

Sketching Use-Case Scenarios Based on Use-Case Goals and Patterns

Mirosław Ochodek, Krystian Koronowski, Adam Matysiak, Piotr Miklosik and Sylwia Kopczyńska

Abstract Use cases are a scenario-based technique used to express functional requirements from the user perspective. They are often elicited using a top-down approach by first identifying their goals and later documenting their scenarios. In the paper, we investigate the possibility of supporting analysts in progressing from definitions of use-case goals to full documentation of their scenarios. We propose a semi-automatic approach to generate use-case scenarios based on use-case patterns. The proposed approach is a result of an empirical analysis of 217 use cases from 12 projects. The analysis revealed that a notion of use-case transaction could be used to organize use-case patterns into a catalog of patterns. We have implemented a prototype tool called UC-Sketch to illustrate the proposed idea. The acceptance of the proposed approach by its potential users was assessed through the use of Technology Acceptance Model.

Keywords Use cases · Use-case patterns · Use-case transactions · Technology acceptance model

1 Introduction

Use cases are a scenario-based technique for expressing functional requirements from the user perspective. They have gained a lot of attention from the software engineering community [1, 2] since their introduction in the 1980s [3].

Several authors recommended to elicit use cases according to a top-down and breadth-before-depth strategy [4, 5]. When following such an approach, firstly, business-level goals are identified [6], which further gives a basis for determining user-level goals and sub-functional goals. As a result, a hierarchy of use cases is created which forms a kind of “unfolding story.” The breadth-before-depth strategy

M. Ochodek (✉) · K. Koronowski · A. Matysiak · P. Miklosik · S. Kopczyńska
Faculty of Computing, Institute of Computing Science, Poznan University of Technology, Ul.
Piotrowo 2, 60-965 Poznań, Poland
e-mail: Mirosław.Ochodek@cs.put.poznan.pl

means that one should elicit use cases at a given level of abstraction before decomposing them to into a set of more precise ones. Consequently, the elicitation process of user-level use cases is often divided into two stages: *identification* of use cases, and *documentation* of their scenarios. The outcome of the first stage is a list of use-case names, usually visualized using UML use-case diagrams. In the second stage, use cases are analyzed and documented with scenarios.

Taking into account existing similarities between scenarios of use cases [7, 8], it would be beneficial to investigate if it is possible to use historical data concerning use cases to support the process of transitioning between use-case names (which express goals of use cases) and use-case scenarios. In this context, the following problem can be formulated:

Problem 1 Given the name of a use case (a definition of its goal), provide a plausible documentation of its scenarios.

A reasonable approach to address the above-stated problem is to identify and document patterns of commonly appearing use cases (or directly copy and modify previously created ones). Several catalogs of use-case patterns have been proposed so far [9, 10]. They provide guidance on how to model frequently appearing types of user goals with use cases, and usually provide some examples or templates. These examples might be used as a basis for authoring new use cases. However, there are at least two problems with such an approach. Firstly, (1) the examples need to be manually rewritten to match the context, and secondly (2) the aforementioned patterns usually focus on very specific, domain-oriented goals which limits their generalizability.

In the paper, we propose a different approach to organizing catalogs of use-case patterns. It allows structural patterns of use cases to be defined independently of the business domain at three levels of abstraction: use-case actions, use-case transactions, and scenarios, and associates them with use-case goals. In addition, we propose a method and prototype tool that enables semi-automatic generation of use-case scenarios based on use-case patterns. We call this approach “sketching a use case” because it allows the creation of an initial version of use-case documentation quickly. To develop the proposed approach we investigate the following research questions:

- **RQ1:** How to organize a catalog of use-case patterns that binds use-case goals and scenarios, but is independent of business domain?
- **RQ2:** How to store information about use-case patterns and generate use case scenarios in a semi-automatic way?

