Use-Cases Engineering with UC Workbench

Jerzy Nawrocki and Łukasz Olek

Poznań University of Technology, Institute of Computing Science,
ul. Piotrowo 3A, 60-965 Poznań, Poland

{Jerzy.Nawrocki, Lukasz.Olek}@cs.put.poznan.pl

Abstract. Use-case methodology is widely used for writing requirements. Unfortunately, there are almost no tools supporting the usage of the use cases in any software projects. UC Workbench is a tool that tries to fill in this gap. It contains a use-case editor (it spares up to 25% of the effort at introducing new use-cases and up to 40% at updating them), a generator of mockups (a mockup generated by UC Workbench animates the use-cases and illustrates them with screen designs) and Software Requirement Specification documents, and an effort calculator, based on Use-Case Specification documents.

1 Introduction

Use cases have been invented by Ivar Jacobson [10], as a way of specifying functional requirements. Later on they have been incorporated into the Unified Software Development Process [11] and the Rational Unified Process [13]. Since then, they have been gaining more and more popularity. A significant step forward, in the development of the use-case methodology, has been taken by Alistair Cockburn and his colleagues, who proposed the set of the use-case patterns [1, 4]. Unfortunately, the use-case methodology is not adequately supported by the existing tools. Even editing of the use-cases is a problem. The most popular form of the use cases is a sequence of steps specifying the so-called main scenario with a number of exceptions (called also extensions) describing alternative behavior (see e.g. [1], [4], [8]). Since steps are numbered and those numbers are referenced in exceptions, deleting or inserting a step requires the re-numbering of all the subsequent steps, and accordingly all the exceptions associated with them. Unfortunately, it is not easy to find an editor that would support deleting and inserting a step into the use case. Even Rational Requisite Pro [19], a requirements engineering tool, developed by a company for which Ivar Jacobson worked for many years, does not have such functionality. According to our findings the only tool that supports the use cases is the use-case editor, offered by Serlio Software [26]. However, editing is just one, in a wide range of the use-case-related activities that could be supported by a computer. The aim of the paper is to present a tool, UC Workbench, supporting various activities concerning the use cases. UC Workbench is based on a semi-formal language, FUSE, in which the use-cases are described (see Sec. 2). The first and very restricted version of the tool has been described in [16]. The version of UC Workbench presented in this paper supports the following use case engineering activities:
– Editing of use cases (see Sec. 3).
– Performing automatic reviews (also Sec. 3).
– Generating a mockup (Sec. 4).
– Composing a software requirements specification document (SRS document for short) compliant with IEEE Std 830 [9] (Sec. 5).
– Generating effort calculators based on Use Case Points [11, 18] that support defining the scope of a project stage (Sec. 6).

What is important, in UC Workbench, is the fact that there is only one representation of a set of use cases describing a given system, and from that representation a mockup, an SRS document, and effort calculators are derived. It further assures that all those artifacts are always consistent one with the other.

In Sec. 7 a simple quasi-experiment is described, which aimed at checking if UC Editor (which is part of UC Workbench) can be really helpful.

2 FUSE: A Language for Use-Cases Description

The use cases collected by the analyst are edited with the help of UC Editor, a part of UC Workbench. The editor uses FUSE (Formal USE cases) language. It is a simple language formalizing structure of use-cases description to allow generating of mockups and effort calculators (actor descriptions and steps within the use cases are expressed in a natural language).

FUSE is based on use-case patterns collected by Adolph and his colleagues [1]. Use-case written in FUSE follows ScenarioPlusFragments pattern: the description consists of the main scenario split into a number of steps and extensions expressed in the natural language. To make the formal description of the use-cases more precise and readable we have decided to introduce the Either-Or and repeatable steps constructs to FUSE. Moreover, for the sake of readability, FUSE allows nested steps that are especially helpful when combined with the above constructs.

One can argue that the above constructs can be difficult for some end users to understand. In the case of UC Workbench it should not cause problems, as the use cases are accompanied by an automatically generated mockup, which visualizes the control flow by animating the use cases.

