

Poznan University of Technology  
Faculty of Computer Science and Management  
Institute of Computer Science

Master's thesis

## **AUTOMATIC DETECTION OF DEFECTS IN USE CASES**

Alicja Ciemniewska, Jakub Jurkiewicz

Supervisor  
Jerzy Nawrocki, PhD Dr Habil.

Poznań, 2007

Tutaj przychodzi karta pracy dyplomowej;  
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

## STRESZCZENIE

Wymagania są kluczowym elementem każdego projektu programistycznego. Opisują one porządane cechy i funkcjonalność budowanego systemu. Identyfikacja oraz zbierania wymagań są jednymi z pierwszych czynności wykonywanymi w procesie budowy oprogramowania. Badania wskazują, że co piąty projekt informatyczny zakończy się porażką, a połowa z projektów przekroczy budżet, harmonogram lub też nie będzie zawierała kluczowych funkcjonalności. Te same badania jako główną przyczynę takiego stanu rzeczy podają wymagania o niskiej jakości.

Istniejące na rynku narzędzia pozwalają jedynie na zapis wymagań, jednak nie dostarczają żadnych mechanizmów analizy ich struktury i jakości. Analiza taka musi być przeprowadzana przez ekspertów podczas przeglądów wymagań, co jest czasochłonne i często trudne do przeprowadzenia. Dodatkowo ekspert po wykryciu wielu prostych błędów może pominąć błędy poważniejsze. Dlatego też Adolph zaproponował podział przeglądów na dwie fazy: w pierwszej fazie wymagania są przeglądane w celu znalezienia prostych do wykrycia defektów (może to wykonywać asystent eksperta, sekretarka, itp.), w drugiej fazie ekspert może się już skupić na istotnych błędach. Takie podejście pozwala na poprawę jakości wymagań, jednakże nadal jest ono pracochłonne.

Niniejsza praca skupia się na próbie automatyzacji wykrywania prostych błędów w wymaganiach. Tak wspomagany proces zbierania wymagań usprawniłby pracę analityka, pozwoliłby na uniknięcie wielu powszechnie występujących błędów, dzięki czemu tworzone wymagania byłyby znacznie wyższej jakości.

Jedną z najpopularniejszych metod zapisu wymagań są przypadki użycia zaproponowane przez Jackobsona. Wykorzystanie języka naturalnego przy tworzeniu przypadków użycia jest intuicyjne, nie wymaga dodatkowej wiedzy technicznej oraz przyspiesza i upraszcza proces zbierania wymagań. Dobrze napisany przypadek użycia jest czytelny i zrozumiały dla klienta. Ponadto wspomaga komunikację oraz prowadzi do osiągnięcia wspólnego punktu widzenia przez klienta i analityka, co pozwala uniknąć błędów w fazie pozyskiwania wymagań.

Pisanie przypadków użycia nie jest czynnością trywialną, wymaga wyczucia i odpowiedniego doświadczenia. Nie istnieje też żaden ściśle określony standard pisania przypadków użycia, co w konsekwencji prowadzi do tworzenia specyfikacji wymagań niskiej jakości. Dodatkowo każdy wytwórca oprogramowania ma własną politykę i konwencję zapisu wymagań. W wielu publikacjach pojawiają się jednak pewne uniwersalne wskazówki jak tworzyć efektywne przypadki użycia, tak aby spełniały swój główny cel, czyli opis wymagań klienta. Zgodnie z tymi wskazówkami można sprawdzać jakość przypadków użycia szukając w nich defektów świadczących o niskiej jakości.

Do automatycznego wykrywania defektów można zastosować metody przetwarzania języka naturalnego, które pozwalają przeanalizować budowę lingwistyczną zdań. Analiza taka daje możliwość sprawdzania zarówno poprawności językowej, jak i poprawności z punktu widzenia zalecanych dobrych praktyk. Wykrycie defektu zwróciłoby uwagę analityka, że dany fragment można poprawić i zapisać w sposób bardziej przejrzysty. Automatyczne wykrywanie defektów w przypadkach użycia może zatem pomóc w podniesieniu jakości zbieranych wymagań, wyrobieniu dobrych nawyków, a co za tym idzie zwiększyło by efektywność pracy analityka oraz całego zespołu.

W ramach pracy zaproponowano metody automatycznego wykrywania defektów w przypadkach użycia. Zaproponowane metody zostały zaimplementowane jako moduły narzędzia do zbierania wymagań (*UC Workbench*). W pracy ograniczono się do wymagań zapisanych w formie przypadków użycia w języku angielskim.

*The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements(...) No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later.*

- Frederick Brooks, Jr [F.P95]

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Software Quality . . . . .	1
1.2	Aim and scope . . . . .	2
1.3	Acknowledgements . . . . .	2
<b>2</b>	<b>Natural Language Processing</b>	<b>4</b>
2.1	Natural language processing problems . . . . .	4
2.2	Basic terms of NLP . . . . .	5
2.2.1	Parts of speech . . . . .	5
2.2.2	Phrase structures . . . . .	5
2.2.3	Grammatical dependencies . . . . .	7
2.3	NLP fields and tools . . . . .	8
2.4	Parsing with the Stanford parser . . . . .	9
<b>3</b>	<b>UC Workbench and Eclipse Platform</b>	<b>11</b>
3.1	UC Workbench . . . . .	11
3.2	Eclipse Platform . . . . .	12
<b>4</b>	<b>Software defects and concept of the solution</b>	<b>15</b>
4.1	Specification-Level Bad Smells . . . . .	15
4.2	Use-Case Level Bad Smells . . . . .	16
4.2.1	Repeated Actions in Neighbouring Steps (RANS) . . . . .	17
4.2.2	Inappropriate Naming (IN) . . . . .	18
4.3	Step-Level Bad Smells . . . . .	18
4.3.1	Too Complex Sentence Structure (TCSS) . . . . .	18
4.3.2	Lack of the Actor (LOA) . . . . .	19
4.3.3	Misusing Tenses and Verb Forms (MTVF) . . . . .	19
4.3.4	Using Technical Jargon (UTJ) . . . . .	19
4.3.5	Conditional Steps (CS) . . . . .	20
<b>5</b>	<b>Description of implementation</b>	<b>22</b>
5.1	Technologies and Tools . . . . .	22
5.2	Architecture . . . . .	22
5.3	Details of implementation . . . . .	23
5.3.1	Data model . . . . .	23
	Verb Form, Noun Form . . . . .	26
5.3.2	Business Logic . . . . .	26
	Extension point for parsers . . . . .	27

Parsers . . . . .	28
Extension points for analysers . . . . .	29
Analysers . . . . .	30
Additional mechanisms . . . . .	31
5.3.3 User Interface . . . . .	31
Editor Annotations . . . . .	31
Preferences Page . . . . .	31
Action . . . . .	33
5.4 Testing . . . . .	33
<b>6 Evaluation of the solution</b>	<b>35</b>
6.1 Evaluation metrics . . . . .	35
6.2 Results of the experiment . . . . .	36
6.2.1 Specification level . . . . .	36
6.2.2 Use case level . . . . .	37
6.2.3 Step level . . . . .	38
6.3 Performance analysis . . . . .	40
<b>7 Conclusions</b>	<b>42</b>
<b>A Analysed use cases and analysis results</b>	<b>43</b>
A.1 dLibra CRM use cases . . . . .	43
A.1.1 UC1.1 Log in to the system . . . . .	44
A.1.2 UC1.2 Add a new client . . . . .	44
A.1.3 UC1.3 Edit client data . . . . .	45
A.1.4 UC1.4 Delete client . . . . .	47
A.1.5 UC1.5 Browse clients . . . . .	48
A.1.6 UC1.6 Search . . . . .	48
A.1.7 UC1.7 Add a new contract . . . . .	49
A.1.8 UC1.8 Edit contract . . . . .	50
A.1.9 UC1.9 Delete contract . . . . .	51
A.1.10 UC1.10 Add a new installation . . . . .	51
A.1.11 UC1.11 Edit installation . . . . .	52
A.1.12 UC1.12 Delete installation . . . . .	53
A.1.13 UC1.13 Prepare a report . . . . .	54
A.1.14 UC2.1 Log in to the system . . . . .	55
A.1.15 UC2.2 Browse information . . . . .	55
A.1.16 UC2.3 Request for licence . . . . .	55
A.2 Mobile News use cases . . . . .	57
A.2.1 UC1 Post a group message . . . . .	57
A.2.2 UC2 Configure the server . . . . .	58
A.2.3 UC3 Add a new channel group . . . . .	58
A.2.4 UC4 Add a new channel . . . . .	59
A.2.5 UC5 Delete a channel . . . . .	60
A.2.6 UC6 Delete a channel group . . . . .	61
A.2.7 UC7 Delete a user group . . . . .	62
A.2.8 UC8 Update news . . . . .	62

A.2.9	UC9 Delete news . . . . .	63
A.2.10	UC10 Register a new user . . . . .	63
A.2.11	UC11 Download news . . . . .	64
A.2.12	UC12 Run the application . . . . .	65
A.2.13	UC13 Subscribe/unsubscribe news channels . . . . .	65
A.2.14	UC14 Configure user preferences . . . . .	66
A.2.15	UC15 Read news . . . . .	66
A.3	Web JSARA use cases . . . . .	67
A.3.1	UC1 Find data . . . . .	68
A.3.2	UC2 Show statistics . . . . .	68
A.3.3	UC3 Create user account . . . . .	69
A.3.4	UC4 Modify user account . . . . .	70
A.3.5	UC5 Delete user account . . . . .	71
A.3.6	UC6 Create data version . . . . .	71
A.3.7	UC7 Edit data file . . . . .	72
A.3.8	UC8 Database search . . . . .	73
A.3.9	UC9 Upload data . . . . .	74
A.3.10	UC10 Delete data . . . . .	74
A.3.11	UC11 View data . . . . .	75
A.3.12	UC12 Download file . . . . .	76
A.3.13	UC13 RegisterUser . . . . .	76
A.3.14	UC14 Log on . . . . .	77
A.3.15	UC15 Log off . . . . .	77
A.3.16	UC16 Run simulation . . . . .	78
A.3.17	UC17 Simulation time . . . . .	79
A.3.18	UC18 Run applet . . . . .	79
A.3.19	UC19 Create publication . . . . .	80
A.3.20	UC20 Modify publication . . . . .	81
A.3.21	UC21 Delete publication . . . . .	82
A.3.22	UC22 View content . . . . .	82
A.3.23	UC23 Create level . . . . .	83
A.3.24	UC24 Modify level . . . . .	84
A.3.25	UC25 Delete level . . . . .	85
A.3.26	UC26 Add thread . . . . .	86
A.3.27	UC27 AddPost . . . . .	86
A.3.28	UC28 Moderate . . . . .	87
A.3.29	UC29 Quote . . . . .	88
<b>B</b>	<b>Attachments</b>	<b>90</b>
B.1	CD . . . . .	90
<b>C</b>	<b>Division of the performed tasks</b>	<b>91</b>
	<b>Bibliography</b>	<b>92</b>

# Chapter 1

## Introduction

### 1.1 Software Quality

In every serious software project there must be a place for requirements, which describe desired characteristics of the software being developed. Identifying and documenting the requirements are the first activities in most software processes.

The role of requirements in a successful IT project cannot be underrated. According to the Standish Group report [Gro04] almost one of five software projects will never reach successful finish. Furthermore over half of the projects will be over time, over budget and/or will lack critical features and requirements. In the same report it is stated that one of the main reasons of this problems is rooted in the requirements. Other research [NS03] indicate that over 60% of engineers claim that their projects failed due to vague, inconsistent or incomplete requirements. Research conducted by HP shows that if 1\$ is taken as a theoretical cost to fix a defect detected in the requirements phase then the same defect fixed in the design phase would cost up to 3\$, 6\$ in the development phase, 20\$ in the testing phase and even 100\$ after the release[HP99].

In the beginning of the requirements elicitation phase the analyst deals with acquiring information from different sources. Commonly used techniques of obtaining knowledge are interviews with the customer and reading available user documentation, which are heavily based on natural language (NL) [LMI03]. Natural language is well known, generally easy to understand and does not require any additional skills. Therefore use cases [Jac04], which consist of simple sentences written in a natural language, are the most popular form of recording requirements (over 50% of analysts employ use cases in the requirements elicitation phase, according to [NL03]). However, using NL can lead to a diversity of meaning, vagueness and redundancies, etc. It is crucial for the analyst to omit mentioned disadvantages of natural language during the recording of requirements. Therefore recently a number of studies have been proposed to use linguistic tools to support requirements analysis. Fantechi et al. [FGL03] proposed an approach that provides mechanisms for discovering relations between actors and use cases which constitute the requirements specification. These derived relations can be analysed in order to find anomalies, incompleteness and potential problems. Although mentioned capabilities, this concept can be applied after the requirement specification is completed and therefore it cannot be used in the early stage of requirements elicitation. Another method for requirements analysis is to restrict the scope of the language that is used [MP93]. Restricting the language means limiting a set of available words in the lexicon, reducing possible syntactic constructions and allowing only a certain semantic theory to interpret the sentence. From the research [OM96] it appeared that this approach allows to detect and resolve the typical syntactic and semantic ambiguities. However, the language may be so restricted

that it may become irritating and unnatural to use. Moreover the user needs to acquire additional knowledge about the restricted language to use it effectively.

From the above examples it can be observed that the quality of requirements has an enormous influence on the success of the software projects and additionally there is no sufficient number of tools for supporting requirements analysis. Therefore reviews (initially introduced by Weinberg [Wei71]) remains one of the most popular [NL03] ways of improving the quality of software requirements. Fagan [Fag86] was the one who proposed a complete process that enables the detection and removal of defects in software artifacts. The process involves a group of experts that determines whether the artifact has sufficient quality. Knight and Myers [JM93] described an improved inspection technique, which splits reviews into phases. Every phase is performed by people tailored to the purpose of the phase. The first stages focus on detecting easy-to-detect defects and the aims of the later stages are to find major defects. Similar approach, proposed by Adolph et al. [ABCP02], suggests to divide the review process into internal and external parts. The goal of the internal review is to check the readability, implementability, precision and accuracy; this phase is performed many times by small, internal team. The external review is usually done once by all the stakeholders and allows analysing the requirements in depth from the system standpoint. Although the advantages, it has to be noticed that reviews are expensive, time consuming and require a lot of work and effort.

## 1.2 Aim and scope

Effectiveness of reviews could be significantly improved by a system which could discover easy-to-detect defects automatically. The aim of this thesis is to develop a method for detecting requirements defects in an automatic way. The following goals were distinguished to achieve the aim:

- propose and construct a spike solution with Natural Language Processing tools to check whether these tools are mature enough to apply them to analyse requirements expressed as use cases,
- implement the developed method within *UC Workbench* environment [NO05],
- evaluate the developed approach.

This thesis focuses on requirements in a form of use cases, as they consist of simple structure sentences, which are easy to analyse with available Natural Language Processing tools.

In the next section basic terms and tools concerning Natural Language Processing are described. *UC Workbench* - a tool for requirements elicitation and management together with the employed form of use cases are presented in Section 3. Distinguished defects that can be detected automatically and the methods of detection are described in Section 4. Section 5 encloses detailed description of the implemented solution. Results, discussed in Section 6, shows the evaluation of the developed approach. Finally the conclusions are presented.

## 1.3 Acknowledgements

The authors of this thesis would like to thank the supervisor, professor Jerzy Nawrocki, for the inspiration, knowledge and time spend on the problem presented in this thesis.

Additionally the authors are grateful to Łukasz Olek, the manager of *UC Workbench* project, for the consultations and useful advice.

The authors are indebted to the members of *UC Workbench* team, who were always very helpful and supportive.

The authors are also thankful to Bartosz Walter for allowing the access to the Software Requirements Specifications of the projects developed as part of the Software Development Studio at the Poznan University of Technology. This data was used to evaluate the presented spike solution.

## Chapter 2

# Natural Language Processing

*Natural language processing (NLP)* studies the problems of automated generation and understanding of natural languages. Since the half of the twentieth century NLP techniques have been significantly developed due to the need of human - computer communication and human attention has turned to the automation of natural language processing. Computers are expected to assist people in intellectual effort by correcting text documents, decision supporting, extracting important information from large sets of raw textual data, etc. These demands encouraged researchers to investigate possibilities of developing complex NLP systems.

