p-ISSN 1607-3274 Радіоелектроніка, інформатика, управління. 2023. № 2
e-ISSN 2313-688X Radio Electronics, Computer Science, Control. 2023. № 2

ПРОГРЕСИВНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

PROGRESSIVE INFORMATION TECHNOLOGIES

UDC 004.414.38

TECHNOLOGY FOR IDENTIFYING AND FORMING POSSIBLE RELATIONSHIPS BETWEEN USE CASES IN THE PROCESS OF THE INFORMATION SYSTEM DESIGN

Kungurtsev O. B. – PhD, Professor, Professor of the Software Engineering Department, Odessa Polytechnic National University, Odessa, Ukraine.

Zinovatna S. L. – PhD, Associate Professor of the Software Engineering Department, Odessa Polytechnic National University, Odessa, Ukraine.

ABSTRACT

Context. Use cases are widely used as a means of formulating requirements in the development of information systems. All subsequent design stages depend on the quality of their presentation. Structuring use cases can significantly increase their understanding and maintenance in the face of changing requirements..

Objective. Flexible technologies involve working in small teams. The existing communication between teams is not sufficient to highlight sub use cases at the project level. There is a need for automated analysis of the corpus of all use cases.

Method. A mathematical model of a use case which makes it possible to define the criteria for comparing scenarios and eliminate the redundancy of descriptions is proposed. A four-step method for restructuring use cases has been developed. At the first stage, use cases are presented in a formalized form. At the second, they are stored in the repository, which ensures their quick search and placement. At the third stage, procedures of scenario comparison are performed. Scenario similarity criteria are proposed. At the fourth stage, the formation of subordinate use cases is carried out, their texts are coordinated with all interested teams, and the use cases that cause subordinate use cases are corrected.

Results. Experiments providing the formalized compilation of use cases by several development teams followed by automated restructuring were carried out to test the proposed solutions. As a result, new subordinate use cases were correctly identified and the scope of use of previously formed ones was expanded. There was a significant reduction in the time for restructuring.

Conclusions. The proposed method of restructuring use cases improves the clarity and consistency of requirements, the possibility of their adjustment and maintenance, and reduces the compilation time. The method can be used in the design of any information system, where the requirements are presented in the form of use cases.

KEYWORDS: Use Case, Subordinate Use Case, Scenario, Information System Design.

ABBREVIATIONS

UC is a use case;

IS is a information system;

SUC is a subordinate use case.

NOMENCLATURE

Actor is the system user who will perform the said UC step;

cA is a condition of transition to an alternative scenario;

Client is an optional element (introduced when it is necessary to specify who the initiative comes from);

$data_k$ is a data that is input and/or output from the system;

dName is the name of the data;

dType is the type of the data;

eList is a list of template elements of a scenario step;

editText is a texts of scenario step edited by the developer;

ep_j is the number of the main scenario step from which the transition to the alternative scenario takes place;

id is the UC identifier;

mAP is a numbered set of extension scenario steps;

mData is a set of data entered into the system or received from it;

mP is a numbered set of items of the main scenario;

mES is a set of alternative scenarios;

mRef is a set of references of the subordinate UC to the UC that access it;

n_i is a number of alternative scenarios;

nP is the number of the scenario step;

p_i is a step text;

pH is the head step number, possible values: empty string, integer;

pText is a scenario step text;

pType is a scenario step type;

refT is a reference to the command that accompanies the UC;

DOI 10.15588/1607-3274-2023-2-12

rp_k is the number of the main scenario step, to which the return from the alternative scenario takes place;

S_i is a UC scenario;

tp_i is a pre-composed piece of text;

tu_j is a piece of text formulated by the developer;

ucType is a UC type, can take two values: main (for main UCs) or subordinate (for subordinate UCs).

INTRODUCTION

UCs are the main way to represent functional requirements [1], and partially non-functional requirements [2] to the designed IS. The quality of the entire project largely depends on the quality of UC writing. There are a number of recommendations for compiling UCs that relate to general issues of selecting UCs, ways of scenario recording in relation to the tasks solved, and formats for presenting UCs. In [3], the concept of "subordinate UC" (SUC) is introduced to define UC, which is called from some step in the scenario of the main UC. Usually, SUCs are formed from extensions of the main UC. There are at least two reasons for this:

– the extension is used in several places. The formation of a SUC from it will simplify the maintenance of requirements and the code that implements them;

– expansion makes the main UC difficult to understand.

The SUC must be linked to the main UC by an include or extend relationship. Fig. 1 shows examples of such relationships.

Within the framework of a large project, dozens or hundreds of UCs are formed by different development teams. Under such conditions, determining the identity of subordinate UCs selected in various subsystems, and, furthermore, finding repeating fragments of scenarios, is a very complex and time-consuming task. The problem becomes even more difficult in the context of global software development [4].

In this paper, it is proposed to consider the problem associated with the allocation of SUC in a broader sense – the elimination of repetitive requirements and the code corresponding to them.

