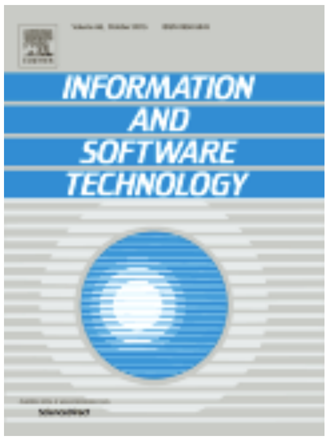


Improving the reliability of transaction identification in use cases[☆]

M. Ochodek[a,*], B. Alchimowicz[a], J. Jurkiewicz[a], J. Nawrocki[a]

[a] *Poznan University of Technology, Institute of Computing Science*
ul. Piotrowo 2, 60-965 Poznań, Poland

Abstract

Context: The concept of transactions is used in Use Case Points (UCP), and in many other functional size measurement methods, to capture the smallest unit of functionality that should be considered while measuring the size of a system. Unfortunately, in the case of the UCP method at least four methods for use-case transaction identification have been proposed so far. The different approaches to transaction identification and difficulties related to the analysis of requirements expressed in natural language can lead to problems in the reliability of functional size measurement.

Objective: The goal of this study was to evaluate reliability of transaction identification in use cases (with the methods mentioned in the literature), analyze their weaknesses, and propose some means for their improvement.

Method: A controlled experiment on a group of 120 students was performed to investigate if the methods for transaction identification, known from the literature, provide similar results. In addition, a qualitative analysis of the experiment data was performed to investigate the potential problems related to transaction identification in use cases. During the experiment a use-case benchmark specification was used. The automatic methods for transaction identification, proposed in the paper have been validated using the same benchmark by comparing the outcomes provided by these methods with on-average number of transactions identified by the participants of the experiment.

Results: A significant difference in the median number of transactions was observed between groups using different methods of transaction identification. The Kruskal-Wallis test was performed with the significance level α set to 0.05 and followed by the post-hoc analysis performed according to the procedure proposed by Conover. Also a large intra-method variability was observed. The ratios between the maximum and minimum number of transactions identified by the participants using the same method were equal to 1.96, 3.83, 2.03, and 2.21. The proposed automatic methods for transaction identification provided results consistent with those provided by the participants of the experiment and functional measurement experts. The relative error between the number of transaction identified by the tool and on-average number of transactions identified by the participants of the experiment ranged from 3% to 7%.

Conclusions: Human-performed transaction identification is error prone and quite subjective. Its reliability can be improved by automating the process with the use of natural language processing techniques.

Keywords: Use-case transactions, Use Case Points, functional size measurement, natural language processing

1. Introduction

Software effort estimation is one of the crucial tasks performed at the early stages of software development to mitigate the risk of budget and schedule overruns. A common approach is to use models that take the size of a system together with factors describing organization productivity as an input, and provide effort estimation as an outcome.

The size of software systems can be measured using code size metrics, such as source lines of code (SLOC). However, these kinds of metrics are more useful when the code of the system is available, because they might be difficult to obtain at early stages of the software development.

Another approach is to measure the size of a system

[☆]This research project operated within the Foundation for Polish Science Ventures Programme co-financed by the EU European Regional Development Fund.

[*]Corresponding author

Email addresses: mochodek@cs.put.poznan.pl (M. Ochodek), balchimowicz@cs.put.poznan.pl (B. Alchimowicz), jjurkiewicz@cs.put.poznan.pl (J. Jurkiewicz), jnawrocki@cs.put.poznan.pl (J. Nawrocki)

based on its functionality. This idea is called functional size measurement (FSM). Probably the most recognizable FSM method is the Albrecht's Function Points Analysis method (FPA) [1]. Because FSM methods employ requirements as an input, they can be effectively used for effort estimation at early stages of software development.

In order to be able to measure such an abstract concept as functionality, the notion of transaction was introduced. It is used to capture the smallest unit of interaction that should be considered while measuring the size of a system. For instance, transactions are used in many derivatives of the Albrecht's FPA method [1], e.g, IFPUG FPA [2]; Mk II FPA [3]; as well as by the use-case-based size/effort estimation methods like Karner's Use Case Points (UCP) [4], Transactions [5], or TTPoints [6, 7].

In order to use transactions to effectively measure the functional size of a system, there has to be a standard procedure for their identification. In the case of FPA great efforts have been made to develop standards and manuals which cover also transaction identification process [8, 9]. A much bigger problem could be observed for the UCP method, for which many different approaches to transactions identification have been proposed so far [4, 5, 10, 11].

In that context two questions arise whether all the definitions and methods for transaction identification in use cases provide *reliable* results, and whether they actually *define the same concept*? Answering these questions is especially important if historical data is supposed to be used for effort estimation or productivity benchmarking.

Therefore in this paper, we would like to discuss and empirically investigate differences and similarities between the different methods for use-case transaction identification, in order to be able to answer the question whether the choice of the method can affect the results of the use-case-based size measurements.

Another important problem related to reliability of use-case-based FSM methods is that transactions are identified based on requirements expressed in natural language, which is usually ambiguous. Therefore, different people using the same method can identify a different number of transactions in the same requirements specification.

To mitigate the problems related to reliability of transaction identification in use cases, we propose to automatized the process of transaction identification, so it can be performed by a software tool instead of people.

The proposed methods for automatic transaction identification were implemented as extensions to the UCWorkbench tool [12]. The implemented methods are able to identify transactions according to the different definitions of use-case transaction and provide objective results.

The paper is organized as follows. In Section 2, a short introduction to use cases is presented. Then in Section 3, the UCP method is briefly described. The history of use-case transactions is presented and discussed in Section 4. The experiment in which four methods for transaction identification were compared is presented in Section 5. The proposed methods for automatic transactions identification are presented in Section 6, and evaluated in Section 7. Finally, in Section 8 the most important findings are summarized.

2. Use cases

Use cases were introduced by Ivar Jacobson in 1986 [13] as a mean for functional requirements description in the telecommunication industry. Since then, use cases have gained a lot of attention and have been wildly applied in the area of requirements elicitation in the software development environment.

Use cases can be presented in two different, but complementary, ways: in the form of diagrams and in the textual form. Use-case diagram allows to present relationships between certain functions, while textual use cases (see Figure 1) allow to show the details of a requirement.

Add a product category

Main flow:

1. Administrator chooses an option to browse defined categories of products.
2. System presents all defined categories.
3. Administrator chooses an option to add a new category.
4. System asks for the data concerning the new category.
5. Administrator enters the data and confirms the operation.
6. System informs that the new category was added.

Alternative and extending flows:

5.A. Not all required data was provided.
  5.A.1. System informs about missing data.
  5.A.2. Back to the step 4.
6.A. Administrator would like to associate the new category with some of the existing categories.
  6.A.1. Administrator selects categories that are related to the added category and chooses the option to associate them.
  6.A.2. System confirms that categories are now associated.

Figure 1: An example of textual representation of a use case.

Every use case in the textual form consists of three main elements which allow to describe a requirement from three different angles. The first element is a meaningful name which should communicate what requirement (user's goal) the use case describe. Set of steps constituting the main flow of actions is the second element of a well written use case. The main flow presents the interaction (in the form of a sequence of steps) between an actor and the system being built. The interaction should demonstrate the typical path of obtaining the goal expressed in the use-case name. Additional paths describing all the alternatives and extensions to the main flow are the third element of every use case.

One of the main characteristics of use cases is that all their elements are written in a natural language what makes them easy to write and read by both analysts and customers.

Apart from use-cases themselves, a Software Requirements Specification (SRS) [14] based on use cases usually consists of two additional parts: actors definitions, describing the main actors which interact with the system being built, and domain-objects definitions describing all the objects of the domain in which the system will operate.

