Automatic Transactions Identification in Use Cases*

Mirosław Ochodek and Jerzy Nawrocki

Poznań University of Technology, Institute of Computing Science,
ul. Piotrowo 3A, 60-965 Poznań, Poland
{Miroslaw.Ochodek,Jerzy.Nawrocki}@cs.put.poznan.pl

Abstract. Since the early 90's of the previous century, use cases have became informal industry standard for presenting functional requirements. The rapid popularity growth stimulated many different approaches for their presentation and writing styles. Unfortunately, this variability makes automatic processing of use cases very difficult. This problem might be mitigated by the use of transaction concept, which is defined as an atomic part of the use case scenario. In this paper we present approach to the automatic transaction discovery in the textual use cases, through the NLP analysis. The proposed solution was implemented as a prototype tool UCTD and preliminarily verified in a case study.

Keywords: Use cases, Use-cases transactions, Use Case Points, Requirements engineering, Effort estimation, Natural language processing.

1 Introduction

In the field of requirements specification there are two extremes. One is a formal approach based on sound mathematical background. People interested in this approach can use such notations as VDM, Z [17], Statecharts [16], Petri Nets [24] and many others. The problem is that those notations lead to quite long specifications (sometimes their size is comparable to code size) and proving correctness of commercial-like programs can be quite time consuming (an interesting case study has been described by Wolfgang Reif [23,13]). But what is more important, mathematically sound specifications usually are unreadable for a typical end-user.

Another extreme is completely unformal approach based on the oral communication. The best example of that approach are user stories, which are one of the main practices of Extreme Programming [6]. Here the main weakness is dependence on human memory. In case of difficult requirements with many competing approaches, each one having indirect impact, relying on human memory can be dangerous.

* This research has been financially supported by the Ministry of Science and Higher Education grant N516 001 31/0269.

B. Meyer, J.R. Nawrocki, and B. Walter (Eds.): CEE-SET 2007, LNCS 5082, pp. 55–68, 2008.

Somewhere in between there are use cases. They have been introduced by Ivar Jacobson [19] and further developed by Cockburn [9] and others [1]. A typical use case describes a user-valued transaction in a sequence of steps, and each step is expressed in a natural language (if needed, one can extend a given step with an alternative behaviour). That makes use cases readable for end-users.

Use cases can be used not only for the system's behaviour specification, but also for the effort estimation. Kerner proposed so-called Use Case Points method (UCP) [20] resembling Function Points [2]. In the UCP size of each use case depends on the number of transactions embedded in it. Unfortunately, the notion of use-case transaction is vague. Due to this, development of sizing tools for use-cases is almost impossible.

In the paper the term of use-case transaction is formalised and a method of automatic identification of transactions in use cases is presented. Transaction identification can be used as a bases for effort estimation and for automatic use-case review.

The paper is organised as follows. In the next section, the rationale for transaction counting is presented as well as existing work concerning their automatic discovery. In Section 4, a formal definition of the use case transaction is introduced, together with an approach for their automatic extraction from the textual use case. The case study for preliminary verification of the model and method is presented in Section 5. Some remarks and lessons learned concerning use case writing style for transaction identification, are presented in Section 6.

2 Use-Case Transactions Counting Problem

The lack of formal standards for the use cases presentation causes many problems with the developing generic and format independent tools and methods. Thus, it is very important to find a common denominator, which might be derived from the use cases despite the differences in the presentation format.

The use case transaction is a decent example of such a concept. It is strictly connected with the use case logic, while dependancy on its notation is limited. It has been introduced by Ivar Jacobson [18]. According to the Jacobson each step should be a transaction, with the four types of phrases included:

- the main actor sends request and data to the system,
- the system validates given data,
- the system performs internal state change operations,
- the system responds to the actor with the operation result.

Transactions are mainly used as a complexity measurement within the Use Case Points (UCP) method [20]. The UCP is an accepted software size metric [4], which might be also used for the effort estimation. Each use case is classified into one complexity class depending on the number of transactions. Use case with three or less transactions is considered as a simple, from four to seven as an average and finally if it has more then seven transactions, it is classified as a complex one. The next step of the UCP method, is a computation of the UUCW (*Unadjusted Use*

Case Weight), which is the sum of simple use cases set cardinality multiplied by 5, cardinality of average set multiplied by 10 and cardinality of complex set by 15.