In a simplified form, the process of UC formation in the design of IS is shown in Fig. 2. At the system analyst level, a list of the main UCs can be generated. If several development teams are working on the project, then the allocation of SUC becomes possible only within some parts of the project.

The purpose of the work is to improve the quality of presentation of functional requirements in the form of use cases by eliminating the redundancy of descriptions and introducing UC structuring.

To achieve the said goal, it is proposed to solve the following tasks:

1) to create a mathematical model of UC that makes it possible to compare fragments of their scenarios;

2) to develop a method for UC restructuring;

3) to test the study results.

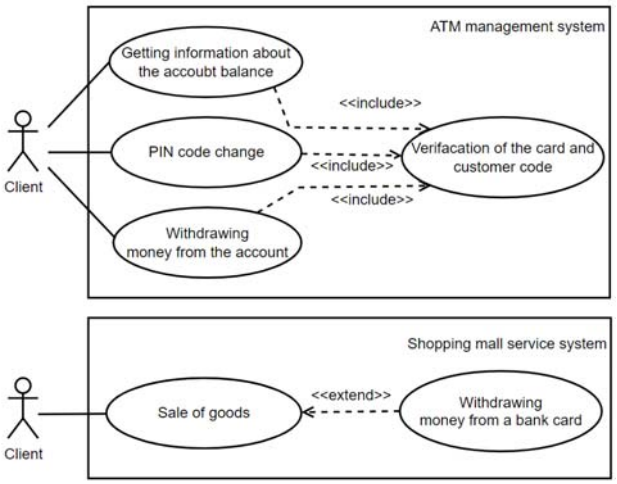

Figure 1 –Relationships between master and subordinate use cases

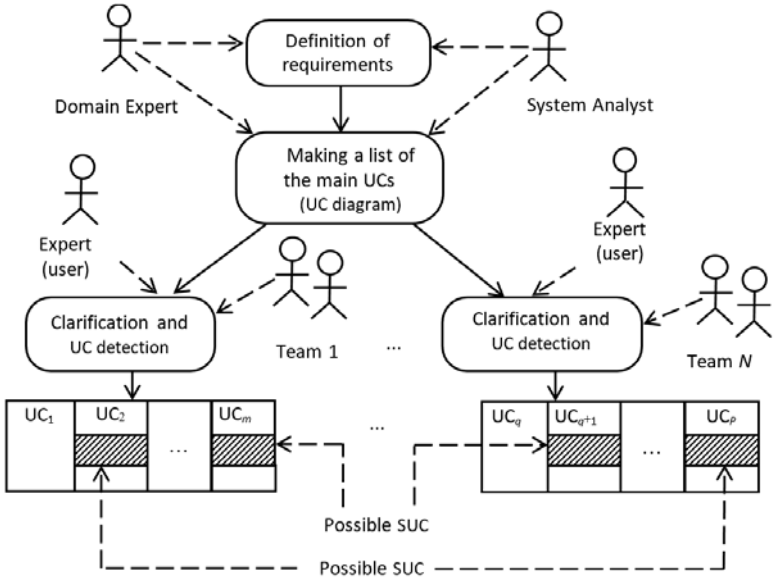

Figure 2 –The process of generating use cases

DOI 10.15588/1607-3274-2023-2-12

1 PROBLEM STATEMENT

Suppose given the set of UC descriptions provided by development teams $mUC=\{UC_1,...,UC_i,..., UC_t\}$, where each element $UC_i=<mUC_i^m, mUC_i^s>$, $i=\overline{1,t}$, consists of mUC_i^m (set of master UC) and mUC_i^s (set of subordinate UC). The problem are create unified set subordinate UCs with a given efficiency $mUC^s=((\bigcup_{i=1}^{t} mUC_i^s)\cup \cup(mUC_{new}^s)), |mRef_j|\geq p, j=\overline{1,|mUC^s|}$, where mUC_{new}^s is formed using the operation of item belonging to the scenario $\in_s$ for all elements of the set $mUC^m=(\bigcup_{i=1}^{t} mUC_i^m)$; create of an updated set of master UC, taking into account new relationships with SUCs $mUC^m \xrightarrow[ref \text{ instead of } text]{} mUC_{new}^m$.

2 REVIEW OF THE LITERATURE

The need to improve the description of UC for their use at various design stages is indicated in [5].

In [6], it is noted that UCs should be the main tool for communication and verification of requirements by the user. However, the authors believe that expanded class diagrams can be a good mechanism for communicating and checking requirements. In our opinion, first, it is necessary to perform structuring of the UC, which in the future will ensure the construction of high-quality class diagrams.Studies carried out within the framework of the energy project [7] have shown the effectiveness of creating a repository for more than 50 UCs. The authors note that this created the conditions for solving a number of tasks of project progress control, documentation management, profitability and safety improvement. In our opinion, the functions of the repository can be extended with the tasks of analysis and UC restructuring.