The paper is organized as follows. In Sect. 2, we discuss the related work concerning use-case patterns. Section 3 concerns RQ1 and describes the organization of a use-case patterns catalog. The following Sect. 4 addresses RQ2 and presents the process of sketching use cases and prototype tool called UC-Sketch. A preliminary evaluation of the usefulness of the proposed approach is presented in Sect. 5. Section 6 discusses limitations and generalizability of the study. The paper is concluded in Sect. 7.

2 Related Work

Use-case patterns could be defined at different levels of abstraction, i.e., for a set of use cases, a single use case, scenario, or step [4]. Also, the form used for expressing use-case patterns differs visibly between the existing catalogs of patterns. Some authors provide exemplary fragments of use-case models that solve some problems; others define use-case patterns as a set of guidelines.

Probably the most recognized catalog of use-case patterns was proposed by Övergaard and Palmkvist [10]. They documented several patterns of use cases and provided a catalog of blueprints that guides how to compose use-case models for some common features, e.g., access control, future task, report generation. A similar approach to documenting use-case patterns was proposed by Issa et al. [7, 9]. From the perspective of the considered Problem 1, these kinds of patterns could be used by copying the examples or templates they provide and adjusting them to a particular context.

Saeki [11] proposed a different approach to derive and organize use-case patterns. He suggested using the UML generalization relationship to derive a specific use case from a pattern, or to use the Decorator pattern to adjust existing use-case patterns to a given context. This approach makes it possible to maintain associations between use-case patterns and use cases. However, the resulting use-case model might be difficult to understand by the readers who are IT-laymen, not familiar with the UML notation.

A different approach to solving the considered problem is to find similarities between previously created requirements and the ones that are currently being analyzed. For instance, the problem of similarity analysis between textual requirements has been considered before by och Dag et al. [12]. In the context of use cases, Kaindle et al. [13] proposed to use similarity measures to find commonality between currently developed and historical use cases. Because the approach relies on previously-created use cases, it might have even wider applicability than use-case patterns. Unfortunately, to calculate similarity measures (and find similar use cases) one has to document at least a part of a new scenario.

There were also several studies focused on generating scenarios. These studies are not directly related to use cases. However, because use cases are a scenario-based technique, they seem applicable also in this context. For instance, Rolland et al. [14] proposed an approach to composing scenarios from so-called requirements chunks—a pair <Goal, Scenario>, and to use three types of relationships: composition, alternative, and refinement. However, they mainly focused on guiding goal modeling rather than documenting and re-using patterns. Ridao et al. [15] studied different types of episodes in scenarios. They formulated four main scenario patterns, i.e., production, collaboration, service, and negotiation. They also consider the possibility of automatically generating sets of candidate scenarios. However, the proposed approach requires the existence of a language extended lexicon (LEL), thus, it is not directly applicable to use-case names. Watahiki and Saeki [8] focused on generating scenarios from case frames (an abstract representation of a sentence

characteristic for a given domain, e.g., [reserve, actor, object]) that fulfill order constraints.

At the level of scenario actions, the research focused mainly on linguistic patterns. For instance, the most recognized are the language patterns proposed by Rolland et al. in the CREWS project [16]. These studies seem complementary to ours because the proposed linguistic patterns might be used to formulate action templates in the proposed approach.

We might conclude that there are at least several approaches that might support solving Problem 1. However, it seems that they do not provide a holistic approach that would cover the possibility of defining patterns at different levels of abstraction: actions, episodes of scenarios, use-case scenarios, and use-case goals. In addition, most of the existing approaches rely on copying and modifying the proposed exemplary use cases or templates to create new instances of use cases.

3 Use-Case Patterns Catalog

To answer RQ1, we have to find a way to associate use-case goals and appropriate patterns of scenarios. The classification of goals should be, as much as possible, independent of business domain.

We decided to investigate this question empirically, by analyzing a set of 217 use cases coming from 12 software projects that developed web applications [17]. In particular, we aimed at investigating the relationship between goals of use cases and use-case transactions forming their scenarios.