A transition between size and effort is done by multiplying the UCP method output by the Productivity Factor (PF), which defines how many man hours are required to cover one Use Case Point. The easiest way to obtain the PF within the certain organisation is to compute it from the historical data or use values presented in the literature (according to authors writing about the UCP, PF varies from 20 to 36 h/UCP [20,27]). The Productivity Factor makes UCP method easy to calibrate, as long as historical data comes from the similar projects and rules used for the counting transactions number and assessing other method components does not differ significantly between the considered projects (*experts should be consistent in their approach to the transactions counting*).

According to some authors [5,8,25] use case transactions might be counted as the number of steps in the use case scenarios (main and alternative). This approach is acceptable as long as the use cases are written according to the Jacobson's one step - one transaction rule. In other case relying on counting steps as transactions, might trigger overestimation problems.

Alistair Cockburn in [9] presented relevant example that the same scenario might be easily written with the different number of steps. Each of the four presented use cases consists of the one transaction but the number of steps varies from the one to five (Cockburn's example, is presented in the figure 1). If we follow the UCP classification rules for the use cases complexity classification, two presented examples would be classified as a simple and the other two as an average.

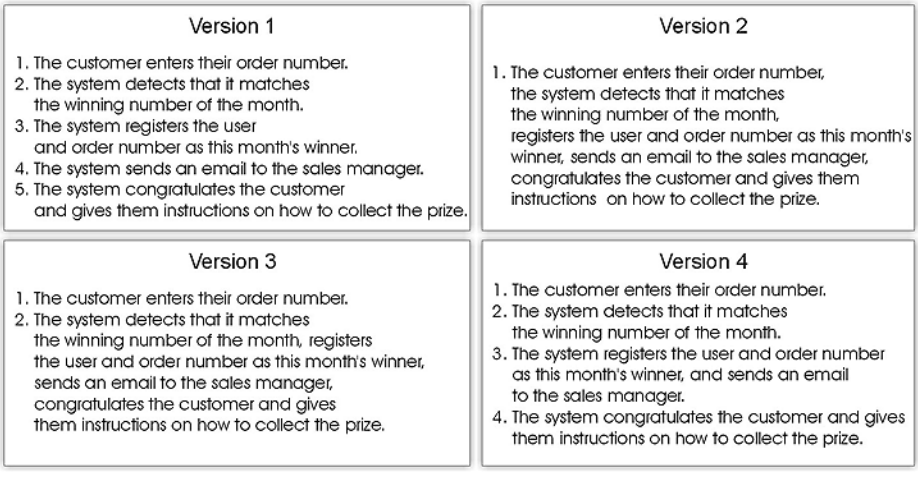


Fig. 1. The example [9] of four versions of the one-transaction use case with scenario varying from 1 to 5 steps. Relying on scenarios length (measured in steps) while counting transactions number may lead to corrupted results.

In the case study presented in the section 5, the UUCW counted for thirty use cases based on the number of steps is 2 times higher then mean UUCW value counted for all experts based on their transactions number assessment.

The important problem concerning transactions identification appears as a difference in formats and styles of writing use cases, which involves other views on the actors actions. The list of some use case structure problems, in the transaction discovery context, is presented in the section 6.

3 Definition of Use-Case Transaction

We have already stated that measuring use case complexity based on the transactions number, seems to be better idea then counting steps in scenarios. However, the notion of transaction should be formally defined in order to provide clear rules for their automatic identification.

The approach to transaction identification for use cases written in Japanese was described by Kusumoto et al. [22]. Authors proposed a system (U-EST), which processed requirements stored in the XMI format and estimating effort based on the UCP method. It is stated that morphological analysis for all statement was conducted in order to find transactions. Each transaction was concerned as a set of subject and predicate that relates to actor's operation and system response. Unfortunately, it is not clear for us, whether authors perceived the pair of actor and system phrases as a transaction or each of them as separate one. The U-EST system was evaluated using data provided by experienced engineers from the Hitachi company. In the case study classification given by the system was compared only with one specialist (authors did not provide exact number of identified transactions, but the final use case classification with the UCP method). The assumption was made that use case grammar is "absurdly simple". Based on our experience, it seems that use cases language, especially in the commerce requirements specifications, is not always as simple as it is suggested in the literature.