The authors propose a formal model of the UC diagram followed by a multiview consistency check. However, the authors do not consider the distribution of functions between UCs.

The need for further formalization of requirements was noted in [9]. It is proposed to use a structured natural language and the corresponding FRET tool. In our opinion, formalization should be introduced wherever there are conditions for this, for example, when forming UC. The problem of presenting UCs with varying degrees of detail is considered in [10]. The complexity of this task is noted and a number of recommendations for “slicing” UC in Agile technologies are given. However, the authors do not propose a formal model for the UC refinement process.

In [11], the problem of low-quality specification of requirements is noted. As a solution, it is proposed to use document templates compiled on the basis of the experience of successfully completed projects. We believe that the use of templates will be especially effective in the formation of UC and SUC. Such templates were proposed in [12] as part of building a model of conceptual classes based on an automated description of UC.

In [13], the influence of natural language on the quality of work with requirement specifications is noted and it is recommended to use natural language processing tools. In [14], it is proposed to improve the quality of project documentation by expanding the use of formal methods for its presentation. This problem extends to a large extent to the description of UC. Both natural language processing tools and the use of special templates that reduce the ambiguity of text fragments can be its solution. Models of UC scenario steps [15] can be a solution to this problem.

3 MATERIALS AND METHODS

Selection of SUC requires comparison of UC scenarios. It is possible to organize a comparison of scenarios of all UCs to select matching sequences of steps. However, since UCs are compiled by different developers, the probability of finding matching texts is negligible. We can use fuzzy string comparison [16] by introducing certain matching coefficients. However, a very low signal-to-noise ratio is expected in this case too. To solve the problem, it is proposed to introduce certain formalization in the description of UC, expressed in the following steps.

1. When describing all UCs, use a unified classification of scenario steps [12].
2. When describing all UCs, use a unified system of generalized data typing [15].

Let us represent UC in the form of a tuple:

$$UC=<id, ucType, mP, mES, mRef, refT>. \quad (1)$$

The *mRef* parameter makes it possible to determine the efficiency of using the subordinate UC. For main UC $mRef=\varnothing$. In the main UC links to subordinate UCs are provided by a special scenario clause.

Let us represent each alternative scenario as a tuple:

$$UC=<id, ucType, mP, mES, mRef, refT>, \quad (2)$$

where $mAP=(p_1, p_2, ... , p_k)$.

If $rp_k=0$, then UC ends.

Use case restructuring method provides for the following steps.

Stage 1 – formalized presentation of UC scenarios.

In the representation of the step of the main scenario, extension scenario or SUC, we will indicate its type. The following types of UC steps are proposed in [12]: Create, Enter data, Request a value, Request a list of values, Select from a list, Request a service, Request with a value, Repeat actions, Successful completion of the UC, Failure of the UC, Call of the UC.

For each step type, a model has been compiled that makes it possible to formalize and automate the formation of the step. As an example, the model of the “Value Request” step is given. The user asks the system for some data. This is usually followed by an evaluation of the ob-

DOI 10.15588/1607-3274-2023-2-12

tained data by the user. The step description template looks like this:

$$requestValue=<nP\ [, Client, tp_1, tu_1], Actor, tp_2, tu_2, data_1\ [, tp_3, tu_3, data_2][, tp_4, tu_4, data_3, tp_5], tp_6, data_1[, tp_7]>. \quad (3)$$

For them, the developer specifies the position.

The elements of the template can be pre-composed pieces of text (tp_i), pieces of text formulated by the developer (tu_j), and data that is input and/or output from the system ($data_k$). The $data_k$ element is formed from two components: the name of the data and its type. This information is used only by the developer and is not visible to the user. Square brackets enclose optional template elements. Below are the values of the template elements:

- tp_{one}= "wishes to receive";
- tu_{one} – is formed by the developer, for example, "repair cost ...";
- tp_2= "requests the system";
- tu_2 – is formed by the developer, for example, "repair cost ...";
- $data_{one}$ – data that is requested from the system;
- tp_3= "based on";
- tu_3 – is formed by the developer, for example, "car brands ... ";
- $data_2$ – data on the basis of which the requested value is determined;
- tp_{four}= "subject to";
- tu_{four} – is formed by the developer, for example, "availability of spare parts ... ";
- $data_3$– data on the basis of which the fulfillment of the condition is checked;
- tp_5= "The system confirms the fulfillment of the condition" – an optional element;
- tp_6= "The system outputs";
- tp_7= "Client/Actor agrees".

Service request step template:

$$reqService=<nP\ [, Client, tp_1, tu_1], Actor, tp_3, tu_2, \{data_1\}, tp_4\ [, tp_5, tu_3]>,$$

where tp = "wishes"; tu_1 is a text that identifies the service (e.g. "undercarriage overview") or document (e.g. "application for a reduced rate"); tp_3="enters"; tu_2 – a phrase that is formed by the user, the name of the service or document; $data_1$ – service or document representation in the project; tp_4= "The system confirms the possibility of performing the service (document) "; tp_5= "Transfer of the control to scenario step"; tu_3– scenario step number.