Use-case transaction is defined as “the smallest unit of activity that is meaningful from the actor’s point of view that is self-contained and leaves the business of the application being sized in a consistent state.” [18] In our previous studies [17, 19], we proposed a categorization scheme of transactions that consist of twelve types of use-case transactions: Create (C), Retrieve (R), Update (U), Delete (D), Link (L), Delete Link (DL), Asynchronous Retrieve (AR), Dynamic Retrieve (DR), Transfer (T), Check Object (CO), Complex Internal Activity (CIA), and Change State (CS). These types refer to general tasks and are not bound to any specific business domain.

During the analysis of the sample of use cases, we observed that more than half of user-level use cases (55 %) consisted of a single transaction that corresponded to the goal of a use case. We also observed that use cases consisting of more than one transaction usually had a *dominant transaction* that reflected the goal of a use case and a set of auxiliary transactions. For instance, in a use case entitled “modify a product,” the modification of the product could be preceded by the presentation of the products that could be modified. In such a case, a *dominant transaction* would be Update and Retrieve would play a supportive role. Therefore, we assume that *the type of dominant transaction might be used to summarize the goal of a use case*. The only exceptions from this rule were the CRUD (Create, Retrieve, Update, Delete) use cases that have more than one type of dominating transaction. Thus, we extended the

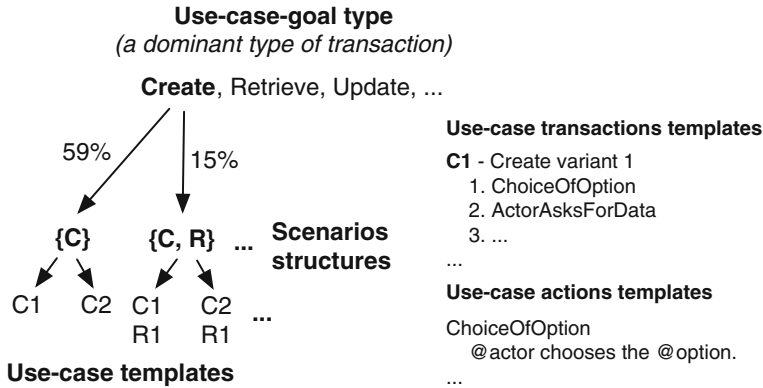


Fig. 1 Organization of the use-case patterns catalog

set of goals categories with CRUD. In fact, the set of categories can be extended if new types of goals are identified.

The resulting structure of a use-case catalog is shown in Fig. 1. It is organized into three layers. The first layer consists of categories of use-case goals. In the second layer, the goals are associated with the structures of use cases. We define structure by providing a set of acronyms of types of transactions it includes (e.g., {C, R} consists of Create and Retrieve). The additional benefit is that we could augment the catalog with the information about the probability of a given structure appearing in the use case of a given type of goal. This information might be used to suggest analyst the most probable pattern. In the third layer, each structure of scenario is associated with a set of *use-case templates*. Such templates are defined at three levels. The smallest unit is called *action template*. These templates are used to compose *transaction templates* that are used to define scenarios in use-case templates.

In the analyzed set of use cases, we identified 57 different structures of use-case scenarios. The most frequently appearing ones are presented in Table 1. They seem to be good candidates for use-case patterns.

3.1 Action Templates

Use-case action represents the smallest unit of activity in a use-case scenario perform by an actor. Each action template has a *unique identifier*, *name*, *parameters*, and *language transforms*.

Parameters are used to determine the fragments of a template that can be configured while generating the text based on the template. A parameter can have a set of default values (one per supported natural language). For instance, a parameter @actor in the action “select object” might have a default value “uytkownik” in Polish and “user” in English.