Another redefinition of the use case transaction was presented by Sergey Diev [11]. The transaction idea remains similar to the Jacobson's, but it is more general. Author defines transaction as the smallest unit of activity that is meaningful from the actor's point of view. What is more important, use case transaction should be self-contained and leaves the business of the application in a consistent state.

3.1 Proposed Transaction Model

We would like to propose a transaction model which is based on the Ivar Jacobson's transaction definition [18].

We enumerates four types of actions, which are relevant from the use case transaction point of view:

- actor's request action (U),
- system data validation action (SV),

- system expletive actions (e.g. system internal state change action), which are neither validation nor confirmation actions (SE),
- system response action (SR).

We perceive transaction as an atomic sequence of activities (actions) performed by actor and system, which is performed entirely or not at all. Each action belongs to one of the four sets U, SR, SV, SE.

Definition 1. The certain transaction is a shortest sequence of actor's and system actions, which starts from the actor's request (U) and finishes with the system response (SR). The system validation (SV) and system expletive (SE) actions may optionally occur within the starting and ending action. The pattern for the certain transaction written as a regular expression:

$$T_{certain} = \text{U+ [SV SE]* SR+}$$

Where

- *U - actor's request action*,
- *SV - system data validation action*,
- *SE - system expletive action*,
- *SR - system response action*.

Definition 2. We mark transaction as an uncertain (but still as transaction) if it starts from the actor's action (U), but it lacks corresponding system response phrase (SR). In this case system expletive action (SE) might be treated as a transaction closure. It is possible only if the next action is the actor's action (U) or the end of scenario. Thus, uncertain transaction might be defined as:

$$T_{uncertain} = \text{(U+ SV* SE+) (?![SV SR])}$$

Where

- *U - actor's request action*,
- *SV - system data validation action*,
- *SE - system expletive action*,
- *SR - system response action*.

Alternative scenarios (extensions) are also included in the transaction discovery process. The role of extensions scenario depends on the type of corresponding action in the main scenario. Extensions which are regarding system actions, are part of the transaction started in the main scenario. If the extension is triggered by the U-type action, the user decision point is reached. This means that main scenario is being forked into two or more alternative paths. To remark decision point, expression in assertion form might be added before action, in order to express actor's intention (*e.g. User wants to add article*). If the corresponding extension also starts with the assertion (*e.g. User wants to add news*), the alternative execution paths are clearly defined. If the actions in such alternative scenario matches patterns presented in definitions 1 or 2, a new transaction is marked.

Presented certain transaction definition should satisfy both conditions defined by Sergey Diev [11]. The actor makes a request (U), which implies that action is meaningful from his point of view. System responds to the actor's request (SR), which means that the business of the application is left in a consistent state.

Based on the transaction definition, we would like to propose a graphical representation of transaction, which is the Transaction Tree. The root element represents a transaction itself. The next two layers states for the types of actions, which are named here as parts. Each part groups corresponding types of phrases. The action is represented in the tree as:

- <Actor> is the actor's name with distinction to actor(real) and system,
- <PredicateSyn> represents set of action's predicate synonyms (for example show, present etc. would be grouped together),
- <Object> is the object phrase.

The Transaction Tree might also include prolog (*eg. Use case starts when actor enters the page; System displays welcome page etc.*) or epilog phrases (*Use case ends etc*).

4 Transaction Identification with NLP Tools

We use NLP (*Natural Language Processing*) techniques to extract the Transaction Tree and mark transactions occurrence in the use case text. The most important stage of this process is detection of actor's and system actions.

Transactions Discovery Process. The transactions identification rules was developed based on thirty one different use cases coming from the literature [1,9]