To formalize the representation of input and output data, the following set of generic types is proposed:

- *List* – list (can represent a linear list, an array, a set, etc.);
- *Struct* – the structure (in the general case it contains fields of different types), must contain the numbering of the fields;
- *Text* – any text;
- *Numb* – any number format;
- *Bool* – boolean value;
- *Void* – the function does not return the value;
- *PClass* – a reference to a class object;

From the point of view of SUC selection, the step of the type "Request for a service" has a special meaning. It may be followed by steps (subordinate), revealing the mechanism for providing the service. It is this sequence of stepss that can be a candidate for a SUC formation. To formalize the semantic relationship between steps, it is proposed to introduce link indicators in the form of step numbers into the texts of steps belonging to the "Service Request" group.

Example 1. A scenario fragment that implements registration will look like this:

…….

N. [0] The client wishes to register in the system. The system confirms the possibility of registration.

N+1. [N] The system displays the registration conditions. The client agrees.

N+2. [N] The system suggests entering an email. The client enters. The system confirms the completion of the registration.

Taking into account the considered types of scenario steps and data, we will represent the scenario step in the form:

$$p=<nP, pH, pType, pText, mData>. \quad (4)$$

In accordance with the accepted classification, a set can contain up to three data.

Each data has the form

$$mData_i=<dName, dType>. \quad (5)$$

Stage 2 – Placement of the UC in the repository.

To simplify the performance of operations with UC (storage, structuring, tracking changes), a repository is created.

Using queries to the repository database (Fig. 3), it is possible to organize the set of all UCs, divided into subsets depending on the commands that work with individual UCs. The common part containing SUC is also selected.

Stage 3 – Comparison of UC scenarios.

Selection of subordinate UCs can be performed at the level of the development team and at the project level.

At the level of the development team, it is possible to select subordinate UCs from the set of UCs that this team is engaged in. We call such a SUC local. Verification and approval of such a selection should occur within the team itself.

At the project level, it is necessary to determine the possibility of using the local SUC of a certain team for UCs developed in other teams. At the same time, the analysis of the possibility of using the SUC should be determined by the team that is proposed to use the SUC. In addition, considering the entire corpus of UCs, it is necessary to ensure the possibility of identifying SUCs that have not previously been created at the level of individual teams.

DOI 10.15588/1607-3274-2023-2-12

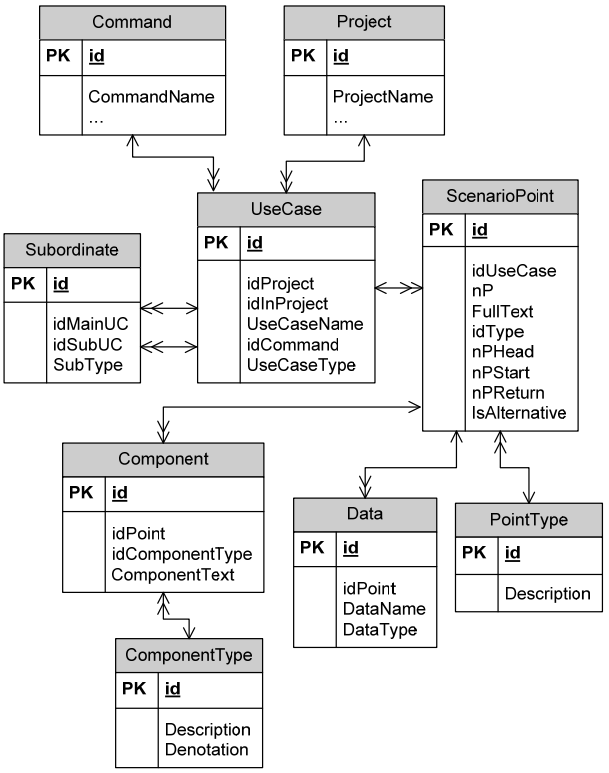

Figure 3 – Repository data model

If a team has selected a SUC within their part of the project and it turns out that the SUC can be used in other parts, then the respective teams must confirm its use in their parts. If a new SUC is selected, then all teams where it will be used must confirm the possibility of its application within their part of the project. Each SUC must have a link to the team (developer). The main operation of the process is finding the occurrence of one scenario into another. Let us determine the main options for comparison:

– the identity of two subordinate UCs is established when the conditions for their call coincide and their sequences of steps coincide;

– a subordinate UC can be selected from two main UCs if a certain common sequence of steps in the scenarios of two main UCs with a length of at least 2 points is determined;

– the entry of a subordinate UC into the UC is fixed if all the points of the subordinate UC coincide with a part of the sequence of UC steps.