### 2.1 Natural language processing problems

Natural language processing is complicated and encounters numerous problems presented below:

- The set of words used by NLP system may not contain all the terms (or all meanings of the terms) occurring in the text being processed. It may lead to a situation when NLP system fails to process the text or an incorrect analysis of a sentence is performed.
- The sentence may be ambiguous. Following types of text ambiguity can be distinguish:

- *Categorial ambiguity* - more than one *part of speech* can be associated with a word. For example in the sentence:

*Time flies like an arrow.*

the word *time* may be interpreted as a noun, verb or adjective

- *Lexical ambiguity* - a word has more than one possible meaning. For example in the following sentence:

*John went to the bank.*

the word *bank* could mean an edge of a river or an institution.

- *Syntactical ambiguity* - the syntax, or grammar, can be understood in more than one way. The following sentence:

*The cotton clothing is usually made of grows in Mississippi.*

can be interpreted in two different ways. Firstly it can be understood that *clothing* is made in *Mississippi* and secondly that the *grows* come from *Mississippi*.

- *Referential ambiguity* - more than one object can be referred to by a *noun phrase*

*George gave John a bike and he said thanks.*

In the above sentence it can be either *George* or *John* who expresses his gratitude. The context of the sentence may be crucial for determining the sense of the sentence.

- *Ellipsis* - incomplete sentence where missing item is not clear

*The dog chased the mouse. A cat too.*

It can be considered whether the *mouse* is chased together by the *dog* and the *cat* or whether the *dog* chases both the *mouse* and the *cat*.

- The analysed sentence might be considered as grammatically correct, but from logical point of view it may have no sense. An example of such sentence was given by Chomsky in [Cho57]:

*Colourless green ideas sleep furiously.*

Such sentence would be accepted by the NLP system, but it may introduce inconsistency to the analysis.

Because of this disadvantages a practical NLP system must be good at coping with ambiguities of word sense, word category, syntactic structure and semantic scope.

## 2.2 Basic terms of NLP

Basic linguistic concepts necessary for complete understanding of the solution described in this thesis are presented in the following subsections.

### 2.2.1 Parts of speech

Words within a language are grouped into sets, which show similar syntactic behaviour, called *parts of speech (POS)*. *Noun*, *verb* and *adjective* are three most important parts of speech. *Noun* answers the questions: *who?* and *what?* and represents people, things and concepts. *Verb* describes actions and states in a sentence. *Adjective* characterises *nouns*.

*Morphology* is a discipline which deals with patterns of word-formation within and across languages. Knowledge of morphological rules, used to build new words in a particular language (e.g. English), makes it possible to predict *POS* of an unknown word occurring in a text. Therefore, a small set of basic forms of words is enough to determine *part of speech* of a word in a sentence.

*Brown Corpus* [WNF64], compiled in 1964 by Kucera and Francis, was initially a set of American English words extracted from a wide variety of sources. Later, *POS* tags were added to the Brown Corpus to mark about 80 parts of speech, compound forms, contractions and foreign words. *Penn Treebank* [Pen] [MS99] bases on the simplified Brown's tagset constructed to eliminate redundancy by taking into account both lexical and syntactic information. A set of tags, both from Brown Corpus and Penn Treebank, used in this thesis is presented in Table 2.1.

Additionally the syntactic structure in the Penn Treebank can be represented in many different ways, for example using simple labelled brackets in a text file, like presented in Figure 2.1.

### 2.2.2 Phrase structures

Words in a sentence cannot occur in any order. Every language defines rules concerning word order. According to these rules words are organised into grouped units called *phrases*. These units are used to construct more complex structures, hence the whole sentence. However, a *phrase* cannot constitute a standalone sentence. An example of *phrase* structure (in a form of a tree) is presented

TABLE 2.1: Tags from Brown Corpus and from Penn Treebank.

Category	Example	Tags	
		Brown Corpus	Penn Treebank
adjective	happy, bad	JJ	JJ
adjective, ordinal number	sixth, 72nd, last	OD	JJ
adjective, comparative	sixth, happier, worse	JJR	JJR
adjective, superlative	happiest, worst	JJS	JJS
adjective, cardinal number	3, fifteen	CD	CD
adverb	often, particularly	RB	RB
adverb, question	when, how, why	WRB	WRB
adverb, degree, question	how, however	WQL	WRB
conjunction, coordination	and, or	CC	CC
conjunction, subordinating	although, when	CS	IN
conjunction, complementizer	that	CS	IN
determiner	this, each, another	DT	DT
determiner, pronoun	any, some	DTI	DT
determiner, pronoun, plural	these, those	DTS	DT
determiner, article	the, a , an	AT	DT
determiner, postdeterminer	many, some	AP	JJ
determiner, possessive	their, yours	PP\$	PRP\$
noun	aircraft, data	NN	NN
singular common noun	woman, book	NN	NN
plural common noun	women, books	NNS	NNS
noun, proper, singular	Warsaw, John	NP	NNP
noun, adverbial	home, yesterday	NR	NN
pronoun, nominal (indefinite)	everything, none, two	PN	NN
pronoun, personal, subject	you, we	PPSS	PRP
pronoun, personal, subject, 3SG	he, she, it	PPS	PRP
pronoun, personal, object	you, them, me	PPO	PRP
pronoun, question, subject	who, whoever	WPS	WP
verb, base present form (not infinitive)	arrive, give	VB	VBP
verb, infinitive	arrive, give	VB	VB
verb, past tense	arrived, gave	VBD	VBD
verb, present participle	arriving, giving	VBG	VBG
verb, past/passive participle	arrived, given	VBN	VBN
verb, present 3SG -s form	arrives, gives	VBZ	VBZ
verb, auxiliary, do , base	do	DO	VBP
verb, auxiliary, have, base	have	HV	VBP
verb, auxiliary, be, infinitive	be	BE	VP
verb, modal	can, will, should	MD	MD
preposition, to	to	IN	TO
preposition	for, about	IN	IN
possessive	's	S	POS

```
(S
  (NP (DT The) (NN man))
  (VP (VBD heard)
    (NP
      (NP (NNS dogs))
      (VP (VBG barking)
        (PP (IN in)
          (NP (DT the) (NN distance.)))))))
```

FIGURE 2.1: Example of using Penn Treebank.

in Figure 2.2. The root ( $S$ ) of the structure represents the whole sentence divided into two main phrases listed below:

- *Noun phrase (NP)* gathers information about the *noun*, the head of NP, which participate in an activity (or state) described by the *verb*. Apart from the *noun*, NP usually contains also: a *determiner*, an *adjective phrase*, and/or *propositional phrase*. Exemplary compound NP is underlined in the sentence below:

The rich young man living next door heard dogs barking in the distance.

- *Verb phrase (VP)* organises all elements syntactically dependent on the *verb*, which is the head of VP.

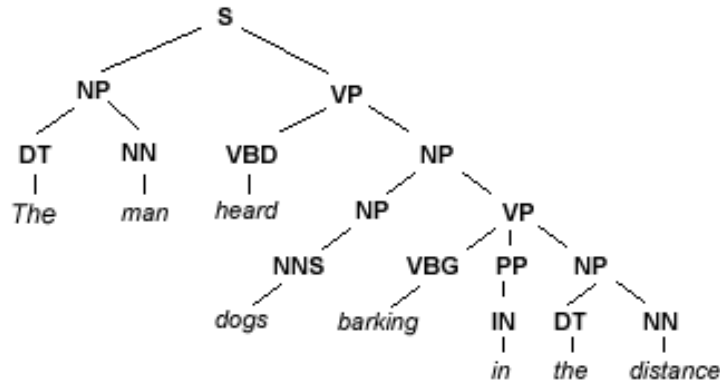


FIGURE 2.2: Example of using Penn Treebank (phrase structure).

On the contrary to *phrase*, *clause* is a group of related words that must contain both a *subject* (a NP that appears before the *verb*) and a *predicate* (the rest of the sentence apart from the *subject*, which must contain a *verb* and other sentence elements supplementary to the *verb*). Two primary types of clauses can be distinguished:

- An *independent clause* expresses a complete thought and it can stand by itself.
- A *dependent clause*, also known as a *subordinate clause*, expresses only part of a thought and it cannot stand alone.

A complex sentence may be composed of different types of *phrases* and *clauses* (which description is out of the scope of this thesis) that constructs a hierarchical structure.

In some languages (e.g. Latin, Polish), called *free words order languages*, the order of words in a *phrase* does not influence the meaning. However, English is more restrictive and it forces specific order of words to indicate concrete sense. This feature of English significantly simplifies the processing of *POS*, types of phrases and clauses in the NLP systems.

### 2.2.3 Grammatical dependencies

Apart from phrase structures presented in previous subsection, *typed dependencies* are another way of representing the structure of a sentence. In every sentence words exist in grammatical relationships with each other. For instance in the sentence:

*A dog is chasing a cat around the table.*

*nouns* *dog* and *cat* are arguments of the *verb* *chase*. The NP *dog* is a *subject* as it appears before the *verb* and the NP *a cat* is an *object* as it occurs after the *verb*. The *typed dependencies* represents the mentioned grammatical relations between individual words. Carroll et al. [CMB99] proposed a hierarchical structure (presented in Figure 2.3) of *typed dependencies*, which are commonly used by NLP systems. Following typed dependencies are used in this thesis (based on: [CB00]):

- *nsubj(head, dependent)* - relation between a *predicate* (head) and a *nominal subject* (dependent), which is a *noun phrase*
- *nsubjpass(head, dependent)* - relation between *predicate* (head) with *passive verb* and *nominal passive subject* (dependent), which is a *noun phrase*.

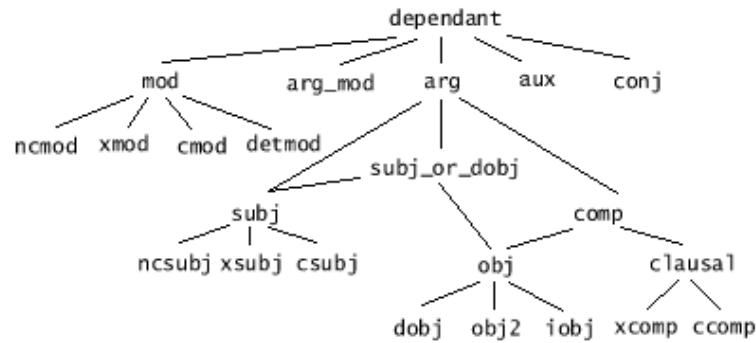


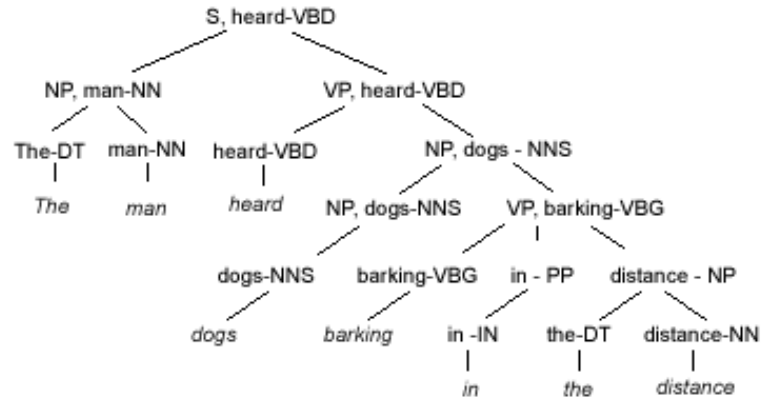
FIGURE 2.3: The grammatical dependencies hierarchy.

- $obj(head, dependent)$  - relation between a *predicate* (head) and its *object* (dependent)
- $dobj(head, dependent)$  - relation between a *predicate* (head) and its *direct object* (dependent)
- $iobj(head, dependent)$  - relation between a *predicate* (head) and its *indirect object* (dependent)
- $pobj(head, dependent)$  - relation between a *predicate* (head) and its *object* (dependent), which is the head of a *noun phrase* following the preposition
- $conj(type, head+)$  - relation between elements (heads) connected by a *conjunction* word, type indicated the type of the *conjunction*
- $expl(head, dependent)$  - relation, which captures an existential *there*

## 2.3 NLP fields and tools

Recently Natural Language Processing techniques have been significantly developed and improved mainly due to fast growth of the Internet, ipso facto, increasing amount of textual information. The list of most dynamically explored fields of NLP is presented below:

- *POS tagging* - analyses human text and assigns appropriate *part of speech* to each word depending on its definition and a context in which the word occurs. Although *POS tagging* may seem one of simpler NLP domains, it has to deal with different types of text ambiguity, as it is often an entry point for further analysis.
- *Parsing* - processes the text in order to determine its grammatical structure, which is usually presented in a form of a tree called *parse tree*. Statistical approach is commonly used by modern parsers. Such parsers learn (using e.g. maximum entropy, neural network) from the set of training data which is manually tagged and later the acquired knowledge is used to process the text being analysed. Parsers that employ additional lexical statistics (considering identities of words and parts of speech) are deemed to be the most effective.
- *Machine Translation (MT)* - investigates the use of software to translate text or speech from one natural language to another. The basic approach, which can be applied in MT, is to perform a simple substitution of words in one natural language for words in another. However, taking under consideration the complexity of a grammar such simple approach is not efficient enough. Therefore more advanced methods, like corpus analysis, have been developed to support better handling of phrase recognition, differences in linguistic typology, translation of idioms and isolation of anomalies.

FIGURE 2.4: Combined structure from the *Stanford parser*.

- *Text summarisation* - considers automatic creation of a shortened version of raw text written in natural language. Text summarisation uses heuristic, linguistic and statistical methods. The basic concept is to count how often certain key words occurs in a text, in which sentences and where the sentences are placed in the text. This approach selects the information (e.g. key words, key clauses or sentences) which seems to be the most important and simply copy it to the summary. More advanced method is to paraphrase parts of the original text which condense a text more strongly, but it requires Natural Language generation techniques.

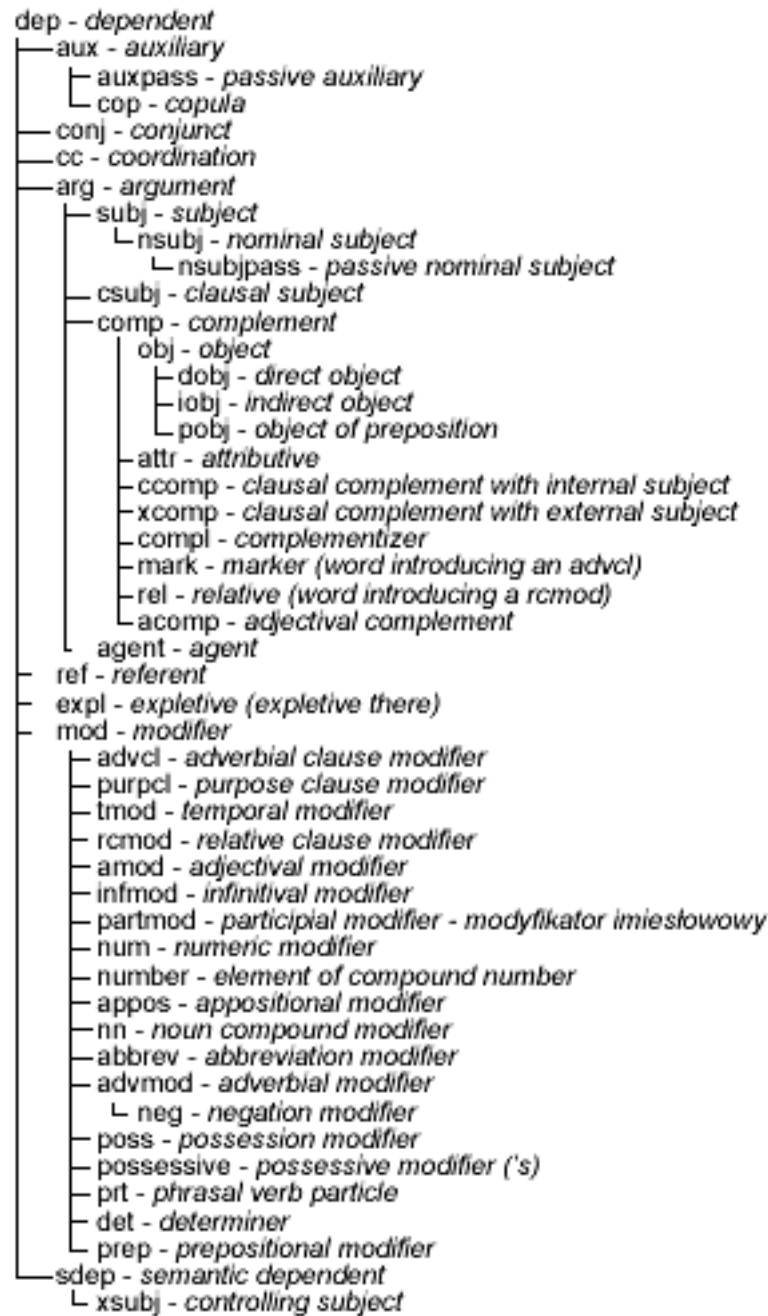
## 2.4 Parsing with the Stanford parser

Parsing, which allows to analyse text on the sentence level, seems to be the most suitable for the purpose of this thesis. Therefore the *Stanford parser* [sta] was chosen to build the spike solution of the proposed defect detection methods. Main features and capabilities of the *Stanford parser* are specified below:

- it provides comprehensive analysis of sentences through generation of two different lexical structures:
  - combined structure [KM02] - a *lexicalized phrase-structure*, which carries both category and *part of speech* tagged head word information at each node (Figure 2.4)
  - *typed dependencies* [dMMM06] - presents information about relations between words; the structure differs from the one proposed by Carroll and is presented in Figure 2.5
- it is possible to train the parser by providing annotated data
- the input can be preliminary marked with *part of speech* tags in order to force the parser to use these tags
- both the binary version and the source code are available
- the parser is written in *Java* programming language and provides API for external usage

Despite presented advantages, following drawbacks of the *Stanford parser* can be distinguished:

- parser memory usage expands roughly with the square of the sentence length
- the parser does not always deal with the ambiguities of words

FIGURE 2.5: The dependencies hierarchy used by *Stanford parser*.