Table 1 The most frequently appearing of configurations of transactions among the analyzed set of 217 use cases from 12 software projects

Goal of use case	Structure of scenarios	Frequency	Goal of use case	Structure of scenarios	Frequency
C	{C}	0.59	L	{L}	0.60
	{C,R}	0.15		{DL,L,R}	0.16
R	{R}	0.47		{L,R}	0.08
	{DR,R,R}	0.13	DL	{DL}	1.00
	{R,R}	0.08	AR	{AR}	1.00
U	{U}	0.73	DR	{DR}	0.86
	{R,U}	0.15		{DR,R}	0.14
D	{D}	1.00	CIA	{CIA}	1.00
T	{T}	0.60	CS	{CS}	0.55
	{R,T}	0.20		{CS,R}	0.27
	{R,T,U}	0.20	C,D,R,U	{C,D,R,U}	0.41
CO	{CO}	0.80		{C,D,U}	0.31
	{CO,R}	0.20		{C,D,L,U}	0.10

Language transforms are boilerplates that are used to generate instances of actions in a given language. They may include references to parameters with the specification of their linguistic forms (e.g., the number—singular, plural; form—normative, accusative, etc.).

Parameters and transforms are important in the context of RQ2 because they provide a simple natural language generation (NLG) mechanism.

The execution of an action might be interrupted by events, which are described in the extensions section of a use case. Event templates are defined similarly to action templates.

3.2 Transaction Templates

Transaction template is defined by a *unique identifier*, *name*, *name of type*, and *description*. Similarly to actions, they can define *parameters*.

The structure of transaction is defined by a sequence of *actions* and corresponding *extensions*. Actions templates are included by referring to their unique identifiers discussed before. Also, values of parameters can be passed between transaction and action templates, which is similar to invoking a programming function with parameters. One can also bind the values of parameters defined at the level of transactions with the parameters in the action template. For instance, if a parameter @actor is defined at the level of transaction, one can bind its value with the parameter @actor defined by the action “select object” that is included in the transaction’s template.

This mechanism allows the flow of data between templates defined at the higher level of abstraction and the ones at the lower levels.

Extensions consists of an *event* that may occur while performing an action, and a sequence of *actions/transactions* describing the alternative scenario, and a set of *parameters*. Optionally, one can state which transaction should be performed after executing the alternative scenario.

3.3 Use-Case Templates

Use-case template has a *unique identifier, name*, definition of *goal*, and *description* giving the purpose of the use-case pattern.

Similarly to definitions of templates at other levels, it can also define a set of *parameters*. The scenario of a use case is a sequence of *transactions* and optional *extensions*. The parameters at the level of a use case can be bound with the parameters of included transactions and extensions (in the same way as for transactions and actions).

4 Sketching a Use Case Based on Its Goal

The process of sketching a use case based on use-case pattern is presented in Fig. 2 and consists of the following steps:

1. The goal of a use case is identified and documented as a use-case name.
2. Based on the goal, an analyst determines the dominant type of transaction in the use cases, chooses one of the structures of use-case scenarios available for the goal and one of the associated use-case templates.
3. A set of templates parameters that might be modified by the analyst is determined. This process begins with identifying parameters in the included action templates. Afterward, the parameters of transaction templates are determined. If any of these parameters is bound to the parameters of included actions, the action parameter is excluded from a set of modifiable parameters because its value will be automatically determined based on the transaction's parameter. In the following step, the actions' transforms are analyzed to determine the required linguistic form of a parameter. Finally, the analyst is provided with the default values of parameters and asked to modify them appropriately.
4. The values of parameters and transforms are merged to generate the text in a given natural language.

The above-described process of sketching use cases was implemented in a prototype tool called UC-Sketch¹ (see Fig. 3). The tool provides most of the features pre-

¹<https://ucsketch.cs.put.poznan.pl>.