A group of semantically related steps should not be split into parts when compared. However, the order of substeps in a group can be arbitrary.

Example 2: A scenario fragment that implements registration in a sequence different from the one in Example 1:

…….

N. [0] The client wishes to register in the system. The system confirms the possibility of registration.

N+1. [N] The system suggests entering an email. The client enters. The system confirms its correctness.

N+2. [N] The system displays the registration conditions. The client agrees. The system confirms the completion of the registration.

Let us formulate the conditions for the coincidence of two steps from different scenarios.

1. Step types p_i and p_j must match ($pType_i = pType_j$).

2. The actual values of Client and Actor are not compared (the same use case can be performed by different executors in different subsystems of the same project).

3. It follows from the scenario step model (3) that the text of a step of a certain type can have different spellings due to optional elements and elements formed by the developer. Since these elements are important for the specific implementation of the steps, the necessary condition for the steps to match is the identity of their structures.

4. Any step of the scenario, except for the points of repeating actions and calling the SUC, provides for the performance of certain operations in the system of the form: creating an object, entering or receiving data, possibly, if certain conditions are met. Therefore, it is necessary to compare all text fragments formulated by the developer (tu_j).

5. The data that is input, output or created within the framework of the scenario step, in accordance with the template, must have a name and type. Both of these parameters are subject to comparison.

To determine the coincidence of two scenario items, it may be necessary that some elements are identical (we denote this operation as ≡) and incomplete or fuzzy (we denote this operation as ≅). The result of a fuzzy match is the value of the similarity coefficient K. In what follows, we will consider the elements similar if their similarity coefficient is not less than a certain threshold value ($K \geq K_{min}$). Thus, we obtain the condition for the coincidence of points p_i and p_j, which belong to scenarios S_1 and S_2, respectively. Here $\in_s$ denotes the operation of an item belonging to a UC scenario.

$$(p_i \in_s S_1) = (p_j \in_s S_2) \text{ if } ((pType_i = pType_j) \Lambda\ (eList_i \equiv eList_j)\ \Lambda\ (editText_i \cong editText_j) \Lambda\ \exists\ ((dType_{i,k} = dType_{j,k})\, k=1,n)). \quad (6)$$

One step of the main scenario, possibly an alternative one, can have several extension scenarios. In order for the compared items to match, their alternative scenarios must also match. The number of alternative scenarios for the compared steps must match, but the order in which they are written can be arbitrary.

Let us formulate the conditions for the coincidence of semantic groups of steps. If a spep p_i is found for which pH="[0]", then an ordered set of steps of the subordinate group g_i=(p_k | $p_k \cdot pH$="[i]")should be formed.

If a step p_j is found in some other UC, such that $p_j \cong p_i\ \Lambda\ p_j \cdot pH$="[0]", then a subordinate group g_j=(p_q | $p_q \cdot pH$="[j]") is created for it.

To forma SUC based on steps p_i and p_j, it is necessary for the number of elements in the groups to be the same |

DOI 10.15588/1607-3274-2023-2-12

g_i | = | g_j | = n and for each step from the set g_i, a matching step was found in the set g_j $p_k \cong p_q$; k=1, n; q=1, n.

Let us formulate the conditions for the coincidence of extensions for points p_i and p_j with n_i and n_j alternative scenarios, respectively. First of all, the conditions for switching to an alternative scenario must match:

$$\exists cA_{i,p}: ((cA_{i,p} \cong cA_{j,k}), p=1, n_i; k=1, n_j)) \Lambda (n_i = n_j). \quad (7)$$

Further, in accordance with (6), the steps of alternative scenarios are compared in pairs, for which condition (7) is satisfied.

If condition (6) is met the first time for scenarios S_1 and S_2, then a new scenario $S_{1,2}$ is created, the first step of which is $pS12_1=p_i$.

If condition (6) is also satisfied for the next pair of steps $p_{i+1} \equiv p_{j+1}$, then step p_{i+1} is added to scenario $S_{1,2}$ and the process of scenario formation continues. Otherwise, the scenario $S_{1,2}$ is destroyed (there is only 1 step in the scenario).

It is proposed to evaluate the degree of coincidence $K_{i,j}$ of two scenarios S_1 and S_2 as an average coefficient of similarity of the steps included into them:

$$K_{i,j} = \frac{\sum k}{n}. \quad (8)$$

Stage 4 – Selection of SUC and UC restructuring.

The execution of the stage involves the following sequence of actions.

1. For each local SUC, its comparison with other local SUC is performed. In case of a match, the SUC is defined as global and a link to the support command is set in it. Local links are replaced with a global one.

2. For each global SUC, the possibility of its inclusion in the UC scenarios is determined. If possible, the UC scenario is edited accordingly. A link to the SUC is set in it.

3. For each UC, a comparison with other UCs (of different localization) is performed. If common parts are selected, then a global SUC is formed, a link to the support command is set in it, scenarios and links in the UC that have a common fragment are edited.