2.1 Repeatable Steps

Sometimes a step can be repeated more than once in a scenario. Assume that using the use-case UC-2: Adding a book to the cart someone wants to describe how to buy books in an Internet-based bookstore. Writing this in the following way:

1. Customer adds a book to the cart (UC-2).
2. Customer pays for the selected books.

is not fully correct, because it suggests that the Customer can add only one book to the cart. We can try to fix it by adding the extension:
1a. Customer wants to add another book.
   1a1. Return to step 1.

Unfortunately, that looks quite artificial and would clutter the description. To solve the problem we have decided to mark steps that can be repeated more than once with a star ("*") sign. Now, we can describe the situation in the following way:

1. *Customer adds a book to the cart (UC-2).
2. Customer pays for the selected books.

2.2 Either-Or

This construct was introduced to express a nondeterministic choice between alternative steps (more details and motivation can be found in [16]). The situation of buying books in a bookstore (extended with the removing a book from the cart) can be described in the following way:

1. Either: Customer adds a book to the cart (UC-2).
2. Or: Customer removes a book from the cart.
3. Or: Customer pays for the books in the cart.

2.3 Nested steps

Sometimes a single step can be subdivided into 2 or 3 other steps. Then it can be convenient to have the "substeps" shown directly in the upper level use case, without the necessity of creating the use case on lower level. For instance, assuming that UC-2: Adding a book to the cart it has only two steps buying books could be described in FUSE in the following way (use-case header and extensions have been omitted):

1. Either: Customer adds a book to the cart.
   1.2. System shows new value of the cart.
2. Or: Customer removes a book from the cart.
3. Or: Customer pays for the books in the cart.

3 UC-Editor and automated reviews

Figure 1 presents a UC-Editor’s screen with two examplary uses of the cases written in FUSE: Running an Online Bookshop and Ordering Books. Each use case is assigned a tag (B1 or U1). While editing use cases, step numbers are not shown on the screen (but they appear in the specification document and in the mockup). References between steps are described by means of labels.

In an ideal case UC-Editor would not only support the editing of the use cases but it would also check their quality automatically. Obviously, that is not possible. However, the FUSE language presented in the previous section allows automatic detection of the following "bad smells" concerning use cases:
Stranger — An actor appearing in a step has not been defined (he is not on the actors list). Sometimes it is caused by using two different words for the same role (e.g. customer and user).

Lazy actor — An actor is on the actor’s list but he does not appear in any step.

Too short or too long scenarios — A scenario contains less than 3 or more than 9 steps [1].

Too many extensions — The total number of steps within the main scenario and in all its extensions is greater than a threshold (in our opinion it should be 16).

Incomplete extensions — An extension contains an event but no steps associated with it.

Too much happiness — More than 50% of the use cases have no extensions (the description is dominated by "happy-day scenarios").

Dangling functionality — A given system-level use case is not used by any of the business-level use cases.

Outsider — A given business-level use case is not supported by any system-level one (it appears to be outside of the system scope, at least for the time being).

The above mentioned bad-smell detection is intended to support the two-tier-review approach [1] to use-case quality assurance.

4 Generating of Mockups

There are two kinds of prototypes [18]: throwaway prototypes (mockups) and evolutionary ones. The latter are the core of every agile methodology. The former could be used to support customer-developers communication concerning requirements, however, their development had to be very cheap and very fast.
The mockups generated by UC Workbench are simple and effective. They focus on the presentation of functionality. They combine the use cases (i.e. behavioural description) with screen designs that are associated with them (that complies with the Adornments pattern [1]). A generated mockup is based on a web browser and it consists of two windows (see Fig. 2):

- the *scenario window* presents the currently animated use cases (it is the left window in Fig. 2) and the current step is shown in bold;
- the *screen window* shows the screen design associated with the current step (it is the right window in Fig. 2).

The animation process starts with the presentation of all the actors and all the business-level use cases (see Fig. 3). By clicking an actor one can get the actor description and the list of the use cases in which the actor participates (see Fig. 4). By selecting a business-level use case one will be able to go down through the use cases DAG (direct acyclic graph) created by the *include* relation.