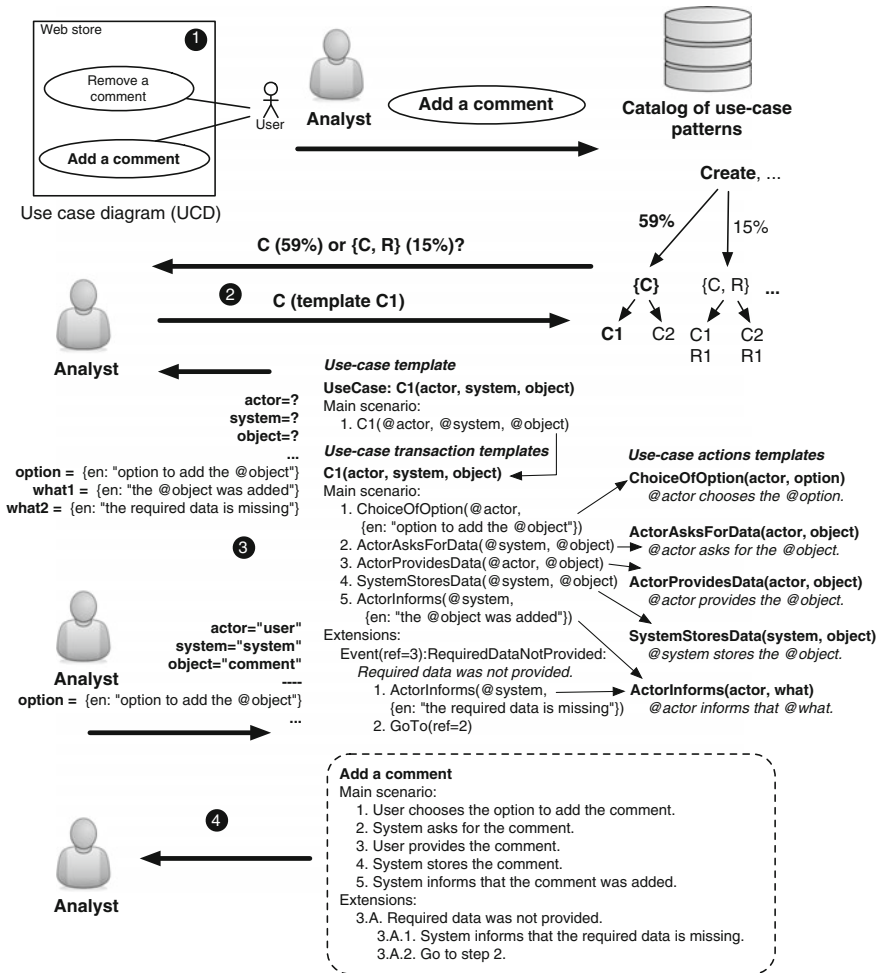


Fig. 2 The overview of the process of sketching a use case based on use-case pattern

sented in the paper and some additional ones increasing its usability. For instance, it predicts the type of dominant transaction based on use-case names. It is also integrated with Wiktionary,² which is used to fetch the required linguistic forms of words.

The built-in catalog of use-case patterns provides a set of patterns identified during the study (available in Polish and English). The patterns are stored using a dedicated XML-schema and can be changed through a back-end of the system.

²<https://www.wiktionary.org>.

UC1

Add a comment

Hint: Create

C - Create

69.57% Create (C)

Preview use case

Use case description

Number / ID

Use case title

Main transaction

Pattern

1. actor chooses the @optionToAdd the @objects.

2. @system asks to provide information about the @objects.

3. actor provides to the @system information about the @objects.

4. @system validates the @objects in respect to @criteria.

5. @system stores the @objects.

6. @system informs that @objectAddedMessage.

Extensions

4. Wrong data provided or missing data.

4.1. @system informs that @wrongOrMissingDataMessage.

4.2. Go to step @returnStepIdWrongMissingData.

Insert parameter

@actor

User

@criteria

@objectAddedMessage

the adding operation c...

@objects

comment

@optionToAdd

option to add

@returnStepIdWrong...

2

@system

System

@wrongOrMissingDat...

wrong or missing data ...

Add new parameter

Add

Insert action

Insert transaction

Insert extension

Back

Save

Turn on autosave

Fig. 3 UC-sketch use-case editor—sketching a use case view

5 Preliminary Evaluation

We decided to perform a technology acceptance forecasting study based on Technology Acceptance Model (TAM) [20] to investigate if the proposed approach and tool have a chance to be accepted by its potential users.