Although the disadvantages, the mentioned capabilities make the *Stanford parser* a very useful tool that may be applied in a variety of NLP contexts.

## Chapter 3

# UC Workbench and Eclipse Platform

### 3.1 UC Workbench

*UC Workbench* [ucw] is a tool for requirements elicitation and management, which has been developed at *Poznan University of Technology* for last three years. The author of *UC Workbench* is Łukasz Olek, who initiated it as his master thesis. Since that time the system has dynamically developed as a set of *Eclipse Platform* plugins and in 2005 was awarded IBM Eclipse Innovation Grant by IBM company. Additionally a number of Polish IT companies show an interest in the system and use it as a primary tool for requirement elicitation. Looking at the requirements from different perspectives together with the feedback from the industry allows to conduct research concerning a variety of aspects of requirements engineering. Therefore recently numerous bachelor, master and doctoral thesis have been strongly connected with *UC Workbench*.

The main purpose of *UC Workbench* is to assist the software analyst during writing and editing requirements in a form of use cases. Three types of use-case forms can be distinguished:

- *use case brief* - shortened version of a use-case consisting of two to six sentences mentioning only the most significant activity and failures
- *casual use case* - a use case written in simple style that includes a main scenario with extensions
- *fully dressed use case* - exhaustive description according to a specified pattern which contains additional information like preconditions, postconditions, triggers, secondary actors, etc.

*UC Workbench* applies the second form of use cases written with the use of *FUSE language* [NO05]. Additionally *UC Workbench* comes with other useful modules, which enable the analyst to:

- create screen sketches and attach them to the steps of use cases
- generate a *mockup* of the system being build
- estimate the effort of the software project
- conduct an asynchronous reviews of the requirements

Screen sketches are simple drafts of the elements of future user interface. These drafts can be easily created in *UC Workbench* with the *Screen Sketch Editor* and later they can be attach to the steps of the use case. The exemplary screenshot of *Screen Sketch Editor* is presented in Figure 3.1. To enhance the customers' imagination about future system the mockup can be generated. *Mockup* is a simple HTML prototype consisting of use cases and screen sketches, which

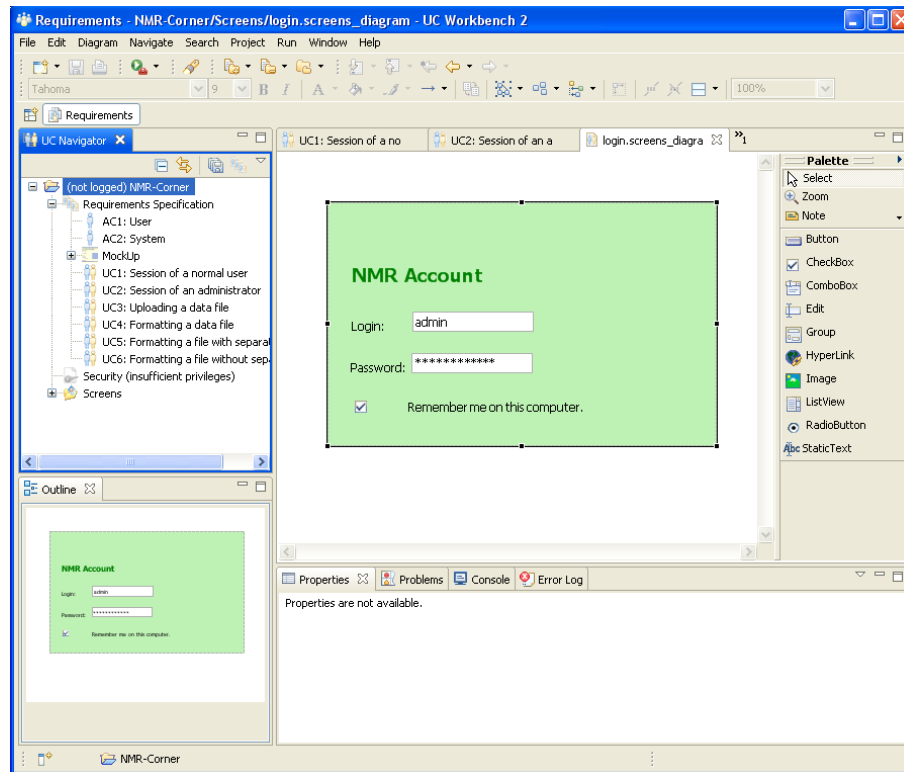


FIGURE 3.1: Screen sketches module of UC Workbench.

supports navigation through requirements specification. Such *mockup* is not bothersome as it does not require any additional tools except from web browser. The screenshot of a *mockup* is presented in Figure 3.2. Time and effort estimation is an important part of software development process as it enables to determine the budget and the project plan. *AutoEC* is an effort calculator integrated into UC Workbench which allows to assess required effort on the base of UC-PlaG method [Och06]. Figure 3.3 presents the screenshots of the *AutoEC* module.

The aim of asynchronous reviews module [Mic06] is to allow reviewers to work remotely. The review is conducted by experts through the website and the results are available for analyst in UC Workbench for further studies. These approach allows to detect mainly major defects, therefore the method to detect easy-to-detect defects is still needed to improve the quality of requirements.

## 3.2 Eclipse Platform

*Eclipse* is a universal tool platform, an open extensible *integrated development environment (IDE)*. *IDE* consists of a variety of tools, extensible frameworks and runtimes for developing and managing software.

The extensibility of the *Eclipse Platform* results from two main elements:

- *plugins* - modules with new functionality which is added to main environment
- *extension points* - a special point where a new module can plug into the platform

*Eclipse* itself consist of a set of plugins which make a number of *extension points* accessible for new *plugins*. The Figure 3.4 presents the overall architecture of the *Eclipse Platform*. *Eclipse* provides uniform mechanisms and user interface for all integrated *plugins*. User has the impression

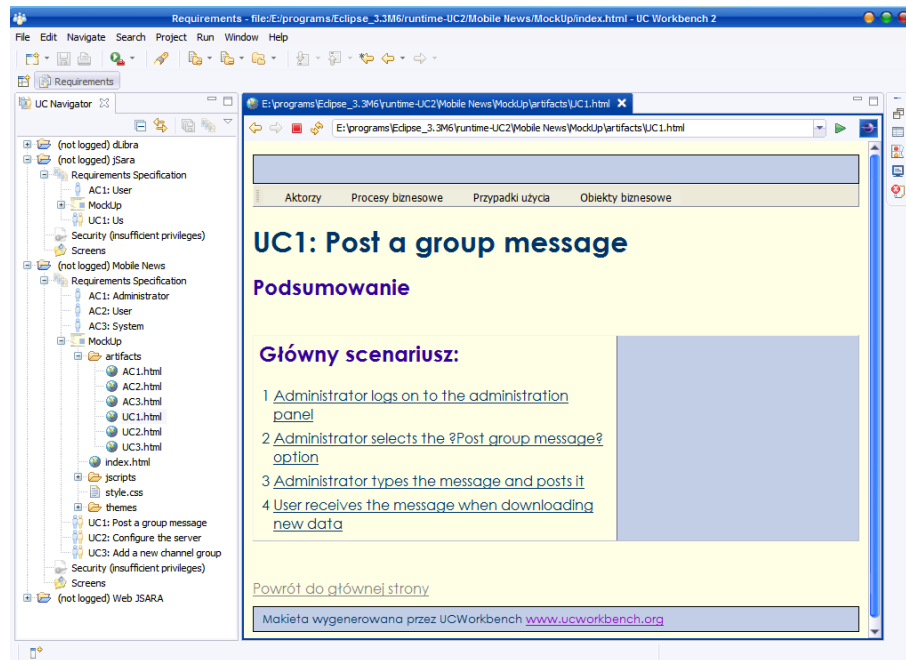


FIGURE 3.2: Mockup module of UC Workbench.

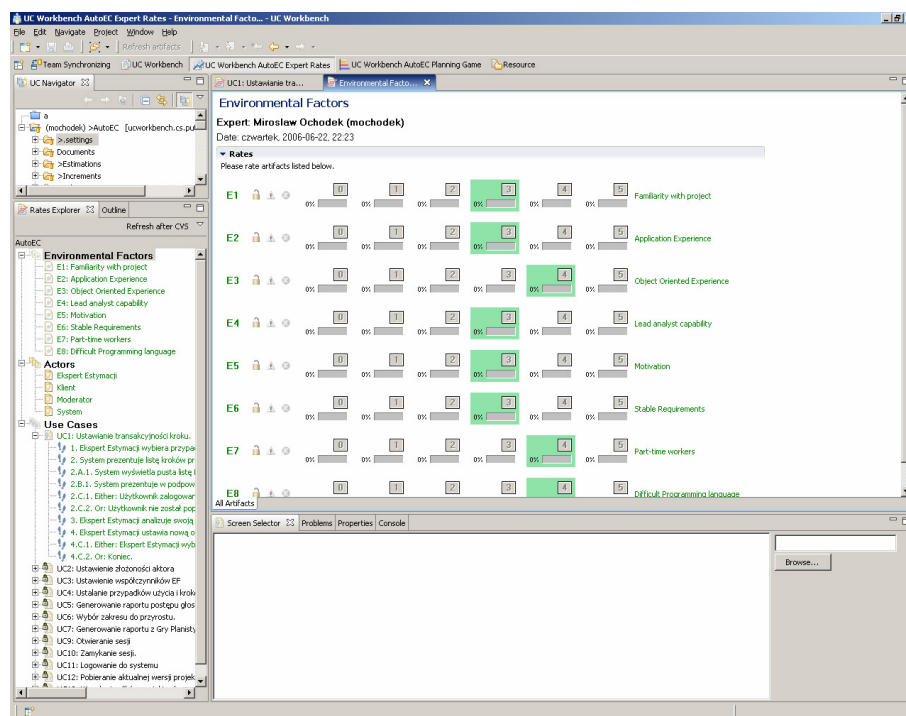


FIGURE 3.3: Effort estimation module (AutoEC) of UC Workbench.

of using one comprehensive product instead of many small tools. While the *Eclipse Platform* serves as an open tools platform, it is designed so that its elements could be used to construct any client application. The minimal set of plugins that is enough to build a rich client application is known as the *Rich Client Platform (RCP)*.

Many projects and programming technologies are based on *Eclipse*. The list of the projects used in *UC Workbench* is presented below:

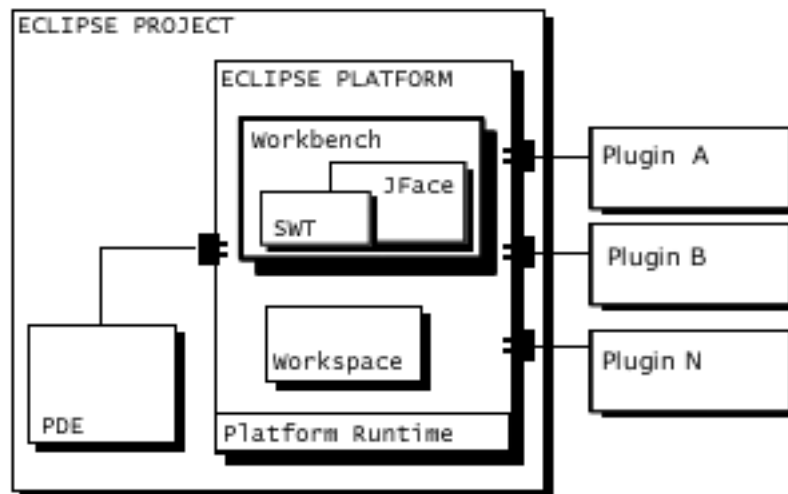


FIGURE 3.4: Overall architecture of the *Eclipse Platform*.

- *Java Development Tools (JDT)* - complete environment for development of applications in *Java* programming
- *Plug-in Developer Environment (PDE)* - extension of *JDT* with facilities for developing plugins and *extension points*
- *Standard Widget Toolkit (SWT)* [SWT] - open source widget toolkit for *Java*. It enables efficient and portable access to the user-interface facilities of the operating systems on which it is implemented.
- *JFace* [JFa] - user interface toolkit (based on *SWT*) with classes for handling many common UI programming tasks.
- *Eclipse Modelling Framework (EMF)* [EMFa] - modelling framework and code generation facility for building applications based on structured data model
- *EMF Transaction* [EMFc] - provides a model management layer built on top of *EMF* for managing *EMF* resources
- *Graphical Editing Framework (GEF)* - framework for creation of a rich graphical editor
- *Graphical Modelling Framework (GMF)* - framework for developing graphical editors based on *EMF* and *GEF*

## Chapter 4

# Software defects and concept of the solution

Adolph [ABCP02] and Cockburn [Coc01] presented a set of guidelines and good practices about how to write effective use cases. In this context "effective" means clear, cohesive, easy to understand and maintain. Reading the guidelines one can distinguish several types of defects in use cases. In this chapter we present those defects that can be automatically detected. Each defect discussed in this Section contains a description of automatic detection method. They have been split into three groups presented in separate subsections: specification-level bad smells (those are defect indicators concerning a set of use cases), use-case level bad smells (defect indicators concerning a single use case), and step-level bad smells (they concern a step - a use cases consists of many steps).

### 4.1 Specification-Level Bad Smells

At the level of requirements specification, where there are many use cases, a quite common defect which we have observed is *Use-Case Duplication (UCD)*. Surprisingly, we have found such defects even in specifications prepared by quite established Polish software houses. The most frequent is duplication by information object. If there are two different information objects (e.g. an invoice and a bill), analysts have a tendency to describe the processes which manipulate them as separate use cases, even if they are processed in the same way. That leads to unnecessary thick documentation (the thicker the document, the longer the time necessary to read it). Moreover, it is dangerous. When someone finds and fixes a defect in one of the use cases, the other will remain unchanged, what can be a source of problems in the future. There are two sources of duplicated use cases:

- **Intentional duplication.** An analyst prefers that style and/or he wants to have a thick document (many customers still prefer thick documents - for them they look more serious and dependable, which is of course a myth). Some of such analysts perhaps will ignore that kind of warning, but some other - more proactive - may start to change their style of writing specification.
- **Unintentional duplication.** There are several analysts, each of them is writing his own part of the specification and before the review process no one is aware of the duplication.