3. Use Case Points

Use Case Points is probably the most recognizable and discussed use-case-based method for effort estimation. The procedure of counting UCP proposed by Karner [4] consists of the following steps:

1. Calculate Unadjusted Actors Weights (UAW). Assign each of the actors to one of three categories based on the type of interface it uses to communicate with the system: *simple* — API, *average* — protocol or terminal, *complex* — GUI. For each category count the total number of assigned actors, and multiply it by the weight assigned to the category (simple=1, average=2, and complex=3). Calculate UAW as a sum of products of number of actors assigned to categories and their weights.
2. Calculate Unadjusted Use Case Weights (UUCW). Count the number of transactions in each use case. Assign use cases into one of three categories based on the number of transactions: *simple* ($T < 4$) , *average* ($4 \leq T \leq 7$), and *complex* ($T > 7$). Calculate UUCW as a sum of products of the number of use cases assigned to categories and their weights (simple=5, average=10, and complex=15).
3. Assess Technical Complexity Factors (TCF). Evaluate the influence of each technical complexity factor, presented in Table 1, by assigning value between 0 and 5. Calculate *TF_Prod* as a sum of products of the factors weights and their influence. Calculate TCF according to Equation 1.
$$TCF = 0.6 + (0.01 \times TF\_Prod) \quad (1)$$
4. Assess Environmental Factors (EF). Evaluate the influence of each environmental factor, presented in Table 1, by assigning value between 0 and 5. Calculate *EF_Prod* as a sum of products of the factors weights and their influence. Calculate EF according to Equation 2.
$$EF = 1.4 + (-0.03 \times EF\_Prod) \quad (2)$$
5. Calculate Unadjusted Use Case Points (UUCP) according to Equation 3.
$$UUCP = UAW + UUCW \quad (3)$$
6. Calculate Use Case Points (UCP) according to Equation 4.
$$UCP = UUCP \times TCF \times EF \quad (4)$$
7. To obtain the effort estimation measured in man-hours, one has to multiply UCP by *Productivity Factor (PF)*. The default value for PF proposed by Karner was 20 hours per UCP. Schneider and Winters [15] proposed rule for choosing PF. They suggested to count the number of environmental factors F1–F6 which influence is predicted to be less than 3, and factors F7–F8 which influence is predicted to be greater than 3. If the counted total is equal to two or less, 20 [h/UCP] should be used; if the total is between 3–4, PF is set to 28 [h/UCP]; if the calculated number is greater than 4, the highest value of 36 [h/UCP] should be used.

Unfortunately, the UCP method proposed by Karner is not formally valid, because it involves calculations that are based on several algebraically inadmissible scale-type transformations [16, 17] (similar problems were observed for Albrecht's FPA [18]). For example, the complexity of each use case is first measured using the ratio scale — the number of transactions, which is then transformed to the three-point ordinal scale (simple, average, and complex), and again transformed back to a ratio-type scale by the arbitrary assignment of the weights. Therefore, the last transformation and further calculation of UUCW as a sum of products of the number of use cases assigned to categories and their weights are not mathematically valid operations.

4. Use-case transactions

The notion of use-case transaction is used in UCP to calculate UUCW (see Section 3). Transactions are also a basis for other use-case-based FSM methods, i.e., Transactions [5] and TTPoints [6].

The use-case transaction was defined, for the first time, by the inventor of use cases — Ivar Jacobson. He

Table 1: Technical Complexity Factors and Environmental Factors.

Technical Complexity Factors		
Factor	Description	Weight
T1	Distributed system	2
T2	Performance	1
T3	End-user efficiency	1
T4	Complex processing	1
T5	Reusable code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Security features	1
T12	Access for third parties	1
T13	Special training required	1

Environmental Factors		
Factor	Description	Weight
F1	Familiarity with the standard process	1.5
F2	Application experience	0.5
F3	Object-oriented experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Stable requirements	2
F7	Part-time workers	-1
F8	Difficult programming language	-1

stated that each use-case transaction should consist of four types of actions [19, 20]:

- actor's request — the main actor sends request and data to the system;
- system validation — the system validates the given data;
- system internal-state change — the system performs operations leading to change of its internal state;
- system response — the system responds to the actor with the operation result.

Therefore, the Jacobson's use-case transaction can be understood as a "round trip", where actor stimulates the system, and the system provides a response to that stimulus. This kind of operation is atomic from the actor's point of view. The initial idea was that each step should constitute a transaction, however, it seems that nowadays using separate steps for actor's requests and system responses seems to be a more favourable approach (e.g. see use cases in [21, 22, 23]).

The concept of use-case transaction was applied to FSM and effort estimation by Karner in his UCP method [4]. The definition of transaction used in UCP [17], states that use-case transaction is "a set of activities, which is either performed entirely, or not at all." (In the definition presented by Anda [24], the term "event" was used instead of "a set of activities".) It seems that this definition follows the idea of a "round trip" — transaction is a set of activities performed *between* actor and the system. However, the interpretation of the definition is not clear, as in two examples of use cases presented by Anda et al. [24, 25] each actor's and system steps are counted as separate transactions.

Another definition was presented by Diev [11], which states that use-case transaction has to satisfy two conditions:

- UCT-C1. Use-case transaction is the smallest unit of activity that is meaningful from the actor's point of view.
- UCT-C2. Use-case transaction is self-contained and leaves the business of the application being sized in a consistent state.

Diev's definition was also used as a basis for defining the semantic transaction types, which will be discussed separately in Section 4.1.

Another approach to transactions identification was proposed by Robiolo and Orosco [5]. They suggested to count the number of stimuli as the number of transactions. In that case, the actor is the subject of the sentence and the stimulus triggered by the actor is the verb.

Recently, Collaris and Dekker [10] attempted to clarify the procedure of transactions identification. They referred to the Jacobson's "round trip" definition of transaction as the proper one. They also commented on existing techniques for transactions identification. For instance, they made a remark that counting transactions is not the same as counting stimuli. They also disagreed with Diev [26], who stated that it is possible that more than one scenario can constitute a single transaction. Their remark was that there are at least as many transactions as there are flows. For instance, the success and the failure of the same operation should be counted as two transactions.

The question arises who is right, and who is wrong? In our opinion neither Diev nor Collaris and Dekker are wrong. The problem could be that the same term "use-case transaction" is used by different authors to name different concepts. Diev's definition of transaction is based on the *elementary process* known from Function

Point Analysis. Garmus and Herron [2] explained elementary process by presenting a scenario of filling in a form. Assume one has to fill a form consisting of three pages. The elementary process will be finished when all of these pages are completed and submitted, because the intention was to fill in the whole form (submitting a single page is neither meaningful nor leaves a business in a consistent state). If the "round trip" definition of use-case transaction is considered, those three steps would constitute three different transactions — if all sub-forms were submitted one after another.

4.1. Semantic transaction types

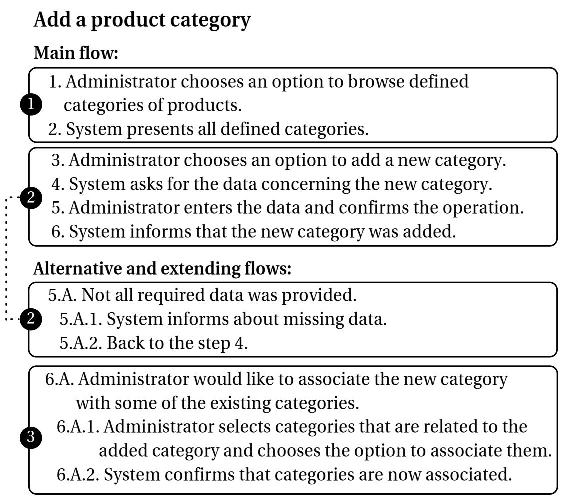
Fetcke et al. [27] who proposed mapping between use cases and Function Point Analysis, did not find an easy way to map interaction in use-case scenarios to FPA transactions. They suggested to perform analysis according to the Function Points counting rules instead. Therefore, it seems that different approaches to analysis of use cases should be employed when counting Jacobson's "round trip" transactions, and elementary-process-based transactions.

In order to be able to identify an elementary-process-based transaction, one has to understand what meaningful goals can be obtained by the actor within the scenarios of the use case being analyzed. If one looks at transactions from the semantic point of view, it could be observed that some of transactions in use cases share similar goals. Based on such similarities of goals, 12 semantic transaction-types have been distinguished so far [6]:

1. Create (C) - describes creating a new instance of a domain object. Data provision is the central point of this kind of transactions. It could be followed by the validation of the entered data and presentation of the results.
2. Retrieve (R) - allows an actor to retrieve some existing data. It usually consists of two actions: choice of some retrieve option, and displaying the requested data.
3. Update (U) - describes how to alter the already existing domain objects. Similar to the Create transaction. The main focus is on the data provision, however, in this case the domain object needs to exist before this type of transaction can be performed.
4. Delete (D) - describes how the already existing domain object is removed from the system. This type of transaction, usually, includes the following actions: choice of the required domain object, selection of a delete option, and confirmation provided by the system.
5. Link (L) - after performing this type of transaction two or more objects, which are not a composite, are associated. It involves choosing the domain objects, optionally accompanied by the provision of the data; choosing a link option; optionally performing some validation operations; and finally presenting the results.
6. Delete Link (DL) - this is an opposite type of transaction to the Link transaction. Here, the selected links between domain objects are removed.
7. Asynchronous Retrieve (AR) - describes the process of domain object retrieval (similar to the Retrieve transaction), however, in this case the fetched data is not provided as a direct response (e.g., sending an e-mail). Transfer of the data is the main action in this type of transactions.
8. Dynamic Retrieve (DR) - another type of transactions describing data retrieval. In this case the set of criteria upon which the data is retrieved needs to be provided.
9. Transfer (T) - allows the transfer of numerous domain objects from one actor to another. After the object is transferred it is managed by the receiver.
10. Check Object (CO) - the only goal of performing this kind of transaction is to validate the data against some specified rules.
11. Complex Internal Activity (CIA) - can be used in order to describe complex operation performed by the system. The action which is the main point of this type of transactions is usually specific to the system being described (e.g., invoking some complex algorithm).
12. Change State (CS) - very similar to the Update transaction, however, in this case the change made to the domain object influences its behaviour in the system.