TAM is an information system theory that models how users come to accept and use technology. The purpose of this model is to predict the acceptability of information system and to identify the modifications which must be brought to the system to make it acceptable to the users. The model assumes that the acceptability of a system could be determined by measuring two main factors:

- **perceived usefulness (PU)** which is “the degree to which a person believes that using a particular system would enhance his or her job performance.”
- **perceived ease of use (PEOU)** which is “the degree to which a person believes that using a particular system would be free from effort.”

Devis [20] proposed to measure PU and PEOU using six Likert’s items per each of them. All items create a summated rating scale. The answers for each item is treated as being expressed on a 7-point interval scale. The final score of a respondent is the sum of his/her answers to all items.

We conducted the TAM survey based on questionnaires provided in the appendix of Davis’s work [20] within three groups of potential users of UC-Sketch:

- 3rd-year B.Sc. computer science students.

Table 2 Demographic information of the sample (values in brackets: 3rd-year students/4th-year students/practitioners)

Variables	Number (N)	Percent (%)
Research group		
Bachelor students	39	61.90
Master students	5	7.94
Professionals	19	30.16
Years of experience		
0–2	39 (32/3/4)	61.90
3–5	17 (6/1/10)	26.98
6–8	6 (1/1/4)	9.52
9–11	0 (0/0/0)	0.00
12 or more	1 (0/0/1)	1.59
Use case experience		
Never heard	0 (0/0/0)	0.00
Heard, but not used	2 (1/0/1)	3.17
Wrote a few UC	32 (30/2/0)	50.79
Used in one project	14 (4/3/7)	22.22
Used in many projects	15 (4/0/11)	23.81
Work experience (multiple answers possible or blank)		
Programmer	52 (28/5/19)	82.54
Architect	12 (4/1/7)	19.05
Analyst	12 (4/2/6)	19.05
Project Manager	8 (1/1/6)	12.70

- 4th-year students of M.Sc. degree in Software Engineering.
- Professionals working in software development industry (we did not include students who are working in IT companies).

Demographic information of the sample is presented in Table 2. Most of the participants have authored few use cases in their career. Unfortunately, only 16.67 % of respondents performed the role of an analyst (four of them were 3rd-year students, who had less than two years of experience in IT projects).

We began the analysis by verifying reliability and validity of the measurement instrument. To measure reliability we used Cronbach's alpha. It is agreed that the acceptable level of the alpha measure should be equal to 0.70 or above. We calculated alpha for each variable and group of respondents and received values between 0.89 and 0.98. Therefore, the measurement instrument seemed reliable. To verify validity we used Factor Analysis [21], which confirmed that two factors were measured by the instrument.

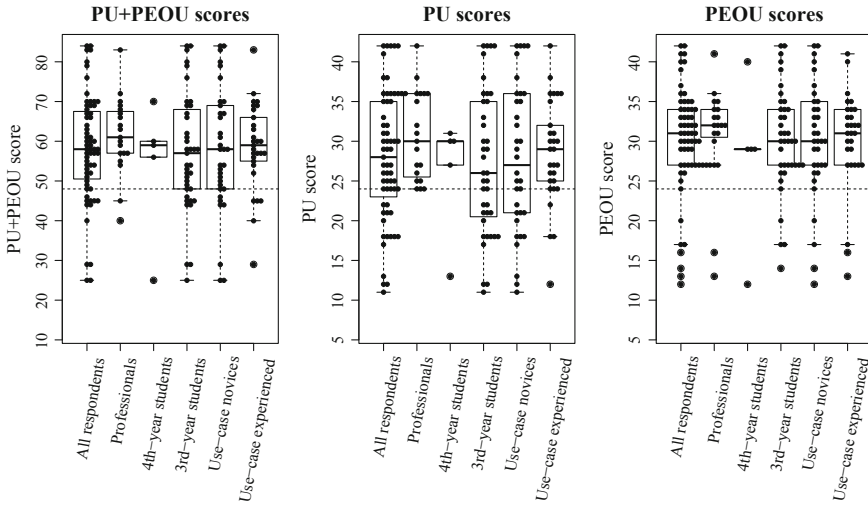


Fig. 4 The results of TAM analysis: PU, PEOU, and PU+PEOU scores (*dashed line*: a central point of the scale)

Finally, to interpret the results we summed up the scores for each participant. The scale ranged from 6 to 42 (6 items $\times$ 1:7) for a single variable and from 12 to 84 for two variables together.