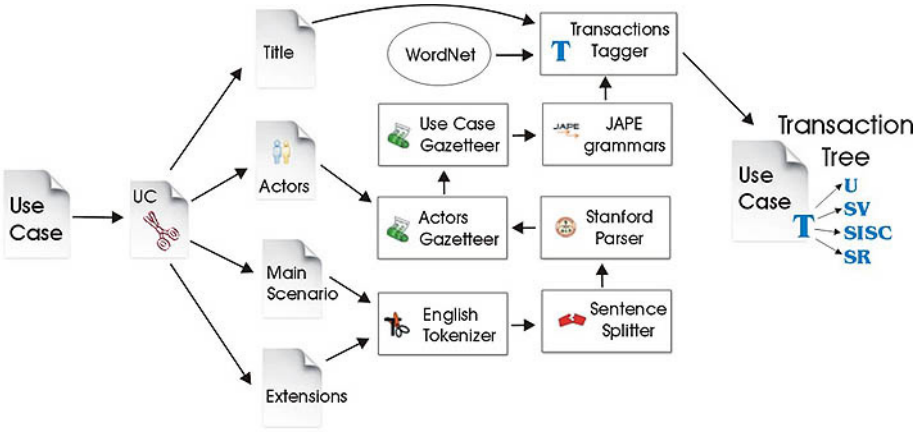


Fig. 2. The UCTD application processing pipeline. Textual use case is split into title, main scenario and extensions documents. Actors names are extracted into the GATE Gazetteer form. The main scenario and extensions part are processed in order to find actions. Finally, the Transaction Tagger performs construction of the Transaction Tree.

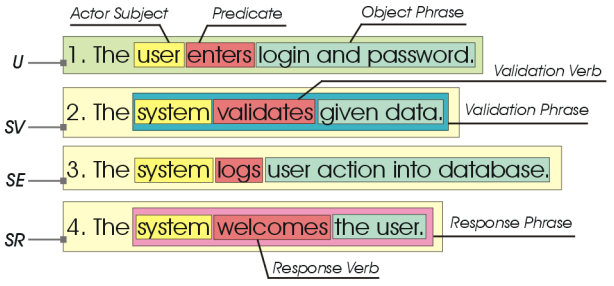


Fig. 3. Example of the use case text with annotations. The most important annotations, like actor subject, predicate, object phrase, validation verb and response verb are marked. According to the defined rules, validations and response phrases are extracted from the text. Finally, each phrase is classified into the part U, SV, SE or SR.

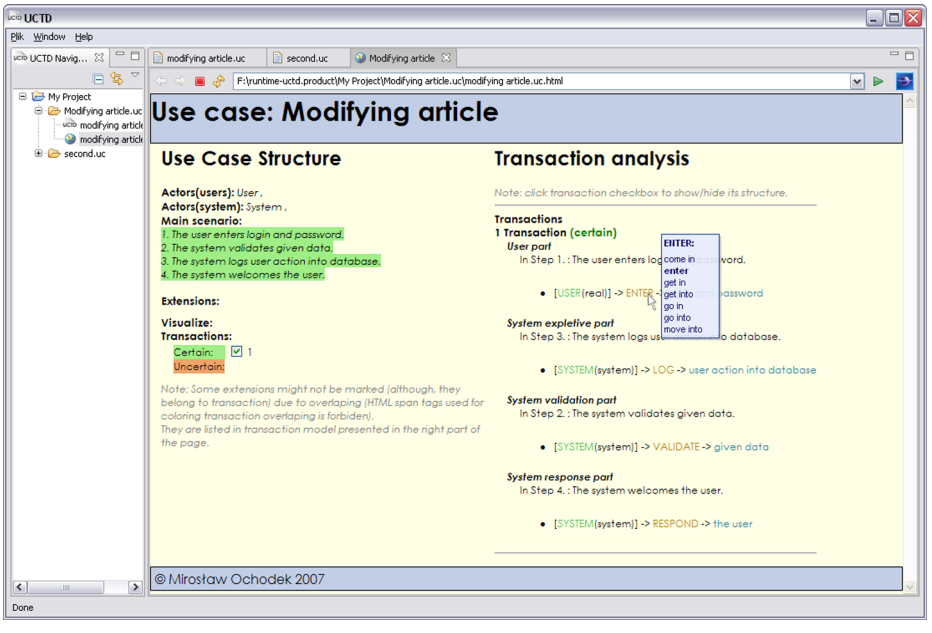


Fig. 4. The UCTD application screen (HTML report). On the left side use case is presented (chosen transaction is highlighted). Transaction Tree is presented on the right side.

and various Internet sources. The variability of styles helped us to develop solution, which is flexible enough to handle different approaches for writing use cases.