In Figure 2 one can find an example of a simple use case with three[1] different transactions: Retrieve (1), Create (2), and Link (3). The first transaction describes how the existing data is fetched and presented to the user. Creating new domain object, and adding it to the system is the main purpose of the second transaction. It is worth mentioning that steps from the alternative flows section can also be a part of the transaction (or even create a separate one). The third transaction presents how different

[1]Note that if the "round trip" definition of a transaction was considered, the second transaction presented in Figure 2, would be decomposed to three transactions: choice of options followed by the presentation of the form (steps 3, 4); successful filling in the form (steps 5, 6); failing to fill in the form (steps 5, 5.A.1, 5.A.2).


Add a product category

Main flow:

1. Administrator chooses an option to browse defined categories of products.
2. System presents all defined categories.
3. Administrator chooses an option to add a new category.
4. System asks for the data concerning the new category.
5. Administrator enters the data and confirms the operation.
6. System informs that the new category was added.

Alternative and extending flows:

5.A. Not all required data was provided.
5.A.1. System informs about missing data.
5.A.2. Back to the step 4.

6.A. Administrator would like to associate the new category with some of the existing categories.
6.A.1. Administrator selects categories that are related to the added category and chooses the option to associate them.
6.A.2. System confirms that categories are now associated.

Figure 2: An exemplary use case with transactions marked using semantic transaction-types.

instances of domain objects are being connected with each other.

5. Experimental comparison

The fact that discussed definitions of use-case transaction seem to have different origins is important, however, the more important question regarding the reliability of UCP is whether they provide the same on-average results when used by people?

In order to be able to answer this question a controlled experiment was conducted. Its goal was to investigate whether there is a cause-effect relationship between the choice of the method for transaction identification and on-average number of identified transactions.

5.1. Experiment design

The *independent variable* considered in the experiment was a method used to identify use-case transactions (four values considered). The *dependent variable* was the number of transactions identified in a use-case-based requirements specification.

Transaction identification methods. Four methods for transactions identification where investigated during the experiment:

- **M1** — counting stimuli-verbs according to Robiolo and Orosco [5];
- **M2** — identifying transactions based on the definition provided in Karner's UCP [4];
- **M3** — identifying transactions based on the definition provided in UCPm [11], which is based on the elementary process taken from Function Point Analysis;
- **M4** — identifying transactions by using semantic transaction types described in Section 4.1.

Software Requirements Specification. Number of identified transactions can depend both on the structure of use-cases and author's writing style. To mitigate the risk of using a specific requirements specification (i.e., too easy, or too difficult to analyze), we decided to use the benchmark requirements specification [28]. It is an instance of a typical use-case-based requirements specification, derived based on the analysis of 524 use cases coming from 16 projects.

Participants. Participants in the experiment were 120 3rd-year students, who were in the middle of the second semester of the two-semester Software Engineering course. The participants were familiar with the concept of use cases as they took part in a lecture about use-case-based requirements specifications. Their tasks during the laboratory classes were to create and review the requirements in a form of use cases for a small software project. During the lecture and the laboratory classes, the students were taught what the structure of a use case is, how to write use cases, and what are the good practices for authoring use-cases.

Participants were randomly assigned to four groups: *G1* that was asked to use the method M1; group *G2* used the method M2; group *G3* used the method M3; and *G4* used the method M4.

5.2. Operation of the experiment

The experiment was executed at the Poznan University of Technology in April 2010.

Prepared instrumentation. Each participant in the experiment was provided with the handouts that contained a description of the UCP method, instructions, and tables for collecting the number of transactions for each use case. Participants also had access to the presentation with the definition of the transaction identification method appropriate for their group, together with examples of its usage.

During the execution of the experiment participants used a web-based system that provided the interactive version of the requirements specification. It enabled participants to visually tag transactions in each use case.

As a result, transactions identified by each participant were automatically recorded by the system, together with the exact steps constituting each transaction,

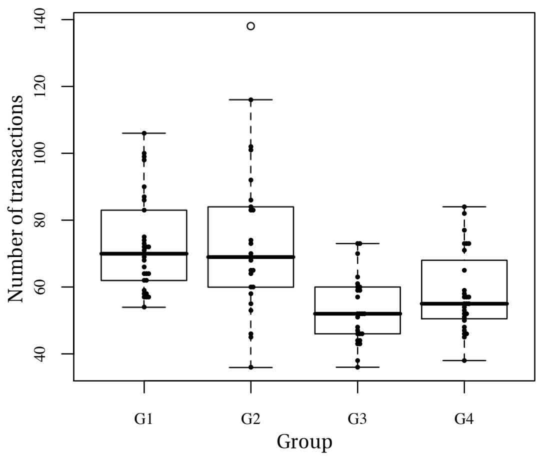


Figure 3: Box-plots presenting the number of transactions identified by the participants of the experiment.

and independently by each participant on the provided forms — in that case only the number of transaction in each use case was recorded.

In order to preserve homogeneity of the samples, the participants were asked whether they used the UCP method, or identified transactions before.

Execution. During the execution of the experiment students were supervised by lecturers, who were obliged to answer questions concerning the UCP method, excluding those referring to the transaction identification process. Participants had 90 minutes to acquaint themselves with the transaction identification method appropriate for their group, and identify transactions in 34 use cases.

5.3. Analysis and interpretation

Data verification. After collecting and reviewing participants' forms (electronic and paper version) 2 of them were rejected because of their incompleteness. Another 7 observations were rejected because we suspected that students did not participated seriously in the experiment (e.g., random steps were tagged as transactions).

Descriptive statistic. Before proceeding to further analysis, the experiment data was investigated to find and handle outlying observations. After completing this process 2 outlying observation were rejected. Figure 3 presents the distributions of the total number of transactions identified by the participants of each group. Descriptive statistics are presented in Table 2.

As a result 109 forms were classified for the further analysis (G1–30, G2–25, G3–26, G4–28).

Table 2: Number of transactions identified by the participants of the experiment (after cleaning).

	G1	G2	G3	G4
#1	75	74	46	54
#2	54	92	44	65
#3	72	46	73	77
#4	90	65	44	47
#5	87	53	46	84
#6	69	45	48	51
#7	57	69	73	59
#8	68	116	51	82
#9	57	73	60	45
#10	62	64	57	73
#11	64	102	52	38
#12	74	55	59	55
#13	71	101	59	55
#14	62	138	52	73
#15	58	83	63	55
#16	72	68	36	58
#17	106	83	43	52
#18	66	36	43	46
#19	73	60	52	48
#20	72	86	61	73
#21	59	58	60	50
#22	64	70	70	71
#23	64	84	52	57
#24	100	65	47	52
#25	58	60	46	53
#26	83		38	57
#27	57			57
#28	99			46
#29	98			
#30	86			
N	30	25	26	28
Min	54	36	36	38
1st Qu.	62	60	46	50.75
Median	70	69	52	55
Mean	72.57	73.84	52.88	58.32
3rd Qu.	81	84	59.75	66.50
Max	106	138	73	84

The next step was to check whether samples come from a normally distributed population. The initial observation made based on the analysis of Q-Q plots [29] was that the assumption about samples normality might be violated. This suspicion was further confirmed by the Shapiro-Wilk test [30]. Therefore, we decided to use a non-parametric statistical tests.

5.3.1. Quantitative analysis

Central tendency. Methods M1, M2, M3, and M4 could be treated as measurement instruments for counting transactions in use cases. By investigating central tendencies of the experiment samples, one can investigate whether the methods on-average provide similar results.

Therefore, the null hypothesis was formulated that the median number of transactions identified in the same specification with the use of all four methods were equal:

$$H_0 : \theta_{G1} = \theta_{G2} = \theta_{G3} = \theta_{G4} \quad (5)$$

The alternative hypothesis was formulated that the median number of identified transactions was not equal for at least two groups:

$$H_1 : \text{Not } H_0 \quad (6)$$

The observed normalized effect size[2] expressed as Cohen's d coefficient [31] was "large"[3] for groups G1–G3, G1–G4, G2–G3, and G2–G4 (the observed values of d ranged from 0.86 to 1.55). The observed effect size between groups G3 and G4 was "medium" (d = 0.49). The smallest effect size was observed for groups G1 and G2 (d = 0.07).

