightly in their implementation.
- By using abstract actors, the number of use cases dramatically decreases, because it is not necessary to differentiate the several variants.
- Future changes in these details may not imply changes in the use case template.

Lessons on project management:

Lesson 3. Prioritise use cases if resources are insufficient.

- When it is obvious that resources are not enough to develop all use cases, which is often the case for small organizations, it is preferable to perform an explicit task of prioritization than to leave this undefined.
- Define criteria for the prioritization of use cases. It is important to provide a rationale for prioritization decisions. In our experience, strategic relevance of use cases, stability and size were used (see Section 3). Criteria should be as objective and measurable as possible.
- Group use cases in prioritization units and apply criteria.

Lesson 4. Do not start to write the use case specification until a template has been defined. Otherwise, chances are that some work has to be redone.

- The large size requires a uniform style to write, learn, validate, compare, etc., in an easy way.
- Skills are acquired more quickly and productivity increases.
- Facilitates the writing of a checklist that makes possible the verification of the use case properties.

The template shall define both the parts of the use case and syntactic conventions.

Lesson 5. Introduce use cases smoothly in the organization. In a small organization, introducing a new activity or artefact is especially tough.

- Try to involve developers by encouragement instead of obligation. Try to find quickly some advantage of using use cases applied to a developer problem.
- Provide templates, methods, procedures, etc., in a very plain way with well-defined steps and parts.
- Do not force the developers to learn new tools further than necessary.

Lesson 6. Monitor continuously use case models to reduce risks. Among the most characteristic issues to monitor, we have:

- The need of abstracting from the user interface details, for avoiding commitment to a specific style of data communication.
- The need of distinguishing the scope, because system requirements do not focus on actor-actor interactions, which become of interest when writing use cases.

- Writing use cases in parallel with system implementation is hard. Functionalities evolve, some parts (e.g., data forms) are not updated timely, etc. Therefore validation is continuously needed.

Lessons on use cases technical aspects: We emphasize which are the most critical aspects (from those described in Section 4) that contribute to use case understandability and to save use case development resources.

Lesson 7. Introduce mechanisms to define the structure of system data and a glossary for domain-specific terms.

- Makes the specification self-contained by encoding knowledge about the domain.
- Ensures coherence among different use cases.
- Makes use cases text (and thus their development time) shorter.
- Supports validation with existent GUI forms.
- Supports maintenance with respect to changes in data or domain definition.

Lesson 8. Do not abuse of use case relationships.

- With large specifications, too many relationships make the structure of the specification difficult to understand.
- Workflows referred to change management get complicated: a change on a requirement makes it difficult to detect all the use cases that are affected.
- They raise some minor technical points: linkage among the included and the one that includes, incorrect use of preconditions, etc.

6 Conclusions

We have presented some relevant issues of the process followed for introducing use cases in a small organization. The experience can be considered successful. A very complete use case model was built for the first project fulfilling the goals enumerated in Section 2. The developers inside the PRISMA organization were a bit sceptical with respect to use case modelling. However, we overcame the difficulties found during this long process and at the end they found use cases useful. The organization recognized the convenience of use case models and they are using them in the ongoing projects.

References

1. I. Jacobson, M. Christerson, P. Johnson, G. Övergaard. *Object-Oriented Software Engineering, a Use-Case Driven Approach.* Addison Wesley, 1992.
2. I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 2004.
3. G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide* (2nd edition). Addison Wesley, 2004.
4. A. Cockburn, *Writing Effective Use Cases*. Addison Wesley, 2002.
5. S. Adolph, P. Bramble. *Patterns for Effective Use Cases*. Addison Wesley, 2004.
6. I. Alexander, N. Maiden (eds.). *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. John Wiley and Sons Ltd, 2004.