---

<sup>1</sup>This chapter has been published as [CJNO07]

If this is the case, the ability to find duplicates in an automatic way will be perceived as very attractive.

**Detection method** is two-phased. In the first stage a signature (finger print) of each use case is computed. It can be a combination of a main actor identifier (e.g. its number) and a number of steps a use case contains. Usually a number of steps in a use case and a number of actors in the specification are rather small (far less than 256), thus a signature can be a number of the integer type. If two use cases have the same signature, they go through the second stage of analysis during which they are examined step by step. Step number  $j$  in one use case is compared against step number  $j$  in the second use case and so-called step similarity factor,  $s$ , is computed. Those similarity factors are combined into use-case similarity factor,  $u$ . If  $u$  is greater than a threshold then two use cases are considered similar and a duplication warning is generated.

A very simple implementation of the above strategy can be the following. We know that two similar use cases can differ in an information object (a noun). Moreover, we assume that most important information about processing an information object is contained in verbs and nouns. Thus, step similarity factor,  $s_i$ , for steps number  $i$  in the two compared use cases can be computed in the following way:

- If *all the corresponding verbs and nouns* appearing in the two compared steps *are the same*, then  $s_i = 1$ .
- If all the corresponding verbs are the same and *all but one corresponding nouns are the same* and a difference in the two corresponding nouns has been observed for the first time, then  $s_i = 1$  and InfObject1 is set to one of the "conflicting" nouns and InfObject2 to the other (InfObject1 describes an information object manipulated with the first use case and InfObject2 is manipulated with the second use case).
- If all the corresponding verbs are the same and *all but one corresponding nouns are the same* and the conflicting nouns are InfObject1 in the first use case and InfObject2 in the second, then  $s_i = 1$ .
- In all other cases,  $s_i$  for the two analysed steps is 0.

Use-case similarity factor,  $u$ , can be computed as a product of step similarity factors:

$$s_1 * s_2 \dots * s_n$$

The described detection method is oriented towards *intentional duplication*. To make it effective in the case of *unintentional duplication* one would need a dictionary of synonyms. Unfortunately, so far we do not have any.

## 4.2 Use-Case Level Bad Smells

Bad smells presented in this section are connected with the structure of a single use case. The following bad smells have been selected to detect them automatically with *UC Workbench*, a use-case management tool developed at the Poznan University of Technology:

- **Too long or too short use cases. (LSUC)** It is strongly recommended [ABCP02] to keep use cases 3-9 steps long. Too long use cases are difficult to read and understand. Too short use cases, consisting of one or two steps, distract a reader from the context and, as well, make the specification more difficult to understand. To detect that bad smell it is enough to count the number of steps in each use case.

**Main Scenario:**

1. System switches to on-line mode and displays summary information about data that have to be uploaded and downloaded.
2. User confirms action.
3. System executes action.

**Extensions:**

- 1.A. TMS is unreachable.
  - 1.A.1. System shows information that there is no connection to TMS.
- 1.B. There is no data to synchronize.
  - 1.B.1. System shows information that no data have to be synchronized.
  - 1.B.2. End of use case.
- 2.A. TMS does not recognize user's login and password.
  - 2.A.1. System displays information about the problem and shows the login form.
  - 2.A.2. User fills the form.
  - 2.A.3. System saves new data.
  - 2.A.4. Go to step 2.

FIGURE 4.1: Example of a use case with too complicated extension.

**Wrong:**

- ~~1. Administrator fills in his user name~~
- ~~2. Administrator fills in his telephone number~~
- ~~3. Administrator fills in his email~~

**Correct:**

*Administrator fills in his user name, telephone number and email*

FIGURE 4.2: Example of repeated actions in neighbouring steps.

- **Complicated extension. (CE)** An extension is designed to be used when an alternative course of action interrupts the main scenario. Such an exception usually can be handed by a simple action and then it can come back to the main scenario or finish the use case. When the interruption causes the execution of a repeatable, consistent sequence of steps, then this sequence should be extracted to a separate use case (Figure 4.1). Detection can be based on counting steps within each extension. A warning is generated for each extension with too many steps.
- **Repeated actions in neighbouring steps.**
- **Inappropriate naming**

The last two bad smells will be described in the subsequent subsections.

**4.2.1 Repeated Actions in Neighbouring Steps (RANS)**

Every step of a use case should represent one particular action. The action may consist of one or more moves which can be taken as an integrity. Every step should contain significant information which rather reflect user intent than a single move. Splitting these movements into separate steps may lead to long use cases, bothersome to read and hard to maintain.

**Wrong:** *Title: Main Use Case*

**Correct:** *Title: Buy a book*

FIGURE 4.3: Example of Inappropriate naming.

**Wrong:** *The system switches to on-line mode and displays summary information about data that have to be uploaded and downloaded*

**Correct:** *The system displays list of user's tasks*

FIGURE 4.4: Examples of too complex sentence structure.

**Detection method:** Check whether several consecutive steps have the same actor (subject) and action (predicate). Extraction of subject and predicate from the sentence is done by the *Stanford parser*. The analyst can be informed that such sequence of steps can be combined to a single step.

#### 4.2.2 Inappropriate Naming (IN)

Every use case should have a descriptive name. The title of each use case presents a goal that the primary actor wants to achieve. There is a few conventions of naming use cases, but it is preferable to use active verb phrase in the use case name. Furthermore, chosen convention should be used consistently in all us cases.

**Detection method:** Approximated method of bad smell detection in use case names, is to check whether use case name satisfies the following constraints:

- The title contains a verb in infinitive (base) form
- The title contains an object

This can be done using the *Stanford parser*. If the title does not fulfil these constraints, a warning is attached to the name.

### 4.3 Step-Level Bad Smells

Bad smells presented in this section are connected with use-case steps. Steps occur not only in the main scenario, but also in extensions. The following bad smells are described here:

#### 4.3.1 Too Complex Sentence Structure (TCSS)

The structure of a sentence used for describing each step of use case should be as simple as possible. It means that it should generally consists of a subject, a verb, an object and a prepositional phrase ([ABCP02]). With such a simple structure one can be sure that the step is grammatically correct and unambiguous. Because of the simplicity, use cases are easy to read and understand by readers. Such a simple structure helps a lot when using natural language tools. It is essential to notice that the simpler the sentence is, the more precisely other bad smells can be discovered. An example of a too complex sentence and its corrected version is presented below.

**Detection method:** Looking at the use cases that were available to us we have decided to build a simple heuristic that checks whether a step contains more than one sentence, subject or predicate, coordinate clause, or subordinate clause. If the answer is YES, then a warning is

**Wrong:** *The form is filled in*

**Correct:** *Student fills in the form*

FIGURE 4.5: Example of lack of the actor.

generated. Obviously, it is just a heuristic. It can happen that a step contains one of the mentioned defect indicator, but it is still readable and easy to understand. Providing a clear and correct answer in all possible cases is impossible - even two humans can differ in their opinion what is simple and readable. In our research we have encountered some examples of this problem. However, we have distinguished some rules, which can be used to verify, whether a sentence is too complex and unreadable. Below we present the verification rules. The numbers in the brackets show applicability of a rule (in how many use cases a given rule could be applied) and its effectiveness (in how many use cases it has found a real defect approved by a human).

- step contains more than one sentence (2 / 2)
- step contains more than one subject or predicate (2 / 2)
- step contains more than one coordinate clause (8 / 4)
- step contains more than one subordinate clause (7 / 5)

#### 4.3.2 Lack of the Actor (LOA)

According to [ABCP02] it should be always clearly specified who performs the action. The reader should know which step is performed by which actor. Thus, every step in a use case should be an action that is performed by one particular actor. Actor's name ought to be the subject of the sentence. Thus, in most cases the name should appear as the first or second word in the sentence. Omission of actor's name from the step may lead to a situation in which the reader does not know who is the executor of the action.

**Detection method:** In the case of *UC Workbench*, every actor must be defined and each definition is kept within the system. Therefore it can be easily verified whether the subject of a sentence describing a step is defined as an actor.

#### 4.3.3 Misusing Tenses and Verb Forms (MTVF)

A frequent mistake is to write use cases from the system point of view. This is easier for the analyst to write in such a manner, but the customer may be confused during reading. Use cases should be written in a way which is highly readable for everyone. Therefore the action ought to be described from the user point of view. In order to ensure this approach, the present simple tense and active form of a verb should be used. Moreover, the present simple tense imply that described action is constant and the system should always respond in a determined way.

**Detection method:** Using combined structure from the *Stanford parser*, the *nsubj* relation [dMMM06] can be determined. It can be checked if the subject is an actor and the verb is in the third person singular active form.

#### 4.3.4 Using Technical Jargon (UTJ)

Use case is a way of describing essential system behaviour, so it should focus on the functional requirements. Technical details should be kept outside of the functional requirements specification.

**Wrong:** ~~Send the message~~  
**Wrong:** ~~System is sending an email~~  
**Wrong:** ~~The email is sent by System~~  
**Correct:** System sends an email

FIGURE 4.6: Example of misusing tenses and verb forms.

**Wrong:** ~~User chooses the second tab and marks the checkboxes~~  
**Correct:** User chooses appropriate options

FIGURE 4.7: Example of using technical jargon.

**Wrong:**  
~~1. Administrator types in his user name and password~~  
~~2. System checks if the user name and the password are correct~~  
~~3. If the given data is correct the system logs Administrator in~~

**Correct:**  
 1. Administrator types in his user name and password  
 2. System finds that the typed-in data are correct and logs Administrator in

**Extensions:**  
 2.A. The typed-in data are incorrect  
 2.A.1. System presents an error message

FIGURE 4.8: Example of conditional steps.

Using technical terminology might guide developers towards specific design decisions. Graphical user interface (GUI) details clutter the story, make reading more difficult and the requirements more brittle.

**Detection method:** We should create a dictionary containing terminology typical for specific technologies and user interface (e.g. button, web page, database, edit box). Then it is easy to check whether the step description contains them or not.

#### 4.3.5 Conditional Steps (CS)

A sequence of actions in the use case depends on some conditions. It is natural to describe such situation using conditionals *if condition then action...* This approach is preferred by computer scientists, but it can confuse the customer. Especially it can be difficult to read when nested *if* statement is used in a use case step. Use cases should be as readable as possible. Such a style of writing makes it complex, hard to understand and follow.

It is preferable to use the optimistic scenario first (main scenario) and to write alternative paths separately in the extension section.

**Detection method:** The easiest way to detect this bad smell is to look for specific keywords, such as *if*, *whether*, *when*. Additionally, using the *Stanford parser* it can be checked that the found

keyword is a subordinating conjunction. In such a case the analyst can be notified that the step should be corrected.

## Chapter 5

# Description of implementation

As a part of this thesis a spike implementation was provided to evaluate the proposed defects detection methods. The implementation is integrated with *UC Workbench* to enhance the quality of the requirements.

### 5.1 Technologies and Tools

During the implementation technologies strictly connected with *Eclipse platform* were mainly employed. Below the list of used technologies and tools together with short descriptions is presented:

- *Java* - programming language originally developed by Sun Microsystems. The main concepts of Java are:
  - object-oriented programming (OOP) paradigm
  - platform independence
  - automatic garbage collection
  - security

*Java* code is typically compiled to bytecode and bytecode is usually either interpreted or compiled to native code for execution at runtime.

- *Eclipse Modelling Framework (EMF)* [EMFa] - described in Section 3.2
- *EMF Transaction* [EMFc] - described in Section 3.2
- *Emfatic* [Emfb] - language for representing EMF Ecore models in textual form
- *Standard Widget Toolkit (SWT)* [SWT] - described in Section 3.2
- *JFace* [JFa] - described in Section 3.2
- *JUnit* [JUn] - framework to write repeatable unit tests for the Java programming language
- *EasyMock* [eas] - open source library, which provides mock objects for interfaces in *JUnit* tests

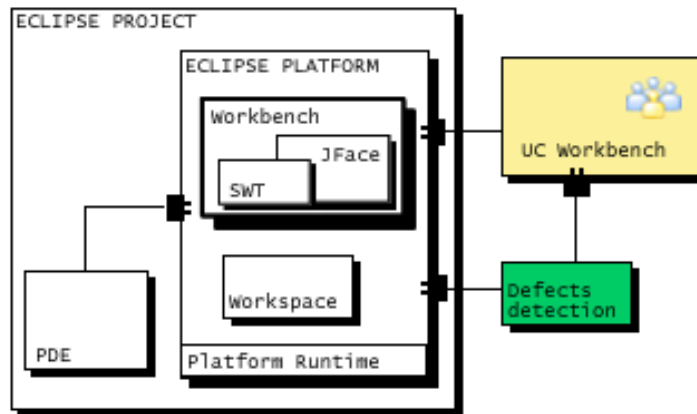


FIGURE 5.1: Overall architecture of the *Eclipse platform*, *UC Workbench* and developed plugins.

## 5.2 Architecture

As said before the proposed spike solution was implemented as a part of *UC Workbench* environment, thus it was created as a plugin within the *Eclipse platform*. The Figure 5.1 presents the overall architecture of *Eclipse platform* together with *UC Workbench* and plugins developed for the purpose of this thesis.

Following plugins were created:

- `org.ucworkbench2.bs.model` - main model with classes for annotating defects
- `org.ucworkbench2.bs.model.tests` - unit tests for the classes from the main model
- `org.ucworkbench2.annotations.parsers` - classes responsible for processing and analysing use cases
- `org.ucworkbench2.annotations.parsers.tests` - unit tests for classes from `org.ucworkbench2.annotations.parsers` plugin
- `org.ucworkbench2.annotations.parsers.ui` - classes responsible for graphical user interface

## 5.3 Details of implementation

### 5.3.1 Data model

The plugin `org.ucworkbench2.bs.model` includes the data model for the defects found in the analysed requirements specification. For every single defect there is a specific interface (with corresponding implementation class) which annotates the elements of the requirements specification. Three interfaces were created for more general view of defects:

- `BadSmell` - represents a general defect found in the requirements specification
- `UseCaseBadSmellAnnotation` - represents use case level defects
- `StepBadSmellAnnotation` - represents step level defects

Following interfaces were created to represent specific defects:

1. Requirement specification defects:

- **DuplicatedUseCasesAnnotation** represents duplicated use-cases defect. Methods of this interface are following:
  - `getDuplicatedUseCases()` - returns the references to duplicated use cases

2. Use case defects

- **InappropriateUseCaseLengthAnnotation** - represents inappropriate use-case length defect. Methods of this interface are following:
  - `getStepsNumber()` - returns the number of steps in the use case with encountered defect
  - `setStepsNumber(int value)` - sets the number of steps in the use case with encountered defect
- **InappropriateUseCaseNameAnnotation** - represents inappropriate name of use-case defect. Methods of this interface are following:
  - `getForm()` - returns the form of the verb used in the name of a use case
  - `setForm(VerbForm value)` - sets the form of the verb used in the name of a use case
  - `getVerbOffset()` - returns the offset of the verb found in the name of a use case
  - `setVerbOffset(int value)` - sets the offset of the verb found in the name of a use case
  - `getVerbLength()` - returns the length of the verb string found in the name of a use case
  - `setVerbLength(int value)` sets the length of the verb string found in the name of a use case
  - `getNumberOfVerbs()` - returns the number of verbs found in the name of a use case
  - `setNumberOfVerbs(int value)` - sets the number of verbs found in the name of a use case
- **RepeatingActionsAnnotation** - represents defect of repeating actions in a use-case. Methods of this interface are following:
  - `getStepsNumbers()` - returns the numbers of steps, which contains repeating actions
- **TooLongExtensionAnnotation** - represents too long extension scenario defect. Methods of this interface are following:
  - `getStepsNumber()` - returns the number of steps in a use case with encountered defect
  - `setStepsNumber(int value)` - sets the number of steps in a use case with encountered defect

3. Step defects

- **ComplexSentenceStructureAnnotation** - represents complicated sentence structure defect. Methods of this interface are following:
  - `getNumberOfSubjects()` - returns number of found subjects in a sentence
  - `setNumberOfSubjects(int value)` - sets number of found subjects in a sentence
  - `getNumberOfPredicates()` - returns number of predicates found in a sentence

- `setNumberOfPredicates(int value)` - sets number of predicates found in a sentence
- `getNumberOfClauses()` - returns number of clauses found in a sentence
- `setNumberOfClauses(int value)` - sets number of clauses found in a sentence
- `getNumberOfVerbPhrases()` - returns number of verb phrases found in the sentence
- `setNumberOfVerbPhrases(int value)` - sets number of verb phrases found in the sentence
- `getParseTreeDepth()` - returns the parse tree depth of a sentence
- `setParseTreeDepth(int value)` - sets the parse tree depth of a sentence
- `ConditionalStepAnnotation` - represents conditional structure defect.
- `InappropriatePredicateFormAnnotation` - represents inappropriate verb form defect. Methods of this interface are following:
  - `getForm()` - returns the form of the verb
  - `setForm(VerbForm value)` - sets the form of a verb
- `InappropriateSentenceStructureAnnotation` - represents inappropriate sentence structure defect.
- `JargonBadSmellAnnotation` - represents using technical jargon defect.
- `LackOfActorAnnotation` - represents lack of actor defect.
- `LackOfObjectAnnotation` - represents lack of object defect.
- `LackOfPredicateAnnotation` - represents lack of predicate defect.
- `LackOfSubjectAnnotation` - represents lack of subject defect.
- `PassiveVoiceBadSmellAnnotation` - represents verb in passive voice defect.