To test the hypotheses we used the Kruskal-Wallis test [32], which is a non-parametric version of ANOVA. The significance level α was set to 0.05. The obtained value of the χ^2 statistics was equal to 31.4347 (df=3). It would allow us to reject the null hypothesis with the significance level less than assumed 0.05 (p-value = 6.885×10^{-7}). Therefore, we concluded that there is a significant difference between the median number of transactions of at least two groups.

Then, we performed the post-hoc pairwise comparison of subgroups according to the procedure proposed by Conover [33]. For each pair of samples two values were calculated: the difference between mean ranks, and the critical difference of the mean ranks. If the difference between mean ranks is greater than the calculated critical value, the difference is treated as significant. According to the results of the analysis, presented in Table 3, the differences between the median number of transactions seemed to be *not* significant only for the comparisons between groups G1–G2 and G3–G4.

The different on-average number of transactions obtained by participants using Karner's [4], and Diev's [11] methods supported the thesis that the definitions, which have different origins, might define different constructs in use cases. (However, the quantitative results are not a sufficient evidence on their own.)

[2]Please note that retrospectively calculated effect size only approximates the real effect size in the populations from which the samples were drawn.

[3]According to Cohen [31] effect size is perceived as "small" if the value of d is equal to 0.2, as "medium" if the value of d is equal to 0.5, and as "large" if d is equal to 0.8.

Table 3: The post-hoc analysis [33] of Kruskal-Wallis test (in each cell, column vs. row: difference of the mean ranks; the critical difference of the mean ranks — in brackets; * denotes a statistically significant difference).

	G1	G2	G3	G4
G1				
G2	3.56 (14.49)			
G3	40.42* (14.34)	36.86* (14.99)		
G4	28.97* (14.06)	25.41* (14.73)	-11.45 (14.58)	

In addition, lack of significant differences in on-average results between groups G1–G2 supported the thesis that stimuli-verbs method proposed by Robiolo and Orosco [5] can be treated as a heuristic approach for identification of transaction based on the Karner's definition of use-case transaction. The same relates to groups G3–G4, however, the semantic transaction-types by their definition are supposed to comply with Diev's definition of use-case transaction.

The observed difference in on-average number of transactions between the groups is also important for the reliability of use-case-based size metrics. For example, the ratio between the mean number of transactions for the groups G2 and G3 was equal to 1.4. Such difference would have a visible impact on the estimated effort if one used the number of transactions as a functional size measurement [5]. If the productivity factor was calculated for the same project, based on two variants of the size measure, and then cross-used to estimate effort for the project, the magnitude of relative error (MRE) would range from 0.3 to 0.4. Moreover, if the extreme size measures provided by the participants belonging to these groups were considered, the MRE would range from 0.7 to 2.8.

An additional aspect that should be considered is the influence of the use-case categories, used in the UCP method to calculate the Unadjusted Use Case Weights (see Section 3). The distributions of UUCW calculated for the participants of each group, presented in Figure 4, seem to confirm the observations made for the number of transactions. However, because of the cut-off effect of complexity classes, if on-average values of UUCW in groups G2 and G3 were used to estimate effort with UCP, the MRE would be equal to 0.1 in both cases. Again, if extreme values of UUCW in the same groups were considered, the MRE would range from 0.4 to 0.6.

Variability. Another important aspect is the inner variability in the results obtained by participants using the same method. The lesser the variability observed for a group, the more reliable is the procedure for transac-

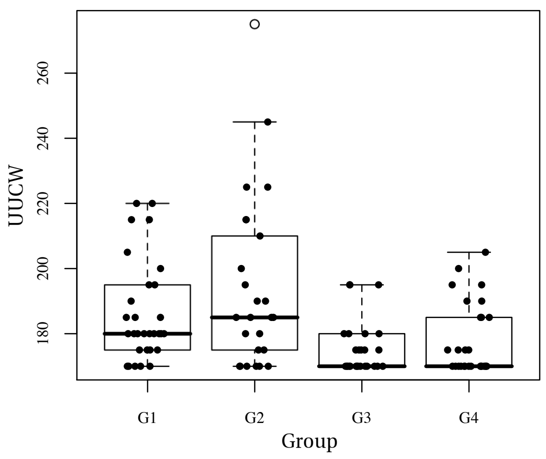


Figure 4: Box-plots presenting the Unadjusted Use Case Weights (UUCW) calculated based on transactions identified by the participants of the experiment.

tion identification.

The observed inner variability in the number of transactions for all groups was high. The ratios between the maximum and minimum number of transactions were equal to 1.96, 3.83, 2.03, and 2.21. However, the conclusions concerning the reliability of investigated methods for transactions identification that were drawn based only on the quantitative analysis of dispersion, should be treated with caution. One has to keep in mind that the students did not have previous experience in counting transactions in use cases.

5.3.2. *Qualitative analysis*

Data collected by the system supporting the experiment allowed us to make some observations concerning the usage of methods for transactions identification.

The most frequent problem for the participants, who used the method M1, was a choice of verbs that are stimuli. It appeared that different people could choose different verbs as stimuli. While verbs which describe typical interaction with the system (e.g., "submit", "select", or "enter") were rather equally marked as stimuli, verbs describing more general activities (e.g., "find", "browse", or "go") were misleading, and often led to different interpretations.

Two main issues were observed regarding the method M2. The first one was concerned with sequences of steps performed by the same actor. If there are two or more consecutive steps performed by the same actor, the system responses between those steps are not explicitly defined. As a result we observed different ways of dealing with this case: either by selecting the whole sequence as one transaction, or by selecting each of the steps as separate transactions. The second problem was related to how alternative flows were handled. Some of the participants tended to mark alternative flows as parts of the transactions identified in the main flow. However, there was also a group of participants who marked each of the alternative flows as a separate transaction.

The main problem regarding the methods M3 and M4 was a granularity of transactions. There was a group of use cases that all of them consisted of three parts: preparation for the main action (e.g., choice of option), the main action itself (e.g., providing the data), presentation of the results (e.g., the system presents added object). Some participants treated all three parts as a single transaction (the goal of transaction was the same as goal of the use case), while other distinguished separate transactions for each part.

The notion of semantic transaction types is the main difference between the methods M3 and M4. The need of pointing the right transaction type made transaction markings more consistent and cohesive. Still a few students had problems with distinguishing between some of the types (e.g., Retrieve and Dynamic Retrieve, or Update and Change State). Another issue with the method M4 was that inexperienced user sometimes misled the actions of a given type with the transaction of similar type. For instance, the validation action which was a part of the Create transaction, was sometimes marked as a separate Check Object transaction.

5.3.3. *Threats to validity*

In the case of threats to *internal validity* the potential problem could be a difference in the students level of knowledge regarding UCP and use cases. To exclude participants with the previous experience with using the UCP method, those who admitted to already used the method would not be considered. All participant also had some experience in writing and reviewing use cases, as described in Section 5.1.

Another factor that could influence the outcome of the experiment was the quality of the instrumentation provided to the participants. To mitigate that risk we provided definitions, and examples as they were published in the original papers. If there was insufficient number of examples available for a method in the original papers, additional examples were prepared by the authors of this study.

In case of threats to the *external validity* there is also the problem of generalizing the outcome of the experiments which participants are students to the population of software companies employees. However, in the case of the experiment regarding methods for transactions identification, it was important that participants had no previous experience in that area, because it could interfere with the provided transaction definitions. It is also probable, however, that the more experienced participants would be also more convergent in their responses. Therefore, the observed differences in the variability should not be taken as exact measure of variability that would be observed for the professionals. On the other hand, in case of our previous case study [34] with six experienced estimators who were not forced to use any specific definition of transaction, the difference in the maximum and minimum number of identified transactions was also high (a factor of 2.1).

6. Automatic transaction identification

The main conclusion from the conducted experiment is that the choice of the method for transaction identification can have a visible impact on the values of the use-case-based FSM. Therefore, if an organization would like to employ use-case transactions for effort estimation and productivity benchmarking, it has to decide to consistently use only one method for their identification. In addition, a set of counting rules should be developed to reduce the intramethod variability.

Another approach to mitigate the problems related to reliability of transaction identification is to automate the identification process. The main advantage of using software tools instead of expert knowledge is that the tool should always provide the same results when used to analyze the same software requirements specification. Another benefit is that the software tool is more efficient than people. Therefore, the number of transactions could be re-calculated after each change to the requirements.