The proposed idea was implemented as a prototype tool UCTD. It is capable of analysing use cases written in English. The application was developed based on the GATE Framework [10]. The grammatical structure analysis was performed with the use of Stanford Parser [21]. The use case processing chain is presented in the figure 2 and consists of the following steps:

1. Preprocessing phase, which involves use case structure verification (see 4), actors extraction into the GATE Gazetteer form (lists of words which are looked up in the text). Use case is split into title, main scenario and extensions documents. Each part is further tokenized and sentences are annotated.
2. POS tagger is used to annotate parts of speech.
3. English Grammar parser is used to annotate parts of sentence (subjects, predicates, objects etc.).
4. Important words and phrases are being looked up in the scenarios text (actors, validations, response phrases etc.).
5. Actor and system actions are marked with the use of JAPE grammars. Additional postprocessing is done to increase accuracy. The example of use case text with annotations is presented in the figure 3.
6. Transaction Tree is built based on the actor's and system actions sequences. Actions predicates are converted into the sets of predicates synonyms using the WordNet dictionary [15] or with the use of explicitly defined lists of synonyms. The example of the Transaction Tree, extracted from the annotated text, is presented in the figure 4 (the UCTD application screen).

5 Transaction Identification – A Case Study

In order to verify proposed approach for transaction identification we conducted case study, based on thirty use cases coming from the industry project, developed for the e-government sector.

The specification was tagged individually by six experts and UCTD tool. We wished to measure system accuracy as well as investigate how the experts deal with the transaction detection problem in the specification coming from the software industry (*The case study was preceded by the warmup phase with the assessment of four use cases taken from the literature examples*). The detailed results of the case study are presented in the table 1.

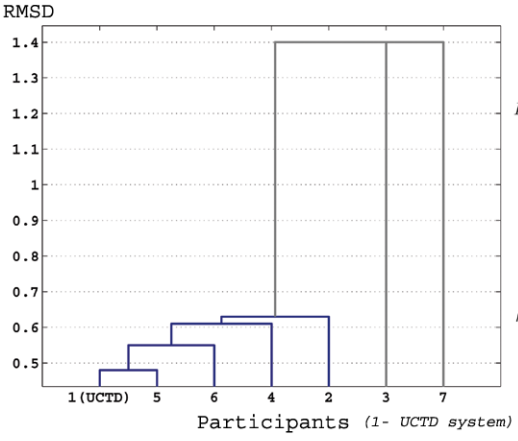
5.1 Comparison of System and Experts Results

Although, use cases structure were correct, we perceived few scenarios as ambiguous from the transactions identification point of view. Thus, we assumed that each expert presents his own approach to transaction counting and neither of their assessments could be accepted as the "standard". In order to measure discrepancy between system and experts opinions, error analysis was performed (concerning differences in number of transactions counted by the participants for each use case). The root mean square deviation (RMSD) matrix, dendrogram (RMSD) and Spearman's rank correlation coefficients matrix are presented in the figure 5. According to the RMSD analysis, experts clearly differs in their decisions. However, three clusters can be distinguished. The group with the greatest cardinality, couples four experts (2, 4, 5, 6) and the UCTD system. The rest of experts (3, 7) differs from themselves as much as from the rest of participants. In this case discrepancy between the UCTD system was not greater then difference between the experts. The average time required for transaction counting

Table 1. Case study results summary. All use cases are presented with the number of steps (in the main scenario and extensions) and transactions counted by the system and experts.

Use Case ID	No. Steps (All)	UCTD(1)	2	3	4	5	6	7
WF1101	14	3	3	3	4	2	3	4
WF1102	14	2	5	4	3	2	3	4
WF1104	5	1	1	1	1	1	1	3
WF1105	7	2	2	2	2	2	1	3
WF1106	6	1	1	3	1	2	1	1
WF1107	4	1	1	1	1	1	1	2
WF1108	4	1	1	1	1	1	1	2
WF1109	6	1	1	2	1	1	1	2
WF1110	14	3	3	7	4	2	2	5
WF1201	9	1	1	3	1	1	1	3
WF1202	6	2	2	3	1	2	1	3
WF1203	3	1	1	2	1	1	1	1
WF1204	6	1	1	2	1	1	1	2
WF2501	9	2	2	3	1	2	2	4
WF2502	5	1	1	2	2	1	1	2
WF2503	17	3	2	3	2	2	2	3
WF2505	7	2	1	2	1	1	2	3
WF2506	11	2	2	3	3	2	1	5
WF3301	3	1	1	2	1	1	1	1
WF3302	5	1	1	3	1	1	1	3
WF3303	2	1	1	1	1	1	1	1
WF4201	4	1	1	1	1	1	1	2
WF4202	3	1	1	1	1	1	1	1
WF4203	9	2	3	5	2	2	1	1
WF4301	18	2	2	5	2	2	2	8
WF4311	7	2	1	1	1	1	1	4
WF4312	6	2	2	3	1	2	1	3
WF4402	5	1	1	3	1	2	1	3
WF4403	4	1	1	2	1	1	1	2
WF4404	3	1	1	2	1	1	1	2
Time	-	41s	39min	35min	17min	28min	34min	46min
$\sum Transactions$	-	46	47	76	45	43	39	83
UCP	320	150	155	170	160	150	150	190