Additionally following interfaces (with corresponding implementation) were distinguished to annotate parts of speech, parts of sentence, dependencies and model elements found in the analysed sentences:

#### 1. Language annotations

- `LanguageAnnotation` - represents general language annotation. Methods of this interface are following:
  - `getDescription()` - returns the description of the annotation
  - `setDescription(String value)` - sets the description of the annotation
- `ConditionAnnotation` - represent the conditional structure found in a sentence
- `JargonAnnotation` - represent the jargon word found in a sentence
- `SentenceStructureAnnotation` - represent the details about the structure of a sentence. Methods of this interface are following:
  - `getNumberOfClauses()` - returns number of clauses found in a sentence
  - `setNumberOfClauses(int value)` - sets number of clauses found in a sentence
  - `getNumberOfVerbPhrases()` - returns number of verb phrases found in the sentence
  - `setNumberOfVerbPhrases(int value)` - sets number of verb phrases found in the sentence
  - `getParetreeDepth()` - returns the parse tree depth of a sentence

- `setParsetreeDepth(int value)` - sets the parse tree depth of a sentence
- **NounAnnotation** - represent the noun in the sentence. Methods of this interface are following:
  - `getForm()` - returns the form of the noun
  - `setForm(NounForm value)` - sets the form of the noun
- **VerbAnnotation** - represent the verb in the sentence. Methods of this interface are following:
  - `getForm()` - returns the form of the verb
  - `setForm(VerbForm value)` - sets the form of the verb
- **PartOfSentenceAnnotation** - represents general part of speech annotations
- **SubjectAnnotation** - represents the subject in the sentence.
- **PredicateAnnotation** - represents the predicate in the sentence.
- **ObjectAnnotation** - represents the object in the sentence.

## 2. Model annotations

- **ActorAnnotation** - represents an actor performing an action in a use case step
- **BusinessObjectAnnotation** - represents a business object used in a use-case step

## Verb Form, Noun Form

In order to indicate forms of nouns and verbs found in the text following constants were provided:

- constants describing noun forms:
  - `SINGULAR_COMMON` - represents singular common noun
  - `PLURAL_COMMON` - represents plural common noun
  - `SINGULAR_PROPER` - represents singular proper noun
  - `PLURAL_PROPER` - represents plural proper noun
- constants describing verb forms:
  - `BASE` - represents verb in base present form
  - `INFINITIVE` - represents verb in infinitive form
  - `PAST_TENSE` - represents verb in past tense form
  - `PRESENT_PARTICIPLE` - represents verb in present participle form
  - `PAST_PARTICIPLE` - represents verb in past participle form
  - `PRESENT_THIRD_SINGULAR` - represents verb in present third singular (-s) form
  - `MODAL_VERB` - represents modal verb

### 5.3.2 Business Logic

The plugin `org.ucworkbench2.annotations.parsers` includes all classes responsible for business logic. The key parts of this plugin, responsible for the process of analysing requirements, are the following classes:

- **EventListener** - manages the overall process of analysing requirements

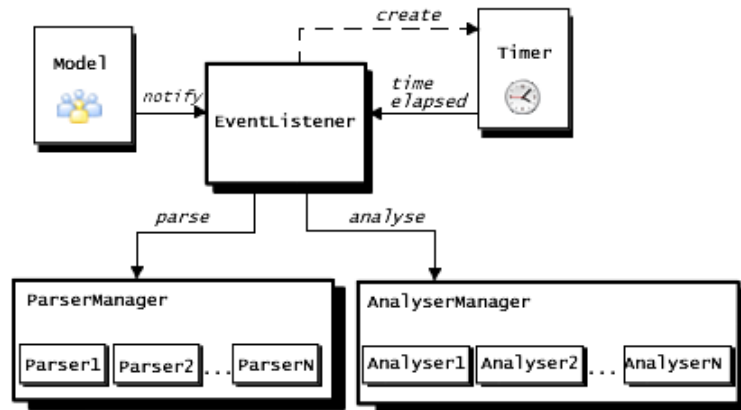


FIGURE 5.2: The overall process of defects detection.

- `TimerThread` - counts the time from the last change of the model and informs the `EventListener` about the elapsed time
- `ParsersManager` - manages all the parsers extracted from the extension point `org.ucworkbench2.annotations.parsers.newAnnotationParser`
- `AnalysersManager` - manages all the analysers extracted from extension points: `org.ucworkbench2.annotations.parsers.newRequirementsSpecificationAnalyser`, `org.ucworkbench2.annotations.parsers.newUseCaseBadSmellAnalyser`, `org.ucworkbench2.annotations.parsers.newStepBadSmellAnalyser`

The schema of the process is presented in Figure 5.2. `EventListener` listens for changes in the model and when notification about new change comes it starts up the further analysis according to the following rules (introduced to increase the performance):

- the changed element differs from the previous one
- the specified time from the last change elapsed

When the changed element does not differ from the previous one, `EventListener` interrupts (if exists) already running `TimerThread` and creates a new instance to start the counting from the beginning. If any of the conditions for starting the further analysis is fulfilled then `EventListener` firstly runs all the available parsers and then all the available analysers.

### Extension point for parsers

Extension point `org.ucworkbench2.annotations.parsers.newAnnotationParser` was designed to simplify addition of new parsers. To define new parser on the basis of this extension point, following elements have to be set:

- *id* - identifier of the parser
- *name* - descriptive name of the parser
- *class* - class which implements the interface `IParserHandler` and which contains the logic of the parser
- *priority* - priority of the parser

As said above every new parser must define a class which implements the interface `IParserHandler`. Methods of this interface are following:

- `startProcessing(Notification notification)` - process the given `Notification` and returns `boolean` value indicating if processing was successful
- `processAnnotatedText(AnnotatedText annotatedText)` - process the given `AnnotatedText` (containing `String` value to be parsed) and returns `boolean` value indicating if processing was successful
- `getPriority()` - returns the priority of the parser in the form of `String`
- `getPriorityNumber()` - returns the priority of the parser as an `integer` value
- `setPriority(String priority)` - sets the priority of the parser

Extraction of available parsers is the main responsibility of class `ParsersManager`. Additionally `ParsersManager` provides methods for running all the parser according to their priorities.

### Parsers

Parsers can be divided into two groups. The first group deals with the language aspects of the text (these parsers are located in the `org.ucworkbench2.annotations.parsers.language` package). Parsers from the second group (located in `org.ucworkbench2.annotations.parsers.model` package) analyse the text using information available in the model. The former use the *Stanford parser* to acquire knowledge about the text structure. On the basis of this knowledge new annotations are created and attached to the text being analysed. The latter process the text without any additional tools, however, they use the native mechanisms of Java programming language. Following parsers were provided:

- `PartOfSpeechParser` - uses the parse tree provided by *Stanford parser* and retrieves information about part of speech of individual words
- `PartOfSentenceParser` - analyses a collection of the typed dependencies and creates annotation depending on the information from the analyses structure. Mappings between the typed dependency and the part of sentences annotations are presented below:
  - `nsubj(head, dependent)` to `PredicateAnnotation` (head) and `SubjectAnnotation` (dependent)
  - `nsubjpass(head, dependent)` to `PassivePredicateAnnotation` (head) and `SubjectAnnotation` (dependent)
  - `obj(head, dependent)` to `ObjectAnnotation` (dependent)
  - `dobj(head, dependent)` to `ObjectAnnotation` (dependent)
  - `iobj(head, dependent)` to `ObjectAnnotation` (dependent)
  - `pobj(head, dependent)` to `ObjectAnnotation` (dependent)
  - `conj(type, head+)` - to `SubjectAnnotation` or `PredicateAnnotation` or `PassivePredicateAnnotation` (heads) depending on the type of other heads
  - `expl(head, dependent)` to `ExpletiveAnnotation` (dependent)
- `StructureParser` - uses the parse tree created by *Stanford parser* to collect statistics about tree depth, number of clauses, number of verb phrases and occurrence of conditional words

- **JargonParser** - detects the occurrence of jargon on the basis of dictionaries defined in extension point `org.ucworkbench2.annotations.parsers.newJargonDictionary`. A new jargon dictionary definition has to provide:
  - *description* - describes the dictionary
  - *filePath* - point to the file with the list of words
- **ActorParser** - marks the occurrence of actor name (from the model) in the analysed text
- **BusinessObjectParser** - marks the occurrence of business object name (from the model) in the analysed text

### Extension points for analysers

Following extension points were designed to simplify addition of new analysers:

- `org.ucworkbench2.annotations.parsers.newRequirementsSpecificationAnalyser` for new analysers on the requirements specification level. Every analyser of this type has to be described by:
  - *id* - identifier of the analyser
  - *name* - descriptive name of the analyser
  - *class* - class which implements the interface `IRequirementsSpecificationAnalyser` and which contains the logic of the analyser
  - *priority* - priority of the analyser

The interface `IRequirementsSpecificationAnalyser` declares method `findBadSmell(RequirementSpecification requirementsSpecification)`, which analyses the `RequirementSpecification` from the point of view of requirements specification level defects.

- `org.ucworkbench2.annotations.parsers.newUseCaseBadSmellAnalyser` for new analysers on the use case level. Every analyser of this type has to be described by:
  - *id* - identifier of the analyser
  - *name* - descriptive name of the analyser
  - *class* - class which implements the interface `IUseCaseBadSmellAnalyser` and which contains the logic of the analyser
  - *priority* - priority of the analyser

The interface `IUseCaseBadSmellAnalyser` declares method `findBadSmell(UseCase useCase)`, which analyses the `UseCase` from the point of view of use-case level defects.

- `org.ucworkbench2.annotations.parsers.newStepBadSmellAnalyser` for new analysers on the step level. Every analyser of this type has to be described by:
  - *id* - identifier of the analyser
  - *name* - descriptive name of the analyser
  - *class* - class which implements the interface `IStepBadSmellAnalyser` and which contains the logic of the analyser

- *priority* - priority of the analyser

The interface `IStepBadSmellAnalyser` declares method `findBadSmell(AnnotatedText annotatedText)`, which analyses the `AnnotatedText` from the point of view of step level defects.

All the interfaces used in the mentioned extension points for new analyser extends `IBadSmellAnalyser` interface which declares the following methods:

- `findBadSmell(Notification notification)` - analyses the given `Notification` and returns `boolean` value indicating if any defect was found during the analysis
- `getPriority()` - returns the priority of the analyser in the form of `String`
- `getPriorityNumber()` - returns the priority of the analyser as an `integer` value
- `setPriority(String priority)` - sets the priority of the analyser

### Analysers

According to the methods presented in Section 4 and extension points described above three groups of analysers were developed:

- requirements specification analysers:
  - `DuplicatedUseCaseAnalyser`
- use-case analysers:
  - `InappropriateUseCaseLengthAnalyser`
  - `RepeatingActionsAnalyser`
  - `TooLongExtensionAnalyser`
  - `NamingAnalyser`
- step analysers:
  - `LackOfActorAnalyser`
  - `LackOfObjectAnalyser`
  - `LackOfPredicateAnalyser`
  - `LackOfSubjectAnalyser`
  - `ConditionalAnalyser`
  - `JargonAnalyzer`
  - `PassiveVoiceAnalyser`
  - `PredicateFormAnalyser`
  - `SentenceStructureAnalyser`

### Additional mechanisms

Although a lot of advantages, the *Stanford parser*, which was used for natural language analysis of use cases, has got one important drawback - it is time and memory consuming. Due to this fact, following mechanisms were implemented:

- Every instance of the *Stanford parser* loads the dictionary and stores it during the whole lifetime of the instance. Class *LexicalizedParserProxySingleton*, which holds the instance of the *Stanford parser*, was developed according to the *singleton* design pattern [GHJV94] to avoid the creation of many instances of the parser to improve the performance.
- Memory usage of the *Stanford parser* expands roughly with the square of the sentence length. In order to be ensure that the analyser will not crash because of the lack of memory, the length of the analysed sentences was limited to 80 characters.
- From the experience with the Stanford Parser it was observed that it consumes a lot of time and memory to process sentences with brackets inside. As the information in the brackets is usually not essential, the brackets are cut out from the sentence before the sentence is being passed to the *Stanford parser*.

#### 5.3.3 User Interface

Classes concerning user interfaces are located in the plugin `org.ucworkbench2.annotations.parsers.ui`. Two main parts of user interface, included in the presented spike solution, can be distinguished. The first part is responsible for informing the user about the detected defect and the second allows the user to set appropriate preferences concerning defects detection.

#### Editor Annotations

When new annotation, representing a defect, is added to any element of the requirement specification then the user is informed about this by a special mark in the *UC Workbench* editor. Firstly the part of the use case, in which the defect was detected, is marked (Figure 5.3). Secondly when the user hovers the mouse cursor over the error or info sign then the message with detailed information about the defect is displayed (Figure 5.4).

Classes responsible for displaying information about detected defect are located in the following packages of the `org.ucworkbench2.annotations.parsers.ui` plugin:

- `org.ucworkbench2.annotations.parsers.ui.adapters.step`
- `org.ucworkbench2.annotations.parsers.ui.adapters.usecase`

Extension point `org.ucworkbench2.ui.texteditor.annotationAdapters` from the project `org.ucworkbench2.ui.texteditor`, which is a part of *UC Workbench*, was used to create the classes for displaying defect annotations in the *UC Workbench* editor.

#### Preferences Page

Preference page (presented in the Figure 5.5) allows the user to set specific preferences concerning detection of the defects. With the mentioned preference page the user is able to:

- enable / disable defects detection mechanism (Figure 5.6)

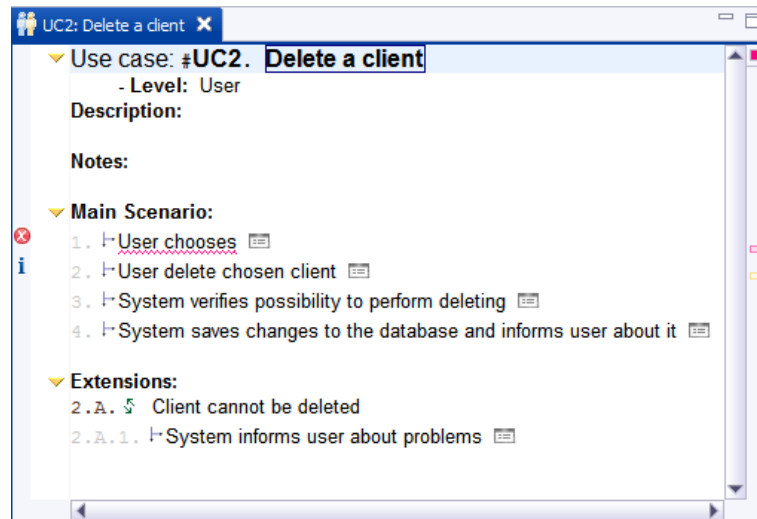


FIGURE 5.3: UC Workbench editor with annotated defects.