Some tools for automatic transaction identification in use cases have been proposed so far [34, 35, 36]. They either process UML diagrams in XMI format or analyze use cases in the textual form with the use of natural languages processing techniques (NLP). Unfortunately, these tools do not take into consideration different definitions of transactions. If a tool was able to identify transactions based on different definitions, it could support effort estimation based on different historical databases, even if transactions in such databases were identified using different methods. Moreover, it would help members of a development team to verify their counts, or to help them to investigate which method they actually use. Finally, the same project can be measured using different methods to obtain a set of size estimations (e.g., the best and worst cases). Therefore, we would like to propose a set of approaches to automatic transaction identification based on stimuli [5], "round trips" [10, 19], and elementary-process-based transactions, using the semantic transactions types [6].

6.1. Graph representation of use cases

The flow of control in a use case can be represented by a directed graph $G(V, A)$. Each vertex (V) represents a single step of a use case — expressed in a natural language. Possible transitions between steps are represented by arcs (A). In addition, arcs can be labeled with descriptions of events or conditions, which start alternative scenarios. An example of a control flow graph for a use case is presented in Figure 5 (sub-figure a).

6.2. Actions and their identification

Unfortunately, the basic graph representation is insufficient for a software tool to automatically identify transactions in use cases. First of all, a single step can consist of more than one action. For instance, in step 5 of the use case presented in Figure 5, there are two actions performed by the actor called administrator. In addition, it is necessary to distinguish actions performed by the system under development (SuD) and actions performed by other actors. Finally, it is necessary to understand the semantics of actions in order to be able to identify the elementary-process-based use-case transactions. Therefore, the initial control flow graph of a use case should be transformed to a new directed graph $G'(V, A)$. Such graph would have vertices (V) that represent actions performed by actors. Each action consists of a subject, which should represent an actor, a predicate describing the activity, and a noun phrase containing an object. The transformation can be done automatically by analyzing the text of each vertex in the base graph $G(V, A)$ and the descriptions of events, with the use of a processing chain performing the following NLP analyses:

1. *Sentence segmentation*: In this stage a string that represent a use-case step is divided into separate sentences.
2. *Word segmentation*: This step implements a process of dividing a string that represent a use-case step into words.
3. *Part-of-speech tagging*: In this step, a part of speech of each word in a use-case step is identified. The process is based on the definition of a word and its context, i.e., relationship with adjacent and related words in a phrase or sentence.
4. *Lemmatization*: It is a process of determining the lemma for a given word (lemma is a canonical

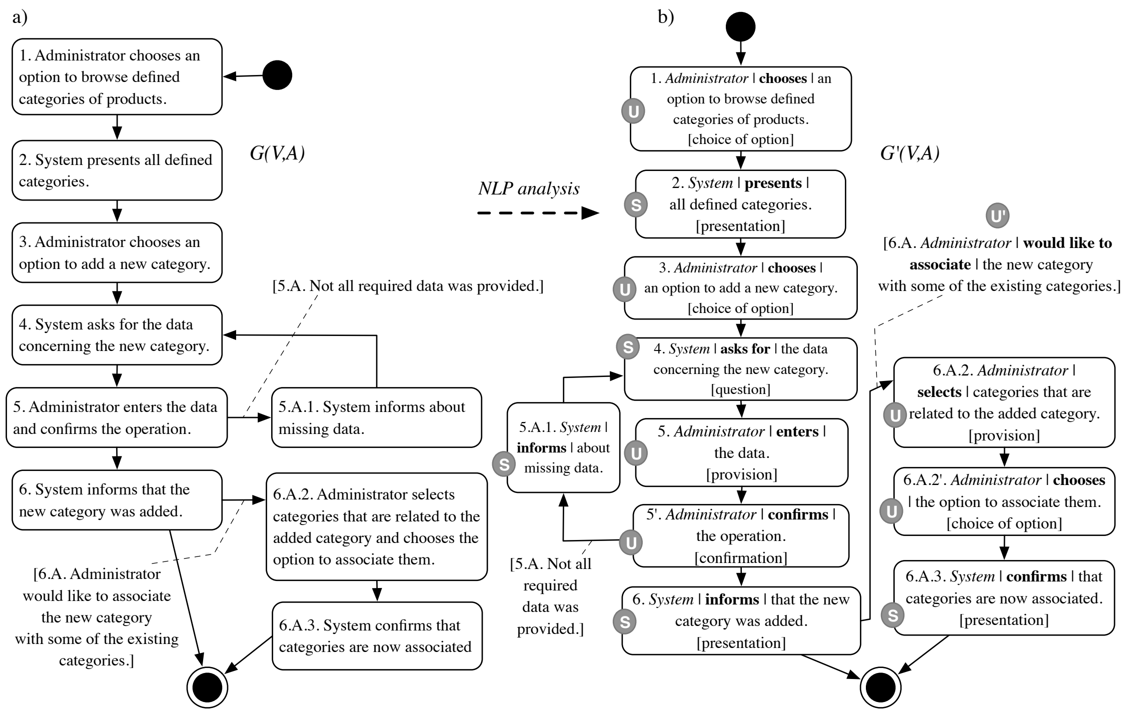

a)
1. Administrator chooses an option to browse defined categories of products.
2. System presents all defined categories.
3. Administrator chooses an option to add a new category.
4. System asks for the data concerning the new category.
5. Administrator enters the data and confirms the operation.
6. System informs that the new category was added.
[5.A. Not all required data was provided.]
5.A.1. System informs about missing data.
6.A.2. Administrator selects categories that are related to the added category and chooses the option to associate them.
6.A.3. System confirms that categories are now associated
[6.A. Administrator would like to associate the new category with some of the existing categories.]
G(V,A)
NLP analysis
b)
G'(V,A)
1. Administrator | chooses | an option to browse defined categories of products. [choice of option]
2. System | presents | all defined categories. [presentation]
3. Administrator | chooses | an option to add a new category. [choice of option]
4. System | asks for | the data concerning the new category. [question]
5. Administrator | enters | the data. [provision]
5'. Administrator | confirms | the operation. [confirmation]
6. System | informs | that the new category was added. [presentation]
5.A.1. System | informs | about missing data.
[5.A. Not all required data was provided.]
U'
[6.A. Administrator | would like to associate | the new category with some of the existing categories.]
6.A.2. Administrator | selects | categories that are related to the added category. [provision]
6.A.2'. Administrator | chooses | the option to associate them. [choice of option]
6.A.3. System | confirms | that categories are now associated. [presentation]
U
S

Figure 5: Use case represented as directed graphs: a) base graph with text in vertices; b) transformed graph with identified actions (U — actions performed by actors not representing the system under development (SuD), S — actions performed by SuD, U' — decision points for actors other than SuD).

form of a set of words). For example, words “category” and “categories” would both have “category” as the lemma. Lemmatization might help finding the occurrences of domain objects in a text.

5. *Grammatical relations*: In this step grammatical relationships between words in a sentence are resolved. As a result of the analysis, all subjects, predicates, and objects in sentences of each use-case step are identified.

6. *Lookup markers*: It is a set of tools that perform a dictionary-based lookup for words that are meaningful for the interaction of actors in use cases. For example, one group of words we mark have the meaning “to show” (ShowWord), e.g., “present” “display”, “show” etc. In order to increase the accuracy of lookup tools we match the words using their lemma and part-of-speech.

7. *Reference marker*: It is a tool similar to a lookup marker, however, it marks the occurrences of actors, use-cases, and domain objects, based on their definitions. The tool also indicates the type of actor (external actor or system under development) based on its definition, i.e., the meta information or actors’s name.

8. *Action marker*: Actions are marked based on the identified grammatical relationships in sentences, i.e., subjects, predicates, and objects. In addition, each action has to have an actor as a subject. Moreover, the type of the action is determined based on its predicate — using the words marked during the lookup stage (e.g., if a predicate is also a ShowWord, the type of action is set to “presentation”). The types of actions that are considered in the proposed methods for transaction identification are presented in Table 4.

Once actions are identified, each vertex in the base graph is replaced by a sequence of vertices that correspond to actions identified in the text of the vertex. An example of the base and transformed control flow graphs are presented in Figure 5.

As a pre-processing stage, the retreating arcs are also identified in the transformed graph [37]. Identification of the retreating arcs makes the analysis of a use-case structure easier, because the retreating arcs represent transitions from the alternative flow to the main flow, and prevent from entering infinite loops while traversing the structure of a graph.

Table 4: The types of actions in use cases that are considered in the proposed methods for transaction identification.

Actor	Name	Description
U	Choice of option	User chooses one of the available options.
U	Provision	User provides data to the system.
U	Confirmation	User confirms the previous action.
U	Modification	User modifies the data in the system.
S	Presentation	System presents some information to the user.
S	Transfer	System transfers data to some other system.
S	Store	System stores data.
S	Question	System asks a user for additional information or for confirmation.
S	Data validation	System validates the provided data.
S	Removal	System removes data.