To generate a mockup the analyst has to associate with use-case steps names of files containing screen designs. The fidelity levels of screen designs are up to the analyst (an interesting discussion about fidelity levels can be found in [21], [25], [14], [17], [23]). The screen design shown in Fig. 2 is at the low fidelity level and it has been created with a tablet connected to PC. After decorating ‘difficult’ use-cases with screen designs one can generate a mockup "at the press of a button". Using UC Workbench one can produce a mockup (i.e. re-write use cases and adorn them with screen designs) within a few hours — that can really support customer-developers communication, especially if there is a danger of unstable, ambiguous or contradictory requirements.
Fig. 3. Choosing actor or business process

Fig. 4. Actor details with the list of use cases in which the actor participates
5 Composing the SRS Document

Someone can think that when a mockup is available no Software Requirements Specification (SRS) document is needed (the functionality is already shown by the mockup). There are two arguments that motivated us to support the composition of the SRS documents:

– **Nonfunctional requirements.** Assume one of the requirements specifies that the system must be able to store up to 20 000 personal records. To which use case this ‘adornment’ should be assigned?
– **Customer’s attachment for documentation.** If one wants to have a collaborating customer, he must show appreciation for his experience and believes. One of Covey’s principles [7] is ‘*First seek to understand, then to be understood*’.

To solve those problems we have decided to extend UC Workbench with an SRS Composer. Our SRS template is base on IEEE Std 830 [9] and it consists of the following sections:

1. Introduction
2. Business Model
3. Functional Requirements
4. Nonfunctional Requirements

Sections 2 and 3 are generated directly from the use-case descriptions. The Business Model contains subsections describing actors, business-level use cases and business objects. The Functional Requirements contain system-level and auxiliary use-cases assigned to subsequent actors. Sections 1 and 4 are written by the analyst as separate \LaTeX files. Actors, business objects and the use-cases are referenced by non-functional requirements via tags. If a non-functional requirement references e.g. a use-case that has been removed then the analyst will get a warning.

6 Effort Estimation and Scope Planning

UC Workbench supports effort estimation and scope planning based on use cases. As a framework we have chosen Use-Case Points proposed by Gustav Karner [12], [20]. In our approach scope planning comprises three layers: Planning Game borrowed from XP (highest layer), Wideband Delphi, and automatic effort calculation (lowest layer). Automatic effort calculation (AutEC) provides initial values that are later modified by experts during Wideband Delphi session. Such a session is part of a Planning Game which provides the customer (or his representative) with information necessary do define the scope for the next development stage.

XP’s Planning Game is played on a table on which story cards are put. In our approach user stories are replaced with system-level use cases and the table is replaced by a computer running AutEC (AutEC is a specialized spreadsheet which relieves estimation experts from tedious calculations).
The original Use-Case Points help to estimate effort for a system defined by a set of use cases. We have adjusted the method to allow to estimate effort for single use cases and then to define scope of the increment by selecting most important – from customer’s point of view – use cases (their total effort cannot exceed man-hours allocated for the increment).

The main part of AutEC screen consists of rows corresponding to the use-cases. Each row contains:

- *use-case name* with a hyperlink to use-case text,
- *effort estimated by AutEC* used by experts as a starting point for their discussion (see below),
- *effort estimated by experts* and following from their discussion,
- *summary of experts estimations* presenting the most optimistic (O), most pessimistic (P), and average effort (A) accompanied with most probable one equal to \((O + 4A + P)/6\).