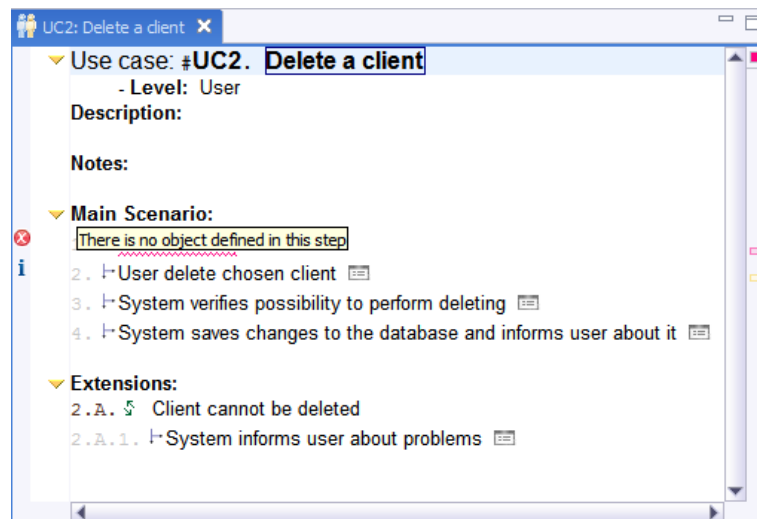


FIGURE 5.4: UC Workbench editor with annotated defects and additional information displayed.

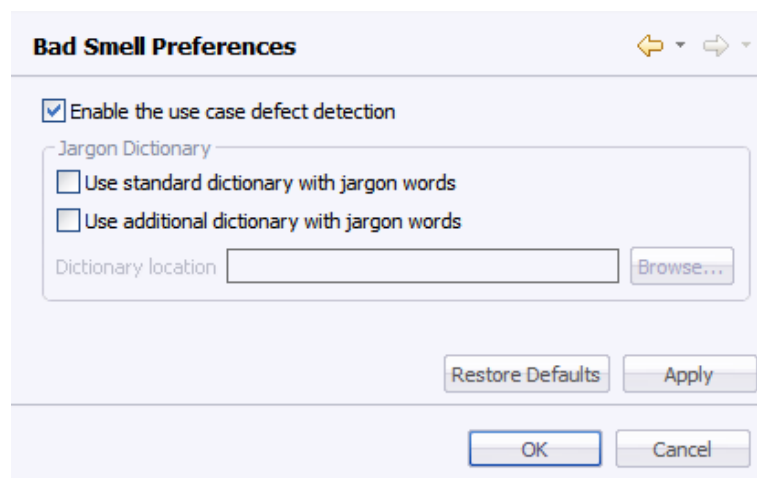


FIGURE 5.5: Preferences page for defects detection.

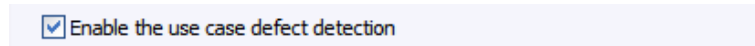


FIGURE 5.6: Enabling / Disabling defects detection mechanism.

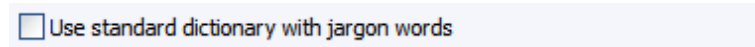


FIGURE 5.7: Enabling / Disabling usage of standard dictionary containing the jargon words.

- enable / disable usage of standard dictionary containing the jargon words (Figure 5.7)
- enable / disable usage of custom dictionary containing the jargon words and pointing the location of this dictionary (Figure 5.8)

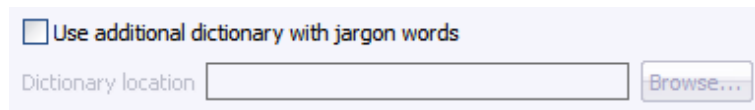


FIGURE 5.8: Enabling / Disabling usage of custom dictionary containing the jargon words.

### Action

Detection of use-case duplication is a complex process which might be time consuming (for instance when project with large number of use cases is considered). Therefore this process is not invoked automatically for every change in a use case, but a separate action (presented in Figure 5.9) was provided in the *UC Navigator* (which is a part of *UC Workbench*).

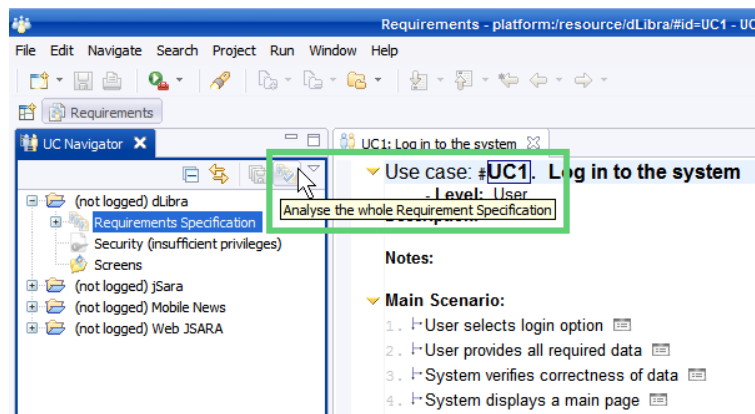


FIGURE 5.9: Action button for analysing whole requirements specification.

## 5.4 Testing

The implementation was tested on the unit level and on the integration level. Unit testing is a procedure used to validate that individual units of source code are working properly. For the purpose of unit testing *JUnit* tool was used together with *EasyMock* library. This approach allows to test every single class independently.

*Eclipse IDE* provides a mechanism for integration testing. It allows to run tests in an environment which does not differ from the final product configuration. This approach enables

developers to detect problems in communication between modules. These problem are not only related to modules being developed, but also to modules from *Eclipse* environment.

## Chapter 6

# Evaluation of the solution

In order to evaluate proposed methods of defects detection in use cases there was an experiment conducted. A set of use cases from three different Software Requirements Specifications (SRS), created as a part of Software Development Studio (SDS) projects, were chosen as the input data for the experiment. SDS is one of the main components of the Master of Software Engineering programme at Poznan University of Technology. The projects are developed for real customers from the university unit, industry or other organisation. Average SDS project engages eight students: two from fifth year, two from fourth year and four from the third year, who acts different roles. Most commonly one person from fourth year is an analyst, whose responsibility is to elicitate requirements from the customer.

### 6.1 Evaluation metrics

During the experiment 53 use cases, which altogether contain 420 steps (from the main scenarios and extensions), were analysed. Before the input data was processed by the described spike solution, its quality was assessed by the authors of this thesis. Every step, use case and whole SRS was analysed against each defect described in this thesis. The same analysis was performed by the spike solution. Tables A.1 - A.63 present the comparison of defects detection analysis performed by the spike solution and the experts. It should be noticed that not all the steps were analysed due to the constraint of sentence length. For such steps the *Too Long Sentence (TLS)* column is set to *Y*.

The spike solution can be compared with a classifier from the domain of decision support. Both approaches analyse objects and assign categories to them. In case of the developed spike solution the objects are represented by different elements from requirements specification (use cases, steps, names of use cases) and the categories are represented by different defects. Every object can be assign to one or more categories.

Similarity to the classifiers allows to evaluate the proposed solution using a confusion matrix and several common classifiers performance metrics that can be calculated from the confusion matrix.

Confusion matrix [Faw03] for binary classification (True/False) is presented in Figure 6.1. The symbols used in the Figure 6.1 are described below:

- $T$  (*true*) - system answer that is consistent with expert decision
- $F$  (*false*) - system answer that is inconsistent with expert decision
- $P$  (*positive*) - system positive answer (defect occurs)

	Expert answer		
		y	n
System answer	Y	TP	FP
	N	FN	TN

FIGURE 6.1: Confusion matrix.

- $N$  (*negative*) - system negative answer (defect does not occur)

On the basis of the confusion matrix following metrics [KS04] can be calculated:

- *Accuracy (AC)* - proportion of the total number of predictions that were correct. It is determined using the equation 6.1.

$$AC = \frac{TP + TN}{TP + FN + FP + TN} \quad (6.1)$$

- *True positive rate (TP rate)* - proportion of positive cases that were correctly identified, as calculated using the equation 6.2.

$$TP \text{ rate} = \frac{TP}{TP + FN} \quad (6.2)$$

- *True negative rate (TN rate)* is defined as the proportion of negatives cases that were classified correctly, as calculated using the equation: 6.3.

$$TN \text{ rate} = \frac{TN}{TN + FP} \quad (6.3)$$

- *Precision (PR)* - proportion of the predicted positive cases that were correct, as calculated using the equation 6.4.

$$PR = \frac{TP}{TP + FP} \quad (6.4)$$

## 6.2 Results of the experiment

For every distinguished defect the metrics described in the previous Section were calculated. In the subsequent subsections the results are presented in accordance with defects levels division.

### 6.2.1 Specification level

Tables 6.1 and 6.2 presents values of metrics for the *Duplicated Use Cases* defect. It can be observed that even in such a small set of use cases the duplication of use cases can occur. The presented method of detection and the developed spike solution allow to detect this defect in order to keep the requirements specification more precise and clear.

TABLE 6.1: Confusion matrix for *Duplicated use cases* defect.

	Expert answer		
		UCD	¬UCD
System answer	UCD	1	0
	¬UCD	0	2

TABLE 6.2: Duplicated use cases metrics.

Accuracy (AC)	TP rate	TN rate	Precision
100%	1	1	1

### 6.2.2 Use case level

As it can be seen from the results *Too long or too short use cases* (Tables: 6.5, 6.6), *Complicated extension* (Tables: 6.7, 6.8) and *Repeated actions in neighbouring steps* (Tables: 6.3, 6.4) defects were detected with the 100% accuracy. Only the accuracy for *Inappropriate naming* (Tables: 6.9, 6.10) defect is lower than 100% what results from the language ambiguities described in Section 2.1. Following use-case name can be presented as an example:

*View content*

In this case word *content* is treated by the *Stanford parser* as an *adjective* instead of a *noun*. Therefore no *obj* dependency is found, ipso facto, no *object* is recognised and the use-case name is marked as invalid.

TABLE 6.3: Confusion matrix for *Repeated actions in neighbouring steps* defect.

		Expert answer	
		RANS	-RANS
System answer	RANS	3	0
	-RANS	0	55

TABLE 6.4: *Repeated actions in neighbouring steps* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
100%	1	1	1

TABLE 6.5: Confusion matrix for *Too long or too short use cases* defect.

		Expert answer	
		LSUC	-LSUC
System answer	LSUC	7	0
	-LSUC	0	51

TABLE 6.6: *Too long or too short use cases* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
100%	1	1	1

TABLE 6.7: Confusion matrix for *Complicated extension* defect.

		Expert answer	
		CE	-CE
System answer	CE	0	0
	-CE	0	58

TABLE 6.8: *Complicated extension* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
100%	-	1	-

TABLE 6.9: Confusion matrix for *Inappropriate naming* defect.

		Expert answer	
		IN	-IN
System answer	IN	5	3
	-IN	2	48

TABLE 6.10: *Inappropriate naming* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
91%	0.71	0.94	0.63

### 6.2.3 Step level

From the results of evaluation of *Lack of the Actor* (Tables: 6.11, 6.12) and *Conditional steps* (Tables: 6.13, 6.14) defects one can observe that every analysed use-case step was correctly classified by the developed spike solution.

TABLE 6.11: Confusion matrix for *Lack of the Actor* defect.

		Expert answer	
		LOA	-LOA
System answer	LOA	27	0
	-LOA	0	370

TABLE 6.12: *Lack of the Actor* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
100%	1	1	1

TABLE 6.13: Confusion matrix for *Conditional steps* defect.

		Expert answer	
		CS	-CS
System answer	CS	1	0
	-CS	0	396

TABLE 6.14: *Conditional steps* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
100%	1	1	1

The analysis of the evaluation results for the rest of step level defects, which accuracy is under 100%, is presented below:

- *Using Technical Jargon* (Tables: 6.15, 6.16) - the *accuracy* for this defect is equal 97%. However, *TP rate* equals 1, what indicates that all the real defects were detected properly, but also some correct steps were marked as invalid. This inaccuracy occurs due to the problem

of homography (words share the same spelling, but have different meanings), for instance word *thread* may be interpreted as a part of a discussion (common meaning) or as a concurrent execution of a program (technical meaning). The detection method of this defect is based on the dictionary of jargon words, but it does not analyse the text from the semantic point of view, therefore it is not able to distinguish only real defects from all the potential candidates.

TABLE 6.15: Confusion matrix for *Using Technical Jargon* defect.

		Expert answer	
		UTJ	¬UTJ
System answer	UTJ	25	10
	¬UTJ	0	362

TABLE 6.16: *Using Technical Jargon* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
97%	1	0.97	0.71

- *Too Complex Sentence Structure* (Tables: 6.17, 6.18) - this defect, similarly to the previous one, has the *accuracy* at the level of 97% and *precision* at the level of 0.70, nonetheless, *TP rate* equals only 0.44. Lower value of *TP rate* is a result of analysing the steps only from the sentence structure perspective. On the other hand reviewers assess the complexity of the step on the basis of its readability and understandability. Although the method does not defect all the expected defects, it also does not confuse the analyst by marking simple steps as too complicated (*TN rate* = 0.99). With such situation on mind, the developed spike solution generates warning message instead of error, to indicate possible defect.

TABLE 6.17: Confusion matrix for *Too Complex Sentence Structure* defect.

		Expert answer	
		TCSS	¬TCSS
System answer	TCSS	7	3
	¬TCSS	9	378

TABLE 6.18: *Too Complex Sentence Structure* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
97%	0.44	0.99	0.70

- *Misusing Tenses and Verb Forms* - values of metrics for this defect (presented in Tables: 6.19, 6.20) point that most of the defects are detected (*TP rate* = 0.94), however, a number of correct steps are marked as invalid (*precision* = 0.55). Such behaviour can be explained by the specificity of English. In English some words can be both verbs and nouns (as described in Section 2.1). Additionally both the third person singular form of verb and plural form of noun is composed by adding *s* to the end of the base form. This leads to a situation when verb can be confused with noun, for instance in the following sentence:

*System displays the list of clients*

word *displays* is recognised as noun in the plural form and not as a verb in the third person singular. The developed spike solution considers such situation as lack of predicate, therefore no form of verb can be distinguished and the defect is detected incorrectly.

TABLE 6.19: Confusion matrix for *Misusing Tenses and Verb Forms* defect.

		Expert answer	
		MTVF	¬MTVF
System answer	MTVF	64	52
	¬MTVF	4	277

TABLE 6.20: *Misusing Tenses and Verb Forms* defect metrics.

Accuracy (AC)	TP rate	TN rate	Precision
86%	0.94	0.84	0.55

### 6.3 Performance analysis

During the experiment time statistics were gathered to assess the performance of the developed spike solution. Table 6.21 presents the simple time statistics.

TABLE 6.21: Time statistics (in milliseconds) of the developed spike solution.

Maximum	Minimum	Average
10406	16	373

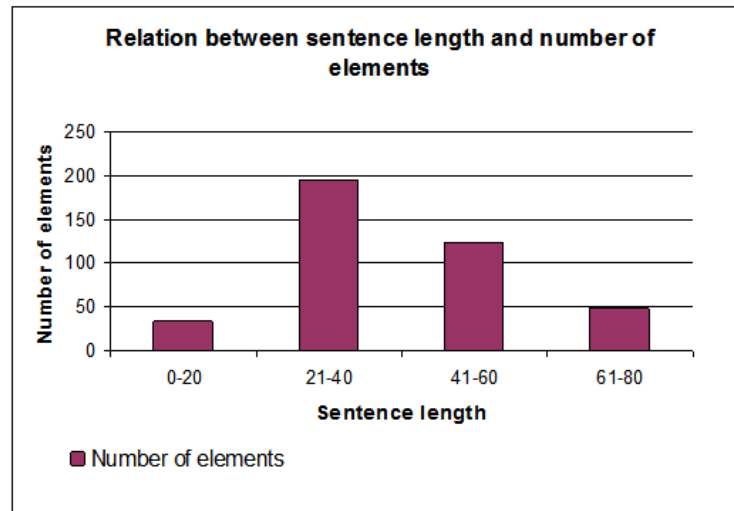


FIGURE 6.2: Relation between sentence length and number of elements.