6.3. Identification of stimuli-verbs transactions

Robiolo and Orosco [5] suggested that in order "to clarify the actor and stimulus identification, a syntactic analysis of the text could be performed. The actor will be the subject of the sentence and, the stimulus triggered by the actor will be the verb." Therefore, each stimulus has to be expressed by a single action that is performed by the external actor. We will denote actions performed by external actors by U. Unfortunately, Robiolo and Orosco did not provide detailed rules that would enable us to automatically differentiate between stimuli actions and other U actions (they discussed the idea of stimuli-verbs on the example). Therefore, we will base the method for transaction identification on the assumption that the number of stimuli is equal to the number of independent sequences of U actions. This assumption defines the lower boundary of the real number of transactions in a use case, because after each independent sequence of U actions there has to be a system response. Therefore, the sequence of U actions has to contain at least one stimulus. However, if some system responses within the sequence of U actions were omitted, there would be some additional (hidden) stimuli. We will provide some guidelines on how to mitigate this problem at the end of this section.

In addition to the already defined U action, let U' denote the event that contains at least one action which is performed by an external actor (we can call it a decision point), and let S denote a single action performed by a system under development.

The procedure of identifying sequences of consecutive U actions could be performed in two steps. The first step is to find all vertices that correspond to U actions and all arcs that are labeled with the U'-kind of events. It can be done by traversing the graph using the depth-first search algorithm (DFS). The next stage is to find sequences of adjacent U (and/or U') actions. For each vertex representing a U action visit all its succeeding vertices. If the succeeding action is also of the type U, then, if it has not been visited before, add it to the current sequence of U actions; if it has already been visited, append the current sequence to the sequence to which the succeeding action belongs to. An example of the outcome of the method is presented in Figure 6. As a result of the analysis performed for the use case presented in Figure 5, four independent sequences of U actions were identified, which implies presence of at least four transactions in that use case.

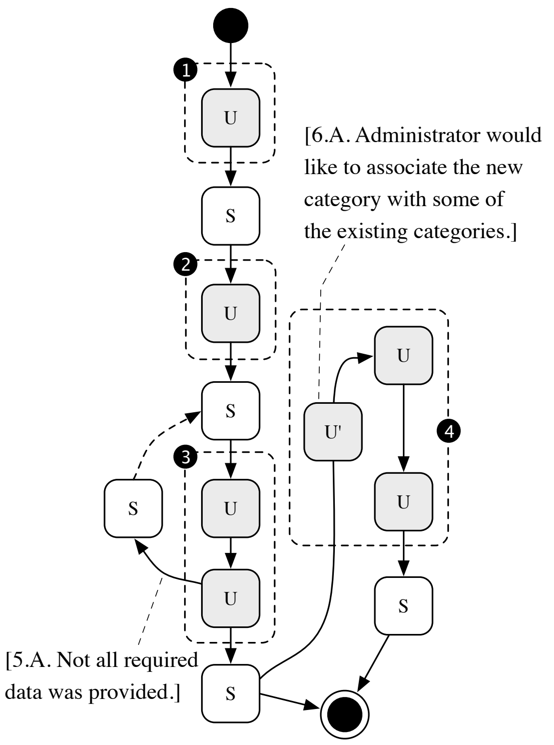

1
U
S
[6.A. Administrator would like to associate the new category with some of the existing categories.]
2
U
S
U
U'
4
3
S
U
U
U
S
[5.A. Not all required data was provided.]
S

Figure 6: The results of automatic transaction identification based on stimuli-verbs approach for the use-case presented in Figure 5.

The accuracy of the method can be further improved by performing a post-hoc analysis of each sequence of U actions, to find the pairs of actions that should be divided by a missing S action. For instance, if two consecutive actions U_i and U_j have the same type "confirmation," we could assume that there should be an additional system response between them. Similarly, if there are two consecutive actions U_i and U_j; and U_i has an alternative flow attached to it which contains at least one S-kind of action, we could also assume that there is a missing S action between these two actions.

6.4. *Identification of round-trip transactions*

The core part of each "round trip" transaction is a pair of actions — a user request and the system response. Therefore, in order to identify the number of transactions in a use case we would have to calculate the number of consecutive sequences of U and S actions.

However, the main problem relates to the interpretation of the alternative flows of a use case. For instance, the question is how to interpret the situation where there are two possible responses for a single request, e.g., a single operation can finish with a success or a failure.

Therefore, we decided to consider two alternative variants of interpreting the alternative flows of a single use case:

- A1: if a forking event contains actions that are performed by external actors (U'), it implies a presence of a new transaction in the alternative flow, because it represents a decision point. If the forking event does not contain any actions performed by the external actor, then, new transactions are identified in the alternative flow in the same way as in the main flow. Therefore, if there is a "round trip" in the alternative flow it is identified as a separate transaction.
- A2: each alternative flow implies an additional transaction, as proposed by Collaris and Dekker [10].

The first stage of processing is to add a new vertex for each arc that is labeled with the U'-type of event. The new vertex corresponds to the action that was found in the description of the event. The next step is to analyze each arc in the graph (a pair of actions $A_i \rightarrow A_j$), in order to find the ending blocks of each transaction.

The arc is considered as an ending block of a transaction, if it connects any of the following pairs of verticies (let *fin* denote the ending vertex of a use-case flow control graph):

- $S \rightarrow U$: it is a situation where the first transaction finishes, and the next one starts;
- $S \rightarrow fin$ or $U \rightarrow fin$: it corresponds to the end of a use case, which also implies the end of transaction;
- $S \rightarrow U'$ or $U \rightarrow U'$: external actor's decision point implies the beginning of a new transaction.

The difference between the approaches A1 and A2 relates to the interpretation of the retreating arcs. For the method A2 each retreating arc is automatically treated as the ending block of the transaction.

An exemplary outcome of the method is presented in Figure 7. It presents transactions identified for the use case presented in Figure 5. The number of identified transactions is equal to 4 for the variant A1, and equal to 5 for the variant A2.

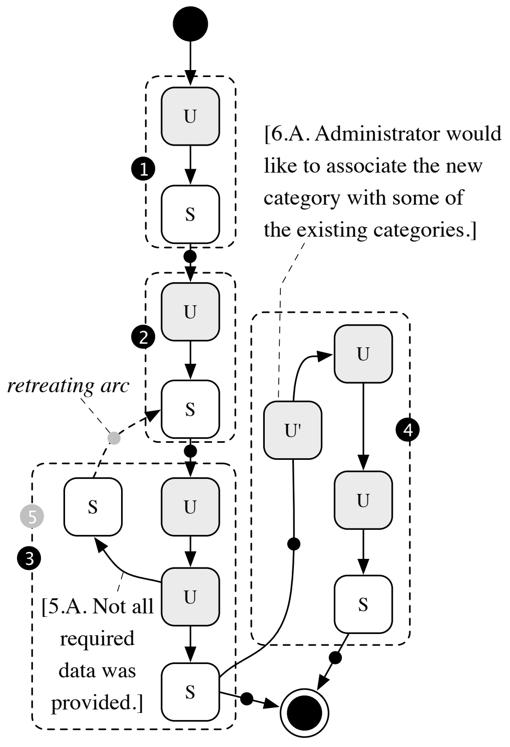

U
S
[6.A. Administrator would like to associate the new category with some of the existing categories.]
1
2
3
4
5
U
S
retreating arc
S
U
U
S
U'
U
U
S
[5.A. Not all required data was provided.]

Figure 7: The results of automatic transaction identification based on the "round trip" definition of use-case transaction for the use case presented in Figure 5.

The accuracy of the method could be also improved by analyzing the consecutive U actions, as in the case of the stimuli-verbs approach. However, such analysis should be performed as a pre-processing phase of the method. If the conclusion is made that there is a missing action between two consecutive U-type actions, an artificial S-vertex should be added between these actions.

6.5. *Identification of semantic transactions*

Identification of elementary-process-based use-case transactions (M3 and M4) is a more demanding task, because it has to consider not only the structure of a control flow graph, but also the semantics of its actions. Therefore, the proposed method will be based on the idea of semantic transaction types, which was presented in Section 4.1.

The first stage of the method is to identify the “round trip” transactions (A1). This kind of transactions represents the smallest unit of interaction between the external actor and the system under development. However, a single transaction does not have to lead to obtaining a meaningful goal, and does not have to leave the business of application in a consistent state.

The next step of the method is to identify the semantic type of each “round trip” transaction. This is performed by analyzing the types of actions constituting the transaction, and by matching the lookup words, according to the decision tree presented in Figure 8.