All operations for selecting a new UC or expanding the scope of its use are coordinated with the developer teams, which must introduce changes into the UC descriptions.

4 EXPERIMENTS

To carry out the experiments, a document "Vision" for the development of an information system on the topic "Automation of the work of a clinic", a list of users of the designed system and a list of UCs of 16 names were compiled. The developers were represented by 4 teams of 2 people. Each group received tasks to form 4 UCs in the UseCaseEditor. The groups were asked, if possible, to form a SUC in addition to the UC.

To test the results of the study, a software product that makes it possible to select SUC on the basis of the entire UC corpus in accordance with the developed methodology was developed.

5 RESULTS

The results were introduced into Table 1 after the discussion with all participants of the experiment. Symbols for UC and SUC were introduced in the table. For example, SUC (1) indicates that it was selected from UC 1 and is not used anywhere else. SUC (1–10–15)s indicates that in terms of content it is SUC (1)s, however, it was found out that it is a part of UC 10 and UC 15. A record of the form (2–6–10*)s means that SUC can be used for UC 2 and UC 6, but its use for UC 10 is a mistake.

The analysis of table data shows that at K_{min}=0.5 the best results were obtained: the scope of SUC 1 and 7 was expanded by three UCs, and 2 new UCs were found.

During the experiments, the time spent on compiling UC and SUC was estimated. On average, 3.5 hours were spent on compiling 1 UC. It took 1.5 hours to select and compile one SUC, as well as adjust the UC within one team. The same work, but with unfamiliar UCs (4 UCs from another team) took 3.7 hours. The calculation of the time spent for the given example in the "manual" search for SUC increased the total time of UC formation by 58%.

6 DISCUSSION

Automation of SUC selection became possible due to the use of UC step models. Further formalization of the UC defination, for example, by using a formalized natural language, is undesirable, since it will create inconvenience for the developer. It follows from the experiment results that the quality of SUC selection significantly depends on the value of the similarity coefficient K_{min}. There is no guarantee that K_{min}=0.5 value will always be the best. The solution could be to use a domain dictionary to define an additional semantic relationship between compared texts, and as minimum, to use synonyms.

Table 1 – The fragment of experimental results on model building by the formed samples

Developer teams			1	2	3	4
UC			1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16
Selection of SUC	Manual mode		(1)s	(7)s	(9–11)s	
	Auto mode	K_{min}=0.2	(1–10–15)s, (2–6–10*)s	(7–13)s, (5–9–16*)s	(12–14)s	(13–15*)s
		K_{min}=0.5	(1–10–15)s, (2–6)s	(7–13)s, (5–9)s	(12–14)	
		K_{min}=0.8	(1–15)			

DOI 10.15588/1607-3274-2023-2-12

The effectiveness of the proposed method of UC restructuring depends on the specific subject area. In the conducted experiment, the tasks for the development of UC were selected taking into account the possibility of selecting SUC. In real conditions, it can be expected that the proportion of SUC in the UC corpus will be 2–3 times lower [3]. This will reduce the time to search for the use of SUC in UC, but not the selection of new SUCs. Therefore, in this case, we can expect a reduction in the time for restructuring by about 30% – 40% as well.

CONCLUSIONS

The analysis of existing technologies for compiling UC was carried out. It was established that working in small teams on projects of medium and high complexity does not allow presenting the UC corpus in a well-structured form.

A mathematical model of the use case characterized by the introduction of the concept of the UC type, references to other UCs and the development team was proposed, which made it possible to further organize the process of comparing the UC and selecting the UC.

For the first time, a method of automated UC restructuring which allows comparing UC scenarios, selecting SUC, correcting the links between UC and SUC was developed. Application of the method makes it possible to improve the structure of the UC corpus, which increases the degree of understanding of the requirements, reduces the time and errors for maintaining requirements due to the elimination of duplication.

The experiments conducted showed the selection of all repeating fragments of scenarios, the correct selection of the SUC and a significant reduction in the time for UC restructuring (about 35%).

The proposed method can be used in any IS project where the functional requirements are presented in the form of UC.