AutEC can estimate the effort only for use-cases in the formal form (see Sec. 2). Use-case effort is computed by AutEC using the following formula derived from Use-Case Points:

\[
\text{UseCaseEffort} = 25 \cdot \text{UCW}(s) \cdot \text{TF} \cdot \text{EF}
\]

\(\text{UCW}\) is Use-Case Weight and it depends on the number of steps, \(s\), in a given use-case. Originally \(\text{UCW}\) was equal to 5, 10 or 15 points depending on the number of 'transactions' (here use-case steps). For up to 3 steps \(\text{UCW}\) was equal to 5, and for 8 or more steps \(\text{UCW}\) was 15. It means that reducing a 7-steps-long use-case by 2 steps has no meaning but reducing a 4-steps-long use-case just by 1 step can decrease its complexity by half. To remove this effect, we have changed this rule and defined \(\text{UCW}\) in the following way:

\[
\text{UCW}(s) = 17 - 20 \cdot \exp -s/5
\]

where \(s\) represents number of steps. For the proposed function the average value of \(\text{UCW}\) for 2 and 3 steps is almost 5 (4.81), and for 4, 5, 6, and 7 steps it is about 10 (10.2) — the same holds for the original \(\text{UCW}\) function proposed by Karner.

\(\text{TF}\) and \(\text{EF}\) denote technical and environmental factors (\(\text{TF}\) depends on 13 factors, and \(\text{EF}\) on 8). The factors are evaluated by the experts and are the subject of their discussion. All the changes made to \(\text{TF}\) and \(\text{EF}\) factors managed by AutEC are immediately visible on the use-case rows in the main part of the AutEC screen. This allows the experts to speculate about influence of non-functional requirements on the effort.

The original Use-Case Points depend not only on use-cases but also on the complexity of the actors. There are three kinds of actors: simple (API interface), average (TCP/IP or text interface), and complex (GUI interface). AutEC assigns effort to actors in the following way:

\[
\text{ActorEffort} = 25 \cdot \text{AW} \cdot \text{TF} \cdot \text{EF}
\]
where $AW$ equals 1 for simple actors, 2 for average, and 3 for complex. AutEC shows the effort connected with actors in a separate row on the main AutEC screen. Only actors involved in a selected set of use-cases are taken into account.

7 Evaluation of the tool

In December 2004 we conducted an experiment comparing the effort needed to prepare the use cases using a general-purpose text processor, MS Word, and UC Editor. Twelve students participated in the experiment. We have divided them into two groups: six people were using MS Word and remaining six UC Editor. Students were provided with drafts of 4 use cases (each of them contained 6-9 steps), and their task was to Prepare the use-case-based specification using the assigned tool. The task took from 40 to 80 minutes. At the next stage the students had to introduce some changes to the specification.

![Effort Comparison](image)

**Fig. 5.** Average effort necessary to prepare use cases using MS Word and UC Editor.

It turned out (see Fig. 5) that using UC Editor instead of MS Word one can save, on average, 25% of time (the standard deviation for UC Editor was 4.2 and for MS Word it was 17.2). The relative savings are even greater when one has to maintain use cases. Due to our experiment, one can save 40% of time (on average) by using UC Editor instead of a general purpose editor (the standard deviation for UC Editor was 1.6 and for MS Word it was 1.5).

We must emphasize that we measured only the effort of writing use cases, so it does not mean that the whole phase of requirements elicitation will take 25% less, when using UC Editor.
8 Conclusions

UC Workbench presented in the paper supports the editing the use cases (we were surprised that Rational Requisite Pro much more supports 'traditional' requirements than use cases), generates mockups that animate use cases, and creates effort estimators adjusted to the current set of use cases. Moreover, UC Workbench can generate Requirements Specification Document complying to one of the templates recommended by IEEE Std 830 [9]. What is important for agile developers, all those artifacts (i.e. mockups, estimators, and requirements documents) can be obtained automatically and each of them is consistent with the others. Early experience, based on university experiments and observations of projects performed both at academia and in industry setting, is quite promising. UC Workbench can save about 25% time when writing use cases from scratch and about 40% when modifying existing ones.

In the nearest future we are going to extend effort estimators (AutEC) with a historical database showing for past projects the estimated and actual effort, technology used in the project, and people involved in it. That should help the experts to provide better effort estimates.

Acknowledgements

We would like to thank our students, who participated in the experiment. We also thank Grzegorz Leopold and Piotr Godek from PB Polsoft – their brevity allowed us to get a feedback from industry and helped us to improve the tool.

This work has been financially supported by the State Committee for Scientific Research as a research grant 4 T11F 001 23 (years 2002-2005).

References