The conditions in the decision three are tested in the top-down order. There are three possible results of each step of the matching process:

- R1: a rule is activated that implies a single type of a transaction. In such case, the type is assigned to the transaction and the matching process finishes.
- R2: a rule is activated that implies possibility of more than one transaction types. In such case, all the suspected transaction types are assigned to the transaction, and matching procedure continues.
- R3: none of the rules were activated. The type of transaction is set to “none.”

The next step of the method, is the consolidation of the “round trip" transactions. The process starts by sorting the transactions based on the reversed order of traversing the control flow graph, with the use of the DFS algorithm. Then, for each transaction a set of succeeding transactions is found[4]. The consecutive transactions T_i and T_j are consolidated, if at least one of the following conditions is true:

- either T_i or T_j is empty;
- there is an arc $A_k \rightarrow A_l$ that connects transactions T_i and T_j; and A_k is a “question” action and A_l is a “provision” action;
- there is an arc $A_k \rightarrow A_l$ connecting transactions T_i and T_j; and both A_k and A_l are “confirmation” actions;
- the type of T_i is “Retrieve”; the type of T_j is “Update” or “Change State”; and there is an arc $A_k \rightarrow A_l$ connecting the transactions, for which A_k is a “presentation” action and A_l is a ”provision” action and both actions process the same domain object or screen.

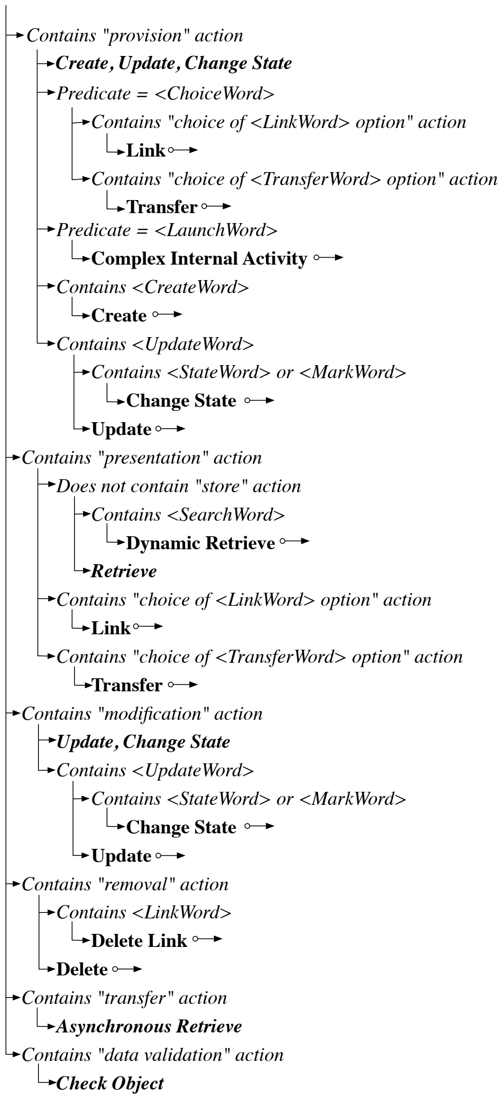

Contains "provision" action
Create, Update, Change State
Predicate = <ChoiceWord>
Contains "choice of <LinkWord> option" action
Link
Contains "choice of <TransferWord> option" action
Transfer
Predicate = <LaunchWord>
Complex Internal Activity
Contains <CreateWord>
Create
Contains <UpdateWord>
Contains <StateWord> or <MarkWord>
Change State
Update
Contains "presentation" action
Does not contain "store" action
Contains <SearchWord>
Dynamic Retrieve
Retrieve
Contains "choice of <LinkWord> option" action
Link
Contains "choice of <TransferWord> option" action
Transfer
Contains "modification" action
Update, Change State
Contains <UpdateWord>
Contains <StateWord> or <MarkWord>
Change State
Update
Contains "removal" action
Contains <LinkWord>
Delete Link
Delete
Contains "transfer" action
Asynchronous Retrieve
Contains "data validation" action
Check Object

Figure 8: A decision tree that is used to determine the semantic type of a transaction.

An example of the outcome of the method is presented in Figure 9. As the result of processing, three transactions were identified for the use case presented in Figure 5.

7. Validation of the proposed methods

The proposed methods for automatic transaction identification were implemented as an extension of the

[4]The transaction T_j succeeds the transaction T_i, if there is an arc $A_k \rightarrow A_l$, where A_k belongs to T_i and A_l belongs to T_j.

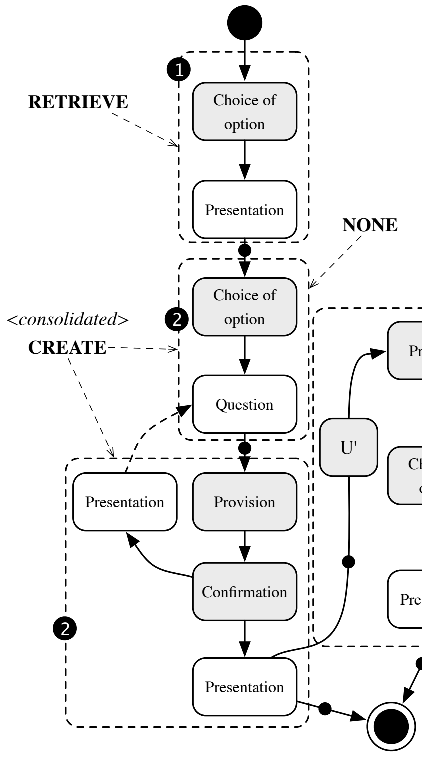


Figure 9: The results of automatic transaction identification based on semantic transaction types for the use case presented in Figure 5.

UCWorkbench tool [12]. We implemented the NLP processing chain for the English language (see Section 6.2), with the use of the Stanford NLP tools [38] (text segmentation, part-of-speech tagger, grammatical relationships, and lemmatizer).

Although, the methods provide objective results, the question arises whether these results "meet the common sense." Otherwise, there could be a problem with interpretation of the results.

Therefore, we decided to validate the proposed methods by comparing their results with the results of human-performed transaction identification. Unfortunately, the number of identified transaction can differ substantially between different people. Therefore, we decided to use the *mean* number of transactions identified by the participants of the experiment as a reference point for the comparison, and perform the analysis of the same benchmark specification containing 34 use cases. In addition, we performed two independent counts using methods M1, M2, and M4, which were performed by two co-authors of the paper to verify whether the mean numbers of transactions are really meaningful. The quantitative results of the comparison are presented in Table 5. The method M3 was excluded from the comparison, because the tool does not directly support it. The number of such transactions is identified using semantic transaction types (M4), as it is not possible to identify the elementary-process-based transactions without understanding the semantics of the interaction within a use case.

It seemed that the on-average numbers of transactions identified by the participants in the experiment are good reference points. The maximum relative error between the counts of experts and the mean numbers of identified transactions by the participants was 12%.

For the stimuli-verbs approach, the implemented methods identified 4% more transactions than the mean number of transactions identified by the participants of the experiment. One of the reason for the difference was that one of the actions was performed outside the system "Candidate performs money transfer (outside the system)." The tool was also more accurate than people in finding decision points of the external actors, what lead to increase in the number of identified transactions.

The same number of transactions was identified by the proposed method when using the "round trip" approach. However, the exact number of transactions identified for certain use cases differed in two cases (the difference was equal to +/-1 transaction). Moreover, the total number of identified transactions was similar to the mean number of transactions identified by the participants of the experiment — the relative error was 3%.

The methods for automatic identification of semantic transactions types, which corresponds to using methods M3 and M4 in the experiment, provided 7% lesser number of transactions than the mean value of the number of transactions identified by the participants. We observed that the automatic method was fragile to multiple changes of states that were realized as multiple, parallel decision points. For instance, in the main flow actor changes the state of the application to "accepted"; and in the alternative flow — preceded by decision point — actor "rejects" the application.

8. Conclusions

The main goal of the paper was to investigate the potential threats to reliability of human-performed transaction identification in use cases, and — if required — to propose the means to improve the objectiveness of the transaction identification process.

The first observation made was that the notion of use-case transactions is vague, as there are at least two different definitions of this concept. The first one comes from Ivar Jacobson — the inventor of use cases. It states

Table 5: Comparison of the number of transactions and UUCW obtained by participants of the experiment, two co-authors of the paper (Expert #1 and Expert #2), and the UCWorkbench tool with the NLP processing chain.