The distributions of scores are presented in Fig. 4. The overall average score of participants was ca. 58 for all groups, which is 10 points above the midpoint of the scale, and could be interpreted as 69 % of the maximum value. From the two variables PU and PEOU, the higher results were observed for PEOU (according to the model it also affects PU). However, even PU was higher than the midpoint of the scale. It was higher for professionals and master students than for 3rd-year students. Similar results were observed when we divided participants based on their experience in applying use cases.

Based on the results, we might conclude that the respondents seemed positive towards using the proposed tool.

However, there are at least two threats to validity. First of all, the sample was not a homogeneous sample of analysts—the most appropriate respondents. The second problem is that the TAM method assesses the product rather than the concept. Therefore, it might be difficult to state whether participants liked/disliked the idea or particular features of the software tool.

6 Limitations and Generalizability

Although the results of the TAM analysis shows that the proposed approach to sketching use cases is perceived as useful, there are some limitations of the proposed approach.

First of all, the process of generating use-case scenarios based on use-case goals at early stages of requirements analysis is subjected to the risk of proposing a misleading scenario because of the uncertainty related to requirements. The other issue is the accuracy of the out-of-the-box generated scenarios. In some cases, the generated scenarios might be perceived as correct although they miss some important details. Therefore, a generated scenarios should be treated as a “sketch” and should be revisited and refined later on.

An important threat to the generalizability of the proposed approach is the approach used to author use cases. The use cases analyzed in the study were written in accordance with the state-of-the-art guidelines for writing use cases [4, 5]. Therefore, at the moment, we can generalize our observations only to such use cases.

7 Conclusions

In the paper, we investigated the problem of semi-automatic generation of use-case scenarios based on use-case goals with the use of use-case patterns.

We propose a new approach to organizing catalogs of use-case patterns and use this information to support analyst in documenting use cases by generating proposals of use-case scenarios.

The analysis presented in the paper allow us to provide the following answers to the research questions:

RQ1: *How to organize a catalog of use-case patterns that binds use-case goals and scenarios, but is independent of business domain?*

Based on the analysis of 217 use cases, we proposed a three-layered organization of use-case patterns catalog. The categorization is based on the concept of semantic type of use-case transaction. The considered types of transactions seem general and not bound to any particular business domain. The first layer of the catalog characterizes use-case goals based on dominant types of transactions in their scenarios. The second layer consists of structures of scenarios (types of transactions) that appear in use cases with a given type of goal. In the third layer, we store templates of use cases.

RQ2: *How to store information about use-case patterns and generate use case scenarios in a semi-automatic way?*

We proposed to store templates of use-cases at three levels: actions, transactions, and use cases. The templates on a higher level are created by including (referring to) the templates on the lower level. The two important mechanisms that allow semi-

automatic generation of use-case scenarios are parameters which can be passed and bound between templates and language transforms that are used to generate text in the natural language.

We call the proposed approach sketching a use case because it allows generating initial documentation (a sketch) of use-case scenarios, which can be further altered by an analyst (e.g., by providing values of parameters, modifying the generated text).

To evaluate the potential acceptance of the proposed approach, we developed a prototype tool called UC-Sketch and performed an acceptance forecasting study on 63 potential users, with the use of Technology Acceptance Model (TAM). The results of the study suggest that the target users of the proposed approach are willing to use it.