per expert was 33 minutes, while the system processed whole corpus in 41 seconds. The correlation analysis was conducted to identify whether differences in experts opinions are of the systematic nature. Generally, correlation coefficient is higher for experts pairs with the lower RMSD. Lower correlation is observed for experts who differed more in their assessments (higher RMSD). Although, correlation is significantly different from zero in most cases, we did not observed variability caused by experts systematic over or underestimation.



a)

$$RMSD = \begin{vmatrix} 0.000 & 0.658 & 1.483 & 0.606 & 0.483 & 0.548 & 1.742 \\ & 0.000 & 1.402 & 0.632 & 0.683 & 0.683 & 1.826 \\ & & 0.000 & 1.472 & 1.538 & 1.722 & 1.402 \\ & & & 0.000 & 0.730 & 0.683 & 1.807 \\ & & & & 0.000 & 0.577 & 1.880 \\ & & & & & 0.000 & 1.949 \\ & & & & & & 0.000 \end{vmatrix}$$

b)

$$\rho = \begin{vmatrix} 1.000 & 0.899 & 0.642 & 0.770 & 0.798 & 0.796 & 0.731 \\ & 1.000 & 0.771 & 0.845 & 0.889 & 0.755 & 0.636 \\ & & 1.000 & 0.654 & 0.813 & 0.626 & 0.523 \\ & & & 1.000 & 0.729 & 0.741 & 0.586 \\ & & & & 1.000 & 0.662 & 0.586 \\ & & & & & 1.000 & 0.688 \\ & & & & & & 1.000 \end{vmatrix}$$

Fig. 5. Experts transaction counting errors comparison. a) root mean square deviation (RMSD) matrix and corresponding dendrogram (nearest distance) are presented for the system and experts. RMSD coupling reveals that experts differs from themselves (and system) in transactions counting. However, the group of four coherent experts and system, could be observed: the UCTD tool (1) and experts (2, 4, 5, 6). b) Spearman's rank correlation coefficients matrix.

If we consider case study set in the UCP method context, the range of use cases complexity (UUCW), based on transactions marked by the experts was from 150 to 190, with the mean value of 158. Majority of use cases were rather simple (3 from 7 experts marked all use cases as simple, which gave UUCW value 150). The variability could be even grater if analysed use cases were more complex (*if more use cases were closer to simple, average and complex sets membership border values*). For comparison the UUCW counted as a number of steps would be 320.

6 Transaction-Driven Use-Case Writing Style

We have noticed that reason for decisions variance might come from the use cases structural problems (e.g. responds steps omissions, sequences of actor's requests etc.). Furthermore, we observed that literature examples used for the warmup phase caused less difficulties for the experts. We have analysed use cases, which involved greatest experts differences to point the most important problems.

In the figure 6, two versions of the same use case are presented. The first one is coming from the requirements specification assessed in the case study. What is interesting transactions number estimated by experts, varied from 2 to 5 (assessments were 2, 3, 4, 5 transactions). There are at least two main problems in the structure of this use case. The first one is omitting system response phrases. This practice was very often observed. This makes use case shorter but as a side effect scenario becomes more ambiguous. The sequence of actor requests might be grouped within the one transaction. This is acceptable when more general