REFERENCES

1. Wazlawick R. S. Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML. San Francisco, Morgan Kaufman, 2014, 376 p.
2. Nilsen A. F., Muller G. Use Cases and Non-functional Requirements Presented in Compact System Description A3s, *INCOSE International Symposium*, 2014, Vol. 24, Issue 1, pp. 1–15. DOI: 10.1002/j.2334-5837.2014.tb03130.x
3. Cockburn A. Writing Effective Use Cases. Addison-Wesley, 2001, 270 p.
4. Mighetti J. P., Hadad G. D. S. A Requirements Engineering Process Adapted to Global Software Development, *CLEI Electronic Journal*, 2016, Vol. 19, Issue 3, pp. 1–21. DOI: 10.19153/cleiej.19.3.7
5. Russell M. Supporting Decision Makers with Use Cases; case study result, Procedia Computer Science, 2019, Vol. 153, P. 294–300. DOI: 10.1016/j.procs.2019.05.082
6. Dobing B., Parsons J. Understanding the Role of Use Cases in UML: A ReUCew and Research Agenda, *Journal of Database Management*, 2000, Vol. 11, Issue 4, pp. 28–36. DOI: 10.4018/978-1-931777-12-4.ch008
7. Clausen M., Apel R., Dorchain M. Use case methodology: a progress report, *Energy Informatics,* 2018, Vol. 1, pp. 274–283. DOI: 10.1155/2018/6854920
8. El Miloudi K., Ettouhami A. A Multiview Formal Model of Use Case Diagrams Using Z Notation: Towards Improving Functional Requirements Quality, *Journal of Engineering*, 2018, Vol. 2018, pp. 1–9.
9. Giannakopoulou D., Pressburger T., Mavridou A. et al. Automated formalization of structured natural language requirements, *Information and Software Technology,* 2021, Vol. 137, pp. 106590. DOI: 10.1016/j.infsof.2021.106590
10. Linders B. Applying Use Cases in Agile: Use Case 2.0, Slicing and Laminating [Electronic resource]. Access mode: https://www.infoq.com/news/2014/02/use-cases-agile
11. Barcelos L. V., Penteado R. D. Elaboration of software requirements documents by means of patterns instantiation, *Journal of Software Engineering Research and Development*, 2017, Vol. 5, pp. 3.1–3.23. DOI: 10.1186/s40411-017-0038-9
12. Kungurtsev O., Novikova N., Reshetnyak M. et al.Method for defining conceptual classes in the description of use cases, *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019*: Wilga, 25 May – 2 June 2019, Proceedings. SPIE, 2019, Vol. 1117624. DOI: 10.1117/12.2537070
13. Ahmed H., Hussain A., Baharom F. The Role of Natural Language Processing in Requirement Engineering, *International Journal of Engineering & Technology*, 2018, Vol. 7, Issue 4.19, pp. 168–171. DOI: 10.14419/ijet.v7i4.19.22041
14. Shah U. S., Jinwala D. C. Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey, *ACM SIGSOFT Software Engineering Notes*, 2015, Vol. 40, Issue 5, pp. 1–7. DOI: 10.1145/2815021.2815032
15. Kungurtsev O. B., Novikova N. O., Zinovatna S. L. et al. Automated object-oriented for software module development, *Applied Aspects of Information Technology,* 2021, Vol. 4, Issue 4, pp. 338–353. DOI: 10.15276/aait.04.2021.4
16. Kalyanathaya K. P., Akila D., Suseendren G. A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP, *Journal of Computational Information Systems*, 2019, Vol. 15, No. 3, pp. 26–32.

Received 13.03.2023.
Accepted 03.05.2023.

УДК 004.414.38

ТЕХНОЛОГІЯ ВИЯВЛЕННЯ Й ФОРМУВАННЯ МОЖЛИВИХ ВІДНОШЕНЬ МІЖ ВАРІАНТАМИ ВИКОРИСТАННЯ В ПРОЦЕСІ ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Кунгурцев О. Б. – канд. техн. наук, професор кафедри Інженерії програмного забезпечення Національного університету "Одеська політехніка", м. Одеса, Україна.

Зіноватна С. Л. – канд. техн. наук, доцент кафедри Інженерії програмного забезпечення Національного університету "Одеська політехніка", м. Одеса, Україна.

DOI 10.15588/1607-3274-2023-2-12

АНОТАЦІЯ

Актуальність. Варіанти використання широко використаються як засіб формулювання вимог при розробці інформаційних систем. Від якості їхнього представлення залежать всі наступні етапи проектування. Структуризація варіантів використання дозволяє істотно підвищити їхнє розуміння й супровід в умовах мінливих вимог. Гнучкі технології передбачають роботу в невеликих командах. Існуючий обмін інформацією між командами недостатній для виділення підлеглих варіантів використання на рівні проекту. Існує необхідність автоматизованого аналізу корпусу всіх варіантів використання. Метою дослідження є підвищення якості представлення функціональних вимог у вигляді варіантів використання шляхом усунення надмірності описів і введення структуризації варіантів використання на рівні всього проекту.

Метод. Запропонована математична модель варіанта використання, яка дозволяє визначити критерії для порівняння сценаріїв. Розроблено метод реструктуризації варіантів використання, який включає чотири етапи. На першому етапі варіанти використання представляються у формалізованому вигляді. На другому – вони зберігаються в репозиторії, що забезпечує їхній швидкий пошук і розміщення. На третьому – виконуються процедури порівняння сценаріїв. Запропоновано критерії подоби сценаріїв. На четвертому – виконується формування підлеглих варіантів використання, узгодження їхніх текстів із із усіма зацікавленими командами, коректування варіантів використання, які викликають підлеглі варіанти використання.