References

1. Neill, C., Laplante, P.: Requirements engineering: the state of the practice. *IEEE Softw.* **20**(6), 40–45 (2003)
2. Tiwari, S., Gupta, A.: A systematic literature review of use case specifications research. *Inf. Softw. Technol.* **67**, 128–158 (2015)
3. Jacobson, I.: Object-oriented development in an industrial environment. *ACM SIGPLAN Notices* **22**(12), 183–191 (1987)
4. Adolph, S., Bramble, P., Cockburn, A., Pols, A.: *Patterns for Effective Use Cases*. Addison-Wesley (2002)
5. Cockburn, A.: *Writing Effective Use Cases*. Addison-Wesley, Boston (2001)
6. Nawrocki, J., Nedza, T., Ochodek, M., Olek, Ł.: Describing business processes with use cases. In: Abramowicz, W. (ed.) *Proceedings of the Business Information Systems Conference. Lecture Notes in Informatics*, vol. P-85, pp. 13–27. Koellen Druck+Verlag (2006)
7. Issa, A., Al-Ali, A.: Use case patterns driven requirements engineering. In: 2010 Second International Conference on Computer Research and Development, pp. 307–313. IEEE (2010)
8. Watahiki, K., Saeki, M.: Scenario patterns based on case grammar approach. In: Fifth IEEE International Symposium on Requirements Engineering, 2001. *Proceedings*, pp. 300–301. IEEE (2001)
9. Issa, A., AlAli, A.: Automated requirements engineering: use case patterns-driven approach. *IET Softw.* **5**(3), 287–303 (2011)
10. Övergaard, G., Palmkvist, K.: *Use Cases: Patterns and Blueprints*. Addison Wesley Professional (2004)
11. Saeki, M.: Reusing use case descriptions for requirements specification: towards use case patterns. In: *Software Engineering Conference, 1999. (APSEC'99) Proceedings. Sixth Asia Pacific*, pp. 309–316. IEEE (1999)
12. och Dag, N.J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: Evaluating automated support for requirements similarity analysis in market-driven development. In: 7th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'01), pp. 190–201 (2001)
13. Kaindl, H., Smiałek, M., Nowakowski, W.: Case-based reuse with partial requirements specifications. In: 2010 18th IEEE International Requirements Engineering Conference (RE), pp. 399–400. IEEE (2010)
14. Rolland, C., Souveyet, C., Achour, C.: Guiding goal modeling using scenarios. *IEEE Trans. Softw. Eng.* **24**(12), 1055–1071 (1998)
15. Ridao, M., Doorn, J., do Prado Leite, J.: Domain independent regularities in scenarios. In: Fifth IEEE International Symposium on Requirements Engineering, 2001. *Proceedings*, pp. 120–127. IEEE (2001)

16. Rolland, C., Achour, C.: Guiding the construction of textual use case specifications. *Data Knowl. Eng.* **25**(1), 125–160 (1998)
17. Ochodek, M., Nawrocki, J., Kwarciak, K.: Simplifying effort estimation based on use case points. *Inf. Softw. Technol.* **53**(3), 200–213 (2011)
18. Diev, S.: Software estimation in the maintenance context. *ACM SIGSOFT Softw. Eng. Notes* **31**(2), 1–8 (2006)
19. Ochodek, M., Alchimowicz, B., Jurkiewicz, J., Nawrocki, J.: Improving the reliability of transaction identification in use cases. *Inf. Softw. Technol.* **53**(8), 885–897 (2011)
20. Davis, F.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.* pp. 319–340 (1989)
21. Spearman, C.: "General intelligence," objectively determined and measured. *Am. J. Psychol.* **15**(2), 201–293 (1904)

Software Engineering: Challenges and Solutions
Results of the XVIII KKIO 2016 Software Engineering
Conference 2016 held at September 15-17 2016 in
Wroclaw, Poland

Madeyski, L.; Śmiałek, M.; Hnatkowska, B.; Huzar, Z.
(Eds.)

2017, XII, 215 p. 53 illus., 32 illus. in color., Softcover
ISBN: 978-3-319-43605-0