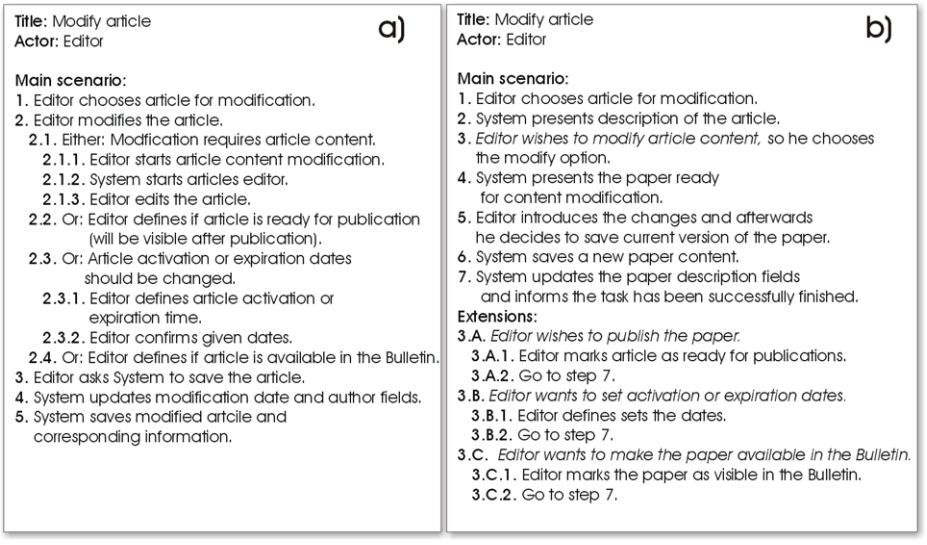


Fig. 6. Use case from the case study corpus. a) original use case, with omitted actions and condition statement, b) modified use case, conforming transaction driven writing style.

action (*e.g. User fills the login form*) is split into many subactions (*e.g. User enters login. User enters password.*). However, such sequence of actor's requests might also emerge as a result of omitting system response. We have noticed that, in some cases, experts where treating single actor's action as transaction, based on their experience. Another important problem is conditional statement in the main scenario. According to the guidelines concerning use cases, main scenario should describe the most common sequence of actions. Introducing many alternative paths in the main scenario makes the use case difficult to read, because the main goal is being blurred.

Second version of the use case, presented in the figure 6, has been rewritten by authors to present more transactional approach to writing use case scenarios. We leave assessment of the two presented versions to the reader.

Based on our experienced, it seems that use cases which cannot be easily split into transactions are potentially ambiguous. Problems in transactions identifications are not an evidence that specification is incorrect, however they should be perceived rather as a "bad smell" (an indicator of potential problem). Providing use cases with a structural problems may lead to a similar variance in experts opinions as in case of presented case study.

We have gathered few guidelines for writing use cases which are suitable for the transactions identification:

- use case should be defined at system level, scenarios should contain actor's and system phrases (*e.g. User does ..., System does ... etc.*). The transaction concept is hardly applicable to the business level use cases.
- actors names (real and system) should be defined explicitly,

- use cases should be provided in a structured textual form (or in other format which might be transformed into text). This means that use case contains sequences of steps, which forms scenarios. Each step begins with a prefix. In order to process extensions, their prefixes should correspond to the step prefix.
- system responses should not be omitted. This helps in finding transaction beginning and closure meaningful from the actor point of view,
- a main scenario should describe the most common sequence of actions. It should not contain conditional statements. If many alternative paths are possible, it seems to be better idea to use extensions triggered on the actor action with assertion statement, which presents actor's intention.

In order to decrease ambiguity of use cases scenarios another action could be taken as incorporating guidelines regarding writing properly structured use cases [1,8,9,26], conducting requirements specification inspections and reviews [3,12]. Many other, potential problems concerning use case structure might be found automatically through the NLP analysis [7,14].

7 Conclusions

In the paper we have presented an approach to automatic transaction counting in use cases. We have proposed a formal model of transaction, which can be used for processing requirements specification. Transaction Tree derived from a textual use case, provides detailed information about its structure (which might be further used not only for effort estimation). Even though the presented model of transaction is formal, it is still easy to applicate. It can be automatically extracted from use-case text in reasonably short time.

The proposed solution for automatic transaction discovery has been implemented as a prototype tool UCTD. Its accuracy was preliminarily verified in the described case study. According to the results, the system did not differ more from the experts than the experts between themselves. Moreover, the tool provided most uniform judgements when compared with the human experts. In addition, conducted case study revealed, that transaction identification is difficult even for people. When experts vary in their opinions, it can significantly impact on the decisions, which are made based on the number of transactions (e.g. size and effort estimation).

The reason for variability in expert opinions might be caused by quality of specification or use case writing style. We believe that knowing number of certain and uncertain transactions might help in detecting structural defects in use cases.