	M1		M2		M4 (M3)	
	#Transactions	UUCW	#Transactions	UUCW	#Transactions	UUCW
Participants-mean	73	186	74	194	58 (53)	178 (175)
Expert #1	73	180	73	175	56	175
Expert #2	69	175	65	170	53	170
UCWorkbench	76	190	76	190	54	170

that use-case transaction consists of the actor's request and a system response. The second one, introduced by Sergey Diev, is based on the elementary process known from Function Point Analysis. Moreover, there are also two additional methods for transactions identification, such as the stimuli-verb approach [5] or semantic transaction types [6].

In this context, the important question is whether there is a cause-effect relationship between the choice of the approach to transaction identification and on-average number of use-case transactions identified by people. In order to investigate this issue we conducted a controlled experiment on the group of 120 students.

The analysis of the experiment data revealed that there can be a significant difference in the median number of transactions identified by people using the "round trip," and elementary-process-based definitions of a use-case transaction. The second observation was made, that the stimuli-verb approach for transaction identification provides similar on-average results as the "round trip" definition. No significant difference in the median number of transactions was observed between Diev's definition of use-case transaction and the semantic transactions types. Therefore, it seems that the stimuli-verb approach is compliant with the "round trip" definition of use-case transaction, and semantic transaction types are compliant with the elementary-process-based definition.

Another threat to reliability of transaction identification in use cases, and use-case-based FSM, was the high intra-method variability. The observed ratios between the maximum and minimum number of identified transactions by participants using the same method, ranged from 1.96 to 3.83. However, one should keep in mind that the participants were students who were familiar with the concept of use cases, but they had no previous experience in identifying use-case transactions.

In order to mitigate the revealed problems related to different approaches to transaction identification and intra-method variability, we proposed a set of methods that can be used to automatically identify transactions in use cases according to each definition of use-case transaction. The proposed methods were implemented as a proof-of-concept tool. The transaction counts provided by the tool were compared with the manual counts done by people — participants in the experiment and FSM experts. The difference in the number of transactions identified by the tool and on-average number of transactions identified by the participants were similar (3–7% depending on the method). In addition, it provides repeatable results, therefore, it mitigates reliability problems that relate to the human factor.

9. Acknowledgments

We would like to thank Sylwia Kopczyńska, Marcin Szeląg, and Przemysław Wesołek for their help in executing the experiment, and the students for their participation in the experiment.

This research project operated within the Foundation for Polish Science Ventures Programme co-financed by the EU European Regional Development Fund.

[1] A. Albrecht, Measuring application development productivity, in: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, 1979, pp. 83–92.

[2] D. Garmus, D. Herron, Function Point Analysis: Measurement Practices for Successful Software Projects, Addison-Wesley Boston, 2001.

[3] C. Symons, Software sizing and estimating: Mk II FPA (Function Point Analysis), Wiley Series In Software Engineering Practice, 1991.

[4] G. Karner, Metrics for objectory. No. LiTH-IDA-Ex-9344:21, Master's thesis, University of Linköping, Sweden (1993).

[5] G. Robiolo, R. Orosco, Employing use cases to early estimate effort with simpler metrics, Innovations in Systems and Software Engineering 4 (1) (2008) 31–43.

[6] M. Ochodek, J. Nawrocki, Enhancing use-case-based effort estimation with transaction types, Foundations of Computing and Decision Sciences 35 (2) (2010) 91–106.

[7] M. Ochodek, J. Nawrocki, K. Kwarciak, Simplifying effort estimation based on Use Case Points, Information and Software Technology 53 (3) (2010) 200–213. doi:10.1016/j.infsof.2010.10.005.

[8] ISO/IEC, ISO/IEC 20968:2002 Mk II Function Point Analysis — Counting Practices Manual (2002).

[9] ISO/IEC, ISO/IEC 20926:2009 Software and systems engineering — Software measurement — IFPUG functional size measurement method 2009 (2009).

[10] R. Collaris, E. Dekker, Software cost estimation using use case points: Getting use case transactions straight, IBM, The Rational Edge.
URL `http://www.ibm.com/developerworks/rational/library/edge/09/mar09/collaris_dekker/`

[11] S. Diev, Software estimation in the maintenance context, ACM SIGSOFT Software Engineering Notes 31 (2) (2006) 1–8.

[12] J. Nawrocki, Ł. Olek, UC Workbench — A tool for writing use cases, in: 6th International Conference on Extreme Programming and Agile Processes, Vol. 3556 of Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 230–234.

[13] I. Jacobson, Use cases – yesterday, today, and tomorrow, Software and systems modeling 3 (3) (2004) 210–220.

[14] IEEE, IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998) (1998).

[15] G. Schneider, J. P. Winters, Applying Use Cases: A Practical Guide, Addison-Wesley, 1998.

[16] J. Ouwerkerk, A. Abran, An evaluation of the design of Use Case Points (UCP), in: A. Abran, R. Dumke, M. Ruiz (Eds.), Proceedings of the International Conference on Software Process and Product Measurement MENSURA 2006, Publish Service of the University of Cádiz www.uca.es/publicaciones, 2006, pp. 83–97.

[17] K. Ribu, Estimating object-oriented software projects with use cases, Master's thesis, University of Oslo, Department of Informatics (2001).

[18] B. Kitchenham, S. Pfleeger, N. Fenton, Towards a framework for software measurement validation, IEEE Transactions on Software Engineering 21 (12) (1995) 929–944.

[19] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison Wesley Longman, Inc, 1992.

[20] A. Cockburn, Writing Effective Use Cases, Addison-Wesley Boston, 2001.

[21] S. Adolph, P. Bramble, A. Cockburn, A. Pols, Patterns for Effective Use Cases, Addison-Wesley, 2002.

[22] G. Övergaard, K. Palmkvist, Use Cases: Patterns and Blueprints, Addison-Wesley, 2005.

[23] K. E. Wiegers, Software Requirements, Microsoft Press, Redmond, WA, USA, 2003.

[24] B. Anda, Comparing effort estimates based on Use Case Points with expert estimates, in: Empirical Assessment in Software Engineering (EASE 2002), Keele, UK, 2002.

[25] B. Anda, H. Dreiem, D. I. K. Sjøberg, M. Jørgensen, Estimating software development effort based on use cases — experiences from industry, in: M. Gogolla, C. Kobryn (Eds.), 4th International Conference on the Unified Modeling Language (UML2001), Lecture Notes in Computer Science, Springer-Verlag, Toronto, Canada, 2001, pp. 487–502.

[26] S. Diev, Use cases modeling and software estimation: Applying Use Case Points, ACM SIGSOFT Software Engineering Notes 31 (6) (2006) 1–4.

[27] T. Fetcke, A. Abran, T. Nguyen, Mapping the OO-Jacobson approach into function point analysis, in: Proceedings of the Technology of Object-Oriented Languages and Systems, 1997. TOOLS 23, 1997, pp. 192–202.

[28] B. Alchimowicz, J. Jurkiewicz, M. Ochodek, J. Nawrocki, Building benchmarks for use cases, Computing and Informatics 29 (1) (2010) 27–44.

[29] M. Wilk, R. Gnanadesikan, Probability plotting methods for the analysis for the analysis of data, Biometrika 55 (1) (1968) 1–17.

[30] S. Shapiro, M. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (3-4) (1965) 591–611.

[31] J. Cohen, Statistical power analysis, Current Directions in Psychological Science 1 (3) (1992) 98–101.

[32] W. Kruskal, W. Wallis, Use of ranks in one-criterion variance analysis, Journal of the American Statistical Association 47 (260) (1952) 583–621.

[33] W. J. Conover, Practical Nonparametric Statistics, 3rd Edition, John Wiley & Sons, 1999.

[34] M. Ochodek, J. Nawrocki, Automatic transactions identification in use cases, in: Balancing Agility and Formalism in Software Engineering: 2nd IFIP Central and East European Conference on Software Engineering Techniques CEE-SET 2007, Vol. 5082 of Lecture Notes in Computer Science, Springer Verlag, 2008, pp. 55–68.

[35] S. Kusumoto, F. Matukawa, K. Inoue, S. Hanabusa, Y. Maegawa, Estimating effort by Use Case Points: method, tool and case study, in: Proceedings of 10th International Symposium on Software Metrics, IEEE, 2004, pp. 292–299.

[36] A. Živkovič, I. Rozman, M. Heričko, Automated software size estimation based on function points using UML models, Information and Software Technology 47 (13) (2005) 881–890.

[37] A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman, Compilers: Principles, Techniques, & Tools, 2nd Edition, Addison-Wesley Pearson Education, Inc., 2007.

[38] M.-C. de Marneffe, C. D. Manning, The Stanford typed dependencies representation, in: Proceedings of the Coling 2008 Workshop on Cross-Framework and Cross-Domain Parser Evaluation, 2008, pp. 1–8.