In the evaluation it was assumed that sentence length is a number of characters the sentence is composed of. From the results it can be observed that the analysis can be performed without delays, which would be noticeable for the user. Only processing long sentences (containing from 61 to 80 characters) takes significant amount of time (as it can be observed at charts in Figures 6.2 and 6.3), however mechanisms described in Section 5.3.2 allows uninterrupted user work. Moreover, the number of sentences in the most time consuming range is relatively small. Most sentences belong to the range between 21 and 60 characters and their processing time is under 250 ms.

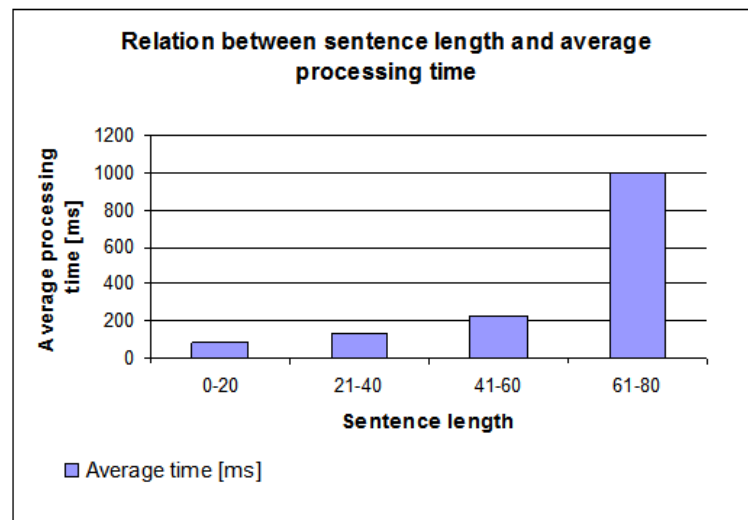


FIGURE 6.3: Relation between sentence length and average processing time.

## Chapter 7

# Conclusions

Requirements are crucial for the success of every software project. Therefore it is important to assure the high quality of the requirements. The main aim of this thesis was to develop methods for detecting requirements defects in an automatic way to support the reviews of the requirements. The aim has been achieved. In particular, a set of defects in use-cases was distinguished on the basis of patterns described by Cockburn [Coc01] and Adolph [ABCP02]. As use cases are written in a natural language, the range of natural language processing tools was investigated and the *Stanford parser* was chosen to develop mechanisms of automatic detection of the proposed set of defects.

To increase the practical meaning of the developed methods the spike solution was build on the basis of the proposed mechanisms. The spike solution was designed and implemented as a module to *UC Workbench/Eclipse Platform* tool to support the software analyst during the requirements elicitation phase.

To asses the quality of the created solution the experiment was prepared and performed on the set of 53 use cases with 420 steps altogether. The analysis of the evaluation allows to state that the developed methods of automatic defects detection are reliable and can significantly improve the quality of the requirements.

Some of the results presented in the thesis have been already published [CJNO07]. However additional experiments with the requirements taken from the industry could be helpful, as they could indicate some improvements to the proposed method.

Moreover, as a future research direction, extending the method to analyse requirements written in other languages (e.g. Polish) can be considered.

## Appendix A

### Analysed use cases and analysis results

Table A.1: Comparison of analysis of defects detection in use cases chosen SRSs performed by the spike solution and the experts (N - defect not found, Y - defect found).

SRS	Spike solution		Exsperts	
	UCD	Number of detected UCD	UCD	Number of detected UCD
dLibra	N	0	N	0
Mobile News	N	0	N	0
Web JSARA	Y	1	Y	1

#### A.1 dLibra CRM use cases

Table A.2: Comparison of analysis of defects detection in use cases from *dLibra CRM* SRS performed by the spike solution and the experts (N - defect not found, Y - defect found).

Use case	Spike solution				Exsperts			
	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN
UC1.1	N	N	N	N	N	N	N	N
UC1.2	N	N	N	N	N	N	N	N
UC1.3	N	N	N	N	N	N	N	N
UC1.5	N	N	N	N	N	N	N	N
UC1.6	N	N	N	Y	N	N	N	Y
UC1.7	N	N	N	N	N	N	N	N
UC1.8	N	N	N	N	N	N	N	N
UC1.9	N	N	N	N	N	N	N	N
UC1.10	N	N	N	N	N	N	N	N
UC1.11	N	N	N	N	N	N	N	N
UC1.12	N	N	N	N	N	N	N	N
UC1.13	N	N	N	N	N	N	N	N
UC2.2	N	N	N	N	N	N	N	N
UC2.3	N	N	N	N	N	N	N	N
Use case	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN

**A.1.1 UC1.1 Log in to the system****Main Scenario:**

1. User selects login option.
2. User provides all required data.
3. System verifies correctness of data.
4. System displays a main page.

**Extensions:**

- 3.a. Data is incomplete or incorrect: counterUCA
  - 3.a.1 System asks for data again.
  - 3.a.2 Back to step 2.

Table A.3: Defects detection results for steps of use case (*dLibra CRM*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
3.a.1	N	N	N	N	N	N	N	N	N	N	N
3.a.2	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.1.2 UC1.2 Add a new client****Main Scenario:**

1. User select option for adding new clients.
2. User fills all required personal client data forms.
3. System verifies correctness of data.
4. System adds a new client to the database and informs user about it.

**Extensions:**

- 3.a. Data is incomplete or incorrect:
  - 3.a.1 System informs user about problems.
  - 3.a.2 User makes a correction to the data.
  - 3.a.3 Back to step 3.
- 3.b. Client with same personal data already exists:

3.b.1 System informs user about that fact.

3.b.2 Back to step 1.

4.a. Client can't be added:

4.a.1 System informs user about reason why client can't be added.

Table A.4: Defects detection results for steps of use case UC1.2 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	Y	Y	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	Y	N
3.a.1	N	N	N	N	N	N	N	N	N	N	N
3.a.2	N	N	N	N	N	N	N	N	N	N	N
3.a.3	N	Y	N	N	N	Y	Y	N	N	N	Y
3.b.1	N	N	N	N	N	N	N	N	N	N	N
3.b.2	N	Y	N	N	N	Y	Y	N	N	N	Y
4.a.1	N	N	N	N	Y	Y	N	N	N	Y	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.3 UC1.3 Edit client data

#### Main Scenario:

1. User select option for editing clients.
2. User modifies personal client data.
3. System verifies correctness of data.
4. System saves a new client data to the database and informs user about it.

#### Extensions:

- 3.a. Data is incomplete or incorrect:
  - 3.a.1 System informs user about problems.
  - 3.a.2 User makes a correction to the data.
  - 3.a.3 Back to step 3.
- 3.b. Client with same personal data already exists:
  - 3.b.1 System informs user about that fact.
  - 3.b.2 Back to step 1.
- 4.a. Client data changes can't be saved:

4.a.1 System informs user about reason why client can't be modified.

Table A.5: Defects detection results for steps of use case UC1.3 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	Y	N
3.a.1	N	N	N	N	N	N	N	N	N	N	N
3.a.2	N	N	N	N	N	N	N	N	N	N	N
3.a.3	N	Y	N	N	N	Y	Y	N	N	N	Y
3.b.1	N	N	N	N	N	N	N	N	N	N	N
3.b.2	N	Y	N	N	N	Y	Y	N	N	N	Y
4.a.1	N	N	N	N	Y	Y	N	N	N	Y	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

#### A.1.4 UC1.4 Delete client

##### Main Scenario:

1. User select option for deleting clients.
2. User delete chosen client.
3. System verifies possibility to perform deleting.
4. System saves changes to the database and informs user about it.

##### Extensions:

- 3.a. Client can not be deleted:
  - 3.a.1 System informs user about the conditions.
  - 3.a.2 Use case ends.
  - 3.a.3 Back to step 3.

Table A.6: Defects detection results for steps of use case UC1.4 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	Y	N	N	N	N	Y
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	Y	N
3.a.1	N	N	N	N	N	N	N	N	N	N	N
3.a.2	N	N	N	N	N	Y	N	N	N	N	Y
3.a.3	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.5 UC1.5 Browse clients

**Main Scenario:**

1. User select option for browsing clients.
2. System displays the list of clients.
3. User may filter clients with specified criteria.
4. User may sort clients.
5. User may view details about selected client.
6. Use case ends when user logs out or selects different option.

**Extensions:**

- 2.a. There are no clients to display:
  - 2.a.1 System displays blank list.
  - 2.a.2 Use case ends.

Table A.7: Defects detection results for steps of use case UC1.5 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	Y	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	Y
4	N	N	N	N	N	Y	N	N	N	N	Y
5	N	N	N	N	N	Y	N	N	N	N	Y
6	N	Y	N	N	N	N	Y	N	N	Y	N
2.a.1	N	N	N	N	N	N	N	N	N	N	N
2.a.2	N	Y	N	N	N	Y	Y	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.6 UC1.6 Search

**Main Scenario:**

1. User select option for searching.
2. User selects subject of search (clients, contracts, installations).
3. System displays list of possible criteria.
4. User creates filter for searching.
5. System search the database and displays the results.

**Extensions:**

- 4.a. Chosen criteria are invalid:

5.a.1 System warns user.

5.a.2 Back to step 3.

5.a. No records found:

5.a.1 System displays blank list.

5.a.2 Use case ends.

Table A.8: Defects detection results for steps of use case UC1.6 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	Y	N	Y	N	N	Y	N	Y
4.a.1	N	N	N	N	N	N	N	N	N	N	N
4.a.2	N	Y	N	N	N	Y	Y	N	N	N	Y
5.a.1	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.7 UC1.7 Add a new contract

#### Main Scenario:

1. User select select a client for whom new contract will be added.
2. User chooses option for adding new contract.
3. System displays transection form.
4. User fills all required data.
5. System verifies information.
6. Sytem saves contract and bounds it to the selected client.

#### Extensions:

- 5.a. Data is incomplete or incorrect:
  - 5.a.1 System informs user about problems.
  - 5.a.2 User makes a correction to the data.
- 6.a. contract can't be saved:
  - 6.a.1 System informs user about that fact.

Table A.9: Defects detection results for steps of use case UC1.7 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	Y	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	Y	N	N	N	N	Y	N	N	N	N
5.a.1	N	N	N	N	N	N	N	N	N	N	N
5.a.2	N	N	N	N	N	N	N	N	N	N	N
6.a.1	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.8 UC1.8 Edit contract

#### Main Scenario:

1. User select select a client.
2. User chooses option for editing an existing contract.
3. System displays transection form.
4. User changes desired data.
5. System verifies information.
6. Sytem saves changed contract.

#### Extensions:

- 5.a. Data is incomplete or incorrect:
  - 5.a.1 System informs user about problems.
  - 5.a.2 User makes a correction to the data.
- 6.a. contract can't be saved:
  - 6.a.1 System informs user about that fact.

Table A.10: Defects detection results for steps of use case UC1.8 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	Y	N	N	N	N	Y	N	N	N	N
5.a.1	N	N	N	N	N	N	N	N	N	N	N
5.a.2	N	N	N	N	N	N	N	N	N	N	N
6.a.1	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.9 UC1.9 Delete contract

#### Main Scenario:

1. User select option for deleting contract.
2. User delete chosen contract.
3. System verifies possibility to perform deleting.
4. System saves changes to the database and informs user about it.

#### Extensions:

- 3.a. Contract can not be deleted:
  - 3.a.1 System informs user about problems.
  - 3.a.2 User check possibility to perform deleting.
  - 3.a.3 Back to step 3.

Table A.11: Defects detection results for steps of use case UC1.9 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	Y	N	N	N	N	Y
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	N	N
3.a.1	N	N	N	N	N	N	N	N	N	N	N
3.a.2	N	N	N	N	N	Y	N	N	N	N	Y
3.a.3	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.10 UC1.10 Add a new installation

#### Main Scenario:

1. User select select a client for whom new installation will be added.

2. User chooses option for adding new installation.
3. System displays installation form.
4. User fills required data.
5. System verifies information.
6. Sytem saves installation and bounds it to the selected client.

**Extensions:**

- 5.a. Data is incomplete or incorrect:
  - 5.a.1 System informs user about problems.
  - 5.a.2 User makes a correction to the data.
- 6.a. Installation can't be saved:
  - 6.a.1 System informs user about that fact.

Table A.12: Defects detection results for steps of use case UC1.10 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	Y	Y	N	Y	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	Y	N	N	N	N	Y	N	N	N	N
5.a.1	N	N	N	N	N	N	N	N	N	N	N
5.a.2	N	N	N	N	N	N	N	N	N	N	N
6.a.1	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.1.11 UC1.11 Edit installation****Main Scenario:**

1. User select select a client.
2. User chooses option for editing an existing installation.
3. System displays installation form.
4. User changes desired data.
5. System verifies information.
6. Sytem saves changed installation.

**Extensions:**

- 5.a. Data is incomplete or incorrect:
  - 5.a.1 System informs user about problems.
  - 5.a.2 User makes a correction to the data.
- 6.a. Installation can't be saved:
  - 6.a.1 System informs user about that fact.

Table A.13: Defects detection results for steps of use case UC1.11 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	Y	N	N	N	N	Y	N	N	N	N
5.a.1	N	N	N	N	N	N	N	N	N	N	N
5.a.2	N	N	N	N	N	N	N	N	N	N	N
6.a.1	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.1.12 UC1.12 Delete installation****Main Scenario:**

1. User select option for deleting installation.
2. User delete chosen installation.
3. System verifies possibility to perform deleting.
4. System saves changes to the database and informs user about it.

**Extensions:**

- 3.a. Installation can not be deleted:
  - 3.a.1 System informs user about problems.
  - 3.a.2 User check possibility to perform deleting.
  - 3.a.3 Back to step 3.

Table A.14: Defects detection results for steps of use case UC1.12 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	Y	N	N	N	N	Y
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	N	N
3.a.1	N	N	N	N	N	N	N	N	N	N	N
3.a.2	N	N	N	N	N	Y	N	N	N	N	Y
3.a.3	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.1.13 UC1.13 Prepare a report

#### Main Scenario:

1. User select option for creating reports.
2. System displays a list of possible fields in the report.
3. User selects fields to be included in the report and rules to filter values from database.
4. User order report generation.
5. System asks for type and localisation of the output file with report.
6. User selects the type and localisation of the output file with report.
7. System generates a report.

#### Extensions:

- 7.a. Report can't be saved in given location:
  - 7.a.1 System displays information.
  - 7.a.2 Back to step 5.

Table A.15: Defects detection results for steps of use case UC1.13 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	Y	N	N	N	N	N
3	Y	-	-	-	-	-	-	-	-	-	-
4	N	N	N	N	N	Y	N	N	N	N	Y
5	N	N	N	Y	N	N	N	N	Y	N	N
6	N	N	N	Y	N	N	N	N	Y	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
7	N	N	N	N	N	N	N	N	N	N	N
7.a.1	N	N	N	N	N	N	N	N	N	N	N
7.a.2	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

#### A.1.14 UC2.1 Log in to the system

See UC1.1 Log in to the system (??).

#### A.1.15 UC2.2 Browse information

##### Main Scenario:

1. User select option for browsing data.
2. System displays the list of licences, keys, contracts and installations (grouping by type).
3. User may filter data with specified criteria.
4. User may sort data.
5. User may view details about selected element.
6. Use case ends when users logs out or select different option.

##### Extensions:

- 2.a. There are no data to display:
  - 2.a.1 System displays blank list.
  - 2.a.2 Use case ends.

Table A.16: Defects detection results for steps of use case UC2.2 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	Y	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	Y
4	N	N	N	N	N	Y	N	N	N	N	Y
5	N	N	N	N	N	Y	N	N	N	N	Y
6	N	Y	N	N	N	N	Y	N	N	Y	N
2.a.1	N	N	N	N	N	N	N	N	N	N	N
2.a.2	N	Y	N	N	N	Y	Y	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

#### A.1.16 UC2.3 Request for licence

##### Main Scenario:

1. User select option for requesting a new licences.
2. System displays the list of user's contracts.
3. User selects one contract for licence request.
4. System contact with dLibra server to obtain all necessary data.
5. System validate given data.
6. System store a request new license and informs user about it.
7. PCSS Team Participant approve request for a new license.
8. User downloads the licence file.