In the near future we would like to elaborate a method and a tool, which would be capable of handling Polish, which is mother tongue for the authors.

Acknowledgments. We would like to thank Alicja Ciemniewska, Piotr Godek, Jakub Jurkiewicz, Wojciech Kopras, Marek Kubiak, Bartosz Michalik and ï£¡ukasz Olek for valuable discussions and involvement in the proposed solution assessment.

This research has been financially supported by the Ministry of Science and Higher Education under grant N516 001 31/0269.

References

1. Adolph, S., Bramble, P., Cockburn, A., Pols, A.: Patterns for Effective Use Cases. Addison-Wesley, Reading (2002)
2. Albrecht, A.J., Gaffney Jr., J.E.: Software function, source lines of code and envelopment effort prediction: a software science validation. Mcgraw-Hill International Series In Software Engineering, pp. 137–154 (1993)
3. Anda, B., Sjøberg, D.I.K.: Towards an inspection technique for use case models. In: Proceedings of the 14th international conference on Software engineering and knowledge engineering, pp. 127–134 (2002)
4. Arnold, M., Pedross, P.: Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department. In: Proc. of the 20th ICSE, pp. 490–493 (1998)
5. Banerjee, G., Production, A.: Use Case Estimation Framework.
6. Beck, K., Fowler, M.: Planning Extreme Programming. Addison-Wesley, Reading (2000)
7. Ciemniewska, A., Jurkiewicz, J., Nawrocki, J., Olek, Ł.: Supporting use-case reviews. In: Abramowicz, W. (ed.) BIS 2007. LNCS, vol. 4439, pp. 424–437. Springer, Heidelberg (2007)
8. Clemmons, R.K.: Project Estimation With Use Case Points. CrossTalk–The Journal of Defense Software Engineering (February 2006)
9. Cockburn, A.: Writing effective use cases. Addison-Wesley, Boston (2001)
10. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A framework and graphical development environment for robust NLP tools and applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (2002)
11. Diev, S.: Software estimation in the maintenance context. ACM SIGSOFT Software Engineering Notes 31(2), 1–8 (2006)
12. Dutoit, A.H., Paech, B.: Rationale-Based Use Case Specification. Requirements Engineering 7(1), 3–19 (2002)
13. Endres, A., Rombach, H.D.: A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories. Addison-Wesley, Reading (2003)
14. Fantechi, A., Gnesi, S., Lami, G., Maccari, A.: Applications of linguistic techniques for use case analysis. Requirements Engineering 8(3), 161–170 (2003)
15. Fellbaum, C.: Wordnet: an electronic lexical database. Mit Pr (1998)
16. Harel, D.: Statecharts: A visual formalism for complex systems. Technical report, Weizmann Institute of Science, Dept. of Computer Science (1986)
17. Harry, A.: Formal Methods Fact File: VDM and Z. Wiley, Chichester (1996)
18. Jacobson, I.: Object-oriented development in an industrial environment. ACM SIGPLAN Notices 22(12), 183–191 (1987)
19. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-oriented software engineering: A use case driven approach (1992)
20. Karner, G.: Resource Estimation for Objectory Projects. Objective Systems SF AB (copyright owned by Rational Software) (1993)
21. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, pp. 423–430 (2003)

22. Kusumoto, S., Matukawa, F., Inoue, K., Hanabusa, S., Maegawa, Y.: Estimating effort by use case points: method, tool and case study. In: Proceedings. 10th International Symposium on Software Metrics, pp. 292–299 (2004)
23. Reif, W.: Formale methoden fur sicherheitskritische software - der kiv-ansatz. Informatik - Forschung und Entwicklung 14, 193–202 (1999)
24. Reisig, W.: Petri nets, an introduction. In: Salomaa, A., Brauer, W., Rozenberg, G. (eds.) EATCS, Monographs on Theoretical Computer Science, Berlin, Springer, Heidelberg (1985)
25. Ribu, K.: Estimating object-oriented software projects with use cases. Master's thesis, University of Oslo, Department of Informatics (2001)
26. Rolland, C., Achour, C.B.: Guiding the construction of textual use case specifications. Data Know Eng. 25(1), 125–160 (1998)
27. Schneider, G., Winters, J.P.: Applying use cases: a practical guide. Addison-Wesley Longman Publishing Co., Inc., Boston (1998)