Результати. Для апробації запропонованих рішень проведені експерименти, які передбачають формалізоване складання варіантів використання декількома групами розроблювачів з наступною автоматизованою реструктуризацією. У результаті були коректно виявлені нові підлеглі варіанти використання й розширена область використання раніше сформованих. Спостерігалося істотне скорочення часу на реструктуризацію. Очікуване скорочення часу на реструктуризацію для реального проекту складе близько 35%.

Висновки. Запропонований метод реструктуризації варіантів використання дозволяє поліпшити дохідливість і погодженість вимог, можливість їхнього коректування й супроводу, скоротити час на складання. Метод може бути використаний при проектуванні будь-якої інформаційної системи, де вимоги представляються у вигляді варіантів використання.

КЛЮЧОВІ СЛОВА: варіант використання, підлеглий варіант використання, сценарій.

ЛІТЕРАТУРА / ЛИТЕРАТУРА

1. Wazlawick R. S. Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML / R. S. Wazlawick. – San Francisco : Morgan Kaufman, 2014. – 376 p.
2. Nilsen A. F. Use Cases and Non-functional Requirements Presented in Compact System Description A3s / A. F. Nilsen, G. Muller. // INCOSE International Symposium. – 2014. – Vol. 24. Issue 1. – P. 1–15. DOI: 10.1002/j.2334-5837.2014.tb03130.x
3. Cockburn A. Writing Effective Use Cases / A. Cockburn. – Addison-Wesley, 2001. – 270 p.
4. Mighetti J. P. A Requirements Engineering Process Adapted to Global Software Development / J. P. Mighetti, G. D. S. Hadad // CLEI Electronic Journal. – 2016. – Vol. 19, Issue 3. – P. 1–21. DOI: 10.19153/cleiej.19.3.7
5. Russell M. Supporting Decision Makers with Use Cases; case study result / M. Russell // Procedia Computer Science. – 2019. – Vol. 153. – P. 294–300. DOI: 10.1016/j.procs.2019.05.082
6. Dobing B. Understanding the Role of Use Cases in UML: A ReUCew and Research Agenda / B. Dobing, J. Parsons // Journal of Database Management. – 2000. – Vol. 11, Issue 4. – P. 28–36. DOI: 10.4018/978-1-931777-12-4.ch008
7. Clausen M. Use case methodology: a progress report / M. Clausen, R. Apel, M. Dorchain // Energy Informatics. – 2018. – Vol. 1. – P. 274–283. DOI: 10.1155/2018/6854920
8. El Miloudi K. A Multiview Formal Model of Use Case Diagrams Using Z Notation: Towards Improving Functional Requirements Quality / K. El Miloudi, A. Ettouhami // Journal of Engineering. – 2018. – Vol. 2018. – P. 1–9.
9. Automated formalization of structured natural language requirements / [D. Giannakopoulou, T. Pressburger, A. Mavridou et al.] // Information and Software Technology. – 2021. – Vol. 137. – P. 106590. DOI: 10.1016/j.infsof.2021.106590
10. Linders B. Applying Use Cases in Agile: Use Case 2.0, Slicing and Laminating [Electronic resource] / B. Linders. – Access mode: https://www.infoq.com/news/2014/02/use-cases-agile
11. Barcelos L. V. Elaboration of software requirements documents by means of patterns instantiation / L. V. Barcelos, R. D. Penteado // Journal of Software Engineering Research and Development. – 2017. – Vol. 5. – P. 3.1–3.23. DOI: 10.1186/s40411-017-0038-9
12. Method for defining conceptual classes in the description of use cases / [O. Kungurtsev, N. Novikova, M. Reshetnyak et al.] // Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019: Wilga, 25 May – 2 June 2019 : Proceedings. – SPIE, 2019. – Vol. 1117624. DOI: 10.1117/12.2537070
13. Ahmed H. The Role of Natural Language Processing in Requirement Engineering / H. Ahmed, A. Hussain, F. Baharom // International Journal of Engineering & Technology. – 2018. – Vol. 7, Issue 4.19. – P. 168–171. DOI: 10.14419/ijet.v7i4.19.22041
14. Shah U. S. Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey / U. S. Shah, D. C. Jinwala // ACM SIGSOFT Software Engineering Notes. – 2015. – Vol. 40, Issue 5. – P. 1–7. DOI: 10.1145/2815021.2815032
15. Automated object-oriented for software module development / [O. B. Kungurtsev, N. O. Novikova, S. L. Zinovatna et al.] // Applied Aspects of Information Technology. – 2021. – Vol. 4, Issue 4. – P. 338–353. DOI: 10.15276/aait.04.2021.4
16. Kalyanathaya K. P. A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP / K. P. Kalyanathaya, D. Akila, G. Suseendren // Journal of Computational Information Systems. – 2019. – Vol. 15, No.3. – P. 26–32.

DOI 10.15588/1607-3274-2023-2-12