**Extensions:**

- 2.a. There are no contracts:
  - 2.a.1 System displays blank list.
  - 2.a.2 Use case ends.
- 3.a. elected contract can not have more licenses:
  - 2.a.1 System informs user about that fact.
  - 2.a.2 Back to step 2.
- 5.a. Data is not valid:
  - 4.a.1 System informs user about incorrect data.
  - 4.a.2 Back to step 2.

Table A.17: Defects detection results for steps of use case UC2.3 (dLibra CRM) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	Y	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	Y	N	N	Y	N	Y
5	N	N	N	N	N	Y	N	N	N	N	Y
6	N	N	N	N	N	N	N	N	N	N	Y
7	N	Y	N	N	N	Y	Y	N	N	N	Y
8	N	N	N	Y	N	N	N	N	Y	N	N
2.a.1	N	N	N	N	N	N	N	N	N	N	N
2.a.2	N	Y	N	N	N	Y	Y	N	N	N	N
3.a.1	N	N	N	N	N	N	N	N	N	N	N
3.a.2	N	Y	N	N	N	Y	Y	N	N	N	Y
5.a.1	N	N	N	N	N	N	N	N	N	N	N
5.a.2	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

## A.2 Mobile News use cases

Table A.18: Comparison of analysis of defects detection in use cases from *Mobile News* SRS performed by the spike solution and the experts (N - defect not found, Y - defect found).

Use case	Spike solution				Exsperts			
	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN
UC1	N	N	N	N	N	N	N	N
UC2	N	N	N	N	N	N	N	N
UC3	N	N	N	N	N	N	N	N
UC4	N	Y	N	N	N	Y	N	N
UC5	N	Y	N	N	N	Y	N	N
UC6	N	N	N	N	N	N	N	N
UC7	N	N	N	N	N	N	N	N
UC8	N	N	N	N	N	N	N	N
UC9	N	Y	N	N	N	Y	N	N
UC10	N	N	N	N	N	N	N	N
UC11	N	N	N	N	N	N	N	N
UC12	N	N	N	N	N	N	N	N
UC13	N	N	N	N	N	N	N	N
UC14	N	N	N	N	N	N	N	N
UC15	N	N	N	N	N	N	N	N
Use case	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN

### A.2.1 UC1 Post a group message

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the 'Post group message' option.
3. Administrator types the message and posts it.
4. User receives the message when downloading new data.

#### Extensions:

Table A.19: Defects detection results for steps of use case UC1 (*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.2 UC2 Configure the server

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the ‘Configure’ option.
3. Administrator chooses and changes the desired settings.
4. Administrator saves configuration settings.

#### Extensions:

- 4.a. Administrator cancels configuration changes.

Table A.20: Defects detection results for steps of use case UC2  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.3 UC3 Add a new channel group

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. System displays administration options.
3. Administrator selects the ‘Group and channel management’ option.
4. System displays a list of defined channel groups and an add/delete group menu.
5. Administrator types the name of a new group and selects ‘Add group’.
6. System checks if a group with the given name has not been already defined and if so, inserts the name of a new group into a database.
7. See step 4.

#### Extensions:

Table A.21: Defects detection results for steps of use case UC3  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	Y	N	N	N	Y	N
5	N	N	N	N	N	Y	N	N	N	N	N
6	Y	-	-	-	-	-	-	-	-	-	-
7	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

#### A.2.4 UC4 Add a new channel

##### Main Scenario:

1. Administrator logs on to the administration panel.
2. System displays administration options.
3. Administrator selects the 'Group and channel management' option.
4. System displays a list of defined channel groups and an add/delete group menu.
5. Administrator chooses a group to which he wants to add a new channel.
6. System displays a list of channels in the selected group and an add/delete menu.
7. Administrator types the name of the channel and the URL of the news service and selects 'Add channel'.
8. System checks if a channel with the given name or URL has not been already defined and if so, inserts the channel information into a database.
9. System adds an information about the new channel to a group message.
10. See step 6.
11. Administrator selects the 'Finish' option.
12. System posts a group message containing information about all new channels in the selected channel group.

##### Extensions:

- 5.a. Administrator adds more channels. Proceed to step 7.

Table A.22: Defects detection results for steps of use case UC4  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	Y	N	N	Y	N	N
5	N	N	N	N	Y	N	N	N	N	N	N
6	N	N	N	Y	N	N	N	N	Y	N	N
7	Y	-	-	-	-	-	-	-	-	-	-
8	Y	-	-	-	-	-	-	-	-	-	-
9	N	N	N	N	N	N	N	N	N	N	N
10	N	Y	N	N	N	Y	Y	N	N	N	Y
11	N	N	N	N	N	N	N	N	N	N	N
12	Y	-	-	-	-	-	-	-	-	-	-
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.5 UC5 Delete a channel

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. System displays administration options.
3. Administrator selects the 'Group and channel management' option.
4. System displays a list of defined channel groups and an add/delete group menu.
5. Administrator chooses a group containing the channel he wants to delete.
6. System displays a list of channels in the selected group and an add/delete menu.
7. Administrator selects the channel(s) he wants to delete and chooses the 'Delete' option.
8. System deletes the selected channels from the database.
9. System posts a group message containing information about the deleted channels in the selected channel group to all users involved (subscribing the deleted channels).
10. System deletes all subscription information concerning the deleted channels.

#### Extensions:

Table A.23: Defects detection results for steps of use case UC5  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	Y	N	N	Y	N	N
5	Y	-	-	-	-	-	-	-	-	-	-
6	N	N	N	Y	N	N	N	N	Y	N	N
7	Y	-	-	-	-	-	-	-	-	-	-
8	N	N	N	Y	N	N	N	N	Y	N	N
9	Y	-	-	-	-	-	-	-	-	-	-
10	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.6 UC6 Delete a channel group

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. System displays administration options.
3. Administrator selects the 'Group and channel management' option.
4. System displays a list of defined channel groups and an add/delete group menu.
5. Administrator selects the group(s) he wants to delete and chooses the 'Delete' option.
6. System asks for confirmation.
7. System deletes all channels from the selected groups (see: UC5, steps 8 to 10).

#### Extensions:

Table A.24: Defects detection results for steps of use case UC6  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	Y	N	N	Y	N	N
5	Y	-	-	-	-	-	-	-	-	-	-
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.7 UC7 Delete a user group

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. System displays administration options.
3. Administrator selects the 'Delete users' option.
4. System displays the users' deletion menu.
5. Administrator selects deletion options (i.e. date of users' last login).
6. Administrator confirms deletion request.
7. System finds all users matching deletion criteria and deletes found user accounts.

#### Extensions:

- 6.a. Administrator cancels user deletion

Table A.25: Defects detection results for steps of use case UC7  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	Y	-	-	-	-	-	-	-	-	-	-
8	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.8 UC8 Update news

#### Main Scenario:

1. Daemon sends a HTTP request to a defined news service.
2. System receives a RSS-like formatted news file.
3. System parses the received news file.
4. System assigns an expiry date and time to each incoming message.
5. System inserts appropriate parts of the news file into a news database.

#### Extensions:

Table A.26: Defects detection results for steps of use case UC8  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	Y	N	N	N	N	Y	N	N
2	N	N	N	Y	N	N	N	N	Y	N	N
3	N	N	N	Y	N	N	N	N	Y	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	Y	N	Y	N	N	Y	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.9 UC9 Delete news

#### Main Scenario:

1. System queries the database for news messages, whose expiry date and time have passed.
2. System deletes all returned messages from the database.

#### Extensions:

Table A.27: Defects detection results for steps of use case UC9  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	Y	-	-	-	-	-	-	-	-	-	-
2	N	N	N	Y	Y	Y	N	N	Y	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.10 UC10 Register a new user

#### Main Scenario:

1. System asks the user if he/she wants to register.
2. User confirms he/she wants to register.
3. System sends a registration request to the server.
4. Server creates a new user account and sends back a user ID.
5. System stores the user ID and instructs the user how to subscribe news channels or configure his/her preferences.

#### Extensions:

- 4.a. User refuses to register.
  - 2.a.1 System displays an information that it cannot be used without prior registration.

2.a.2 User confirms the message.

2.a.3 System terminates.

Table A.28: Defects detection results for steps of use case UC10  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	Y	N	Y	N	N	Y	N	Y	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	Y	N	N	N	N	Y	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	Y	-	-	-	-	-	-	-	-	-	-
4.a.1	Y	-	-	-	-	-	-	-	-	-	-
4.a.2	N	N	N	N	N	N	N	N	N	N	N
4.a.3	N	N	N	N	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.11 UC11 Download news

#### Main Scenario:

1. User chooses to update locally stored news.
2. System sends a HTTP request to the Mobile News server.
3. Server sends all pending group messages.
4. Server sends separate news messages from all subscribed channels.
5. System receives news messages and stores them in a local database.
6. System displays a list of groups with subscribed channels and the number of new messages in each of them.

#### Extensions:

Table A.29: Defects detection results for steps of use case UC11  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	Y	N	N	N	N	Y	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	Y	N	N	N	N	Y	N	N
6	Y	-	-	-	-	-	-	-	-	-	-
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.12 UC12 Run the application

#### Main Scenario:

1. User starts the application.
2. System checks for registration information.
3. If the user is already registered, the system automatically updates news messages from subscribed channels (refer to ‘Download news’ use case). If no, the system attempts to register a new user (refer to ‘Register a new user’ use case).

#### Extensions:

Table A.30: Defects detection results for steps of use case UC12  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	Y	N	N	N	N	N	N	N
2	N	N	N	N	N	Y	N	N	N	N	N
3	Y	-	-	-	-	-	-	-	-	-	-
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.2.13 UC13 Subscribe/unsubscribe news channels

#### Main Scenario:

1. User chooses the ‘Channel subscription’ option.
2. System requests for and downloads a list of available groups and channels.
3. System displays a tree view of available groups and channels and marks those already subscribed by the user.
4. User selects the channels he/she wants to subscribe and/or deselects already subscribed channels to unsubscribe them and chooses the ‘Change subscription’ options.
5. System sends the subscription configuration to the Mobile News server and waits for confirmation.
6. Server alters the user’s subscription configuration in a database and sends a change confirmation.
7. System receives the confirmation and displays it.

#### Extensions:

Table A.31: Defects detection results for steps of use case UC13  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	Y	-	-	-	-	-	-	-	-	-	-
4	Y	-	-	-	-	-	-	-	-	-	-
5	Y	-	-	-	-	-	-	-	-	-	-
6	Y	-	-	-	-	-	-	-	-	-	-
7	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

#### A.2.14 UC14 Configure user preferences

##### Main Scenario:

1. User chooses the ‘Preferences’ option.
2. System displays a list of available options (i.e. font and color settings, local news caching etc.)
3. User configures the option according to his/her preferences and confirm the changes.
4. System saves user preferences configuration and displays main application view.

##### Extensions:

Table A.32: Defects detection results for steps of use case UC14  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	Y	-	-	-	-	-	-	-	-	-	-
3	Y	-	-	-	-	-	-	-	-	-	-
4	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

#### A.2.15 UC15 Read news

##### Main Scenario:

1. User chooses a news group from the ‘Today’ menu.
2. System displays a list of topics of available messages in chosen group.
3. User chooses a topic.
4. System displays the message using user’s appearance preferences.

5. User reads the message and closes it or uses a hyperlink to go to the full message.
6. System marks the message as 'Read'.

**Extensions:**

Table A.33: Defects detection results for steps of use case UC15  
(*Mobile News*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	Y	N	N	N	N	Y	N	N
2	N	N	N	N	N	Y	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	Y	-	-	-	-	-	-	-	-	-	-
6	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3 Web JSARA use cases**

Table A.34: Comparison of analysis of defects detection in use cases from *Web JSARA* SRS performed by the spike solution and the experts (N - defect not found, Y - defect found).

Use case	Spike solution				Experts			
	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN
UC1	N	N	N	N	N	N	N	N
UC2	N	N	N	N	N	N	N	N
UC3	N	N	N	N	N	N	N	N
UC4	Y	N	N	N	Y	N	N	N
UC5	Y	N	N	N	Y	N	N	N
UC6	N	N	N	N	N	N	N	N
UC7	N	N	N	Y	N	N	N	N
UC8	N	N	N	N	N	N	N	Y
UC9	N	N	N	N	N	N	N	N
UC10	N	N	N	N	N	N	N	N
UC11	N	N	N	N	N	N	N	N
UC12	N	N	N	N	N	N	N	N
UC13	N	N	N	N	N	N	N	N
UC14	N	Y	N	Y	N	Y	N	Y
UC15	N	Y	N	Y	N	Y	N	Y
UC16	N	N	N	N	N	N	N	N
UC17	N	N	N	N	N	N	N	Y
UC18	N	N	N	Y	N	N	N	N
Use case	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN

Use case	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN
UC19	N	N	N	N	N	N	N	N
UC20	N	Y	N	N	N	Y	N	N
UC21	N	N	N	N	N	N	N	N
UC22	N	N	N	Y	N	N	N	N
UC23	N	N	N	N	N	N	N	N
UC24	N	Y	N	N	N	Y	N	N
UC25	N	N	N	N	N	N	N	N
UC26	N	N	N	N	N	N	N	N
UC27	Y	N	N	N	Y	N	N	N
UC28	N	N	N	Y	N	N	N	Y
UC29	N	N	N	Y	N	N	N	Y
Use case	RANS	LSUC	CE	IN	RANS	LSUC	CE	IN

### A.3.1 UC1 Find data

#### Main Scenario:

1. User goes to search panel.
2. User gives the searching data name or specific keys which describe the data.
3. System searches the data in database.
4. System gives the path to searching data.

#### Extensions:

- 2.a. User cancels operation.
- 3.a. Return to main page.
- 4.a. Searching data has not been found.

Table A.35: Defects detection results for steps of use case UC1  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	Y	N
3	N	N	N	Y	N	N	N	N	Y	N	N
4	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.2 UC2 Show statistics

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the 'Statistics' option.

3. System displays form to insert statistics period.
4. Administrator writes down period which should be displayed.
5. System displays statistics.

**Extensions:**

- 3.a. Administrator cancels operation.
- 4.a. Proceed to step 1.

Table A.36: Defects detection results for steps of use case UC2  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.3 UC3 Create user account****Main Scenario:**

1. Administrator logs on to the administration panel.
2. System displays list of users and options.
3. Administrator selects the 'Create user account' option.
4. System displays form to insert user information.
5. Administrator writes down user info with login and password.
6. Administrator saves user settings.
7. System acknowledges operation.
8. System returns to administration panel.
9. System refresh list of users.

**Extensions:**

- 4.a. Administrator cancels operation.
- 5.a. Proceed to step 1.
- 6.b. System refuses operation and display message 'User already exists'.
- 7.b. Proceed to step 3.

Table A.37: Defects detection results for steps of use case UC3  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	Y	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	Y	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	N	N	N	N	N	N	N	N
9	N	N	N	N	N	Y	N	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.4 UC4 Modify user account

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. System displays list of users.
3. Administrator selects the ‘Change user account’ option.
4. Administrator selects user and click ‘Select’ button.
5. Administrator changes user info without login.
6. Administrator saves user settings.
7. System acknowledges operation.
8. System returns to administration panel.
9. System refresh list of users.

#### Extensions:

- 5.a. Administrator cancels operation.
- 6.a. Proceed to step 1.

Table A.38: Defects detection results for steps of use case UC4  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	N	N	N	N	N	N	N	N
9	N	N	N	N	N	Y	N	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.5 UC5 Delete user account

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. System displays list of users.
3. Administrator selects the 'Delete user account' option.
4. Administrator selects user and click 'Select' button.
5. System acknowledges operation.
6. System returns to administration panel.
7. System refresh list of users.

#### Extensions:

- 5.a. Administrator cancels operation.
- 6.a. Proceed to step 1.

Table A.39: Defects detection results for steps of use case UC5  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	Y	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	Y	N	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.6 UC6 Create data version

#### Main Scenario:

1. User select 'Save data' option.
2. System displays form to select file and insert information about.
3. User types file name with information about and confirm.
4. System check is that new version.
5. System calculates new version value.
6. System updates information in database.
7. System acknowledges operation.

**Extensions:**

- 3.a. User cancels operation.
- 4.a. System returns to previous page.
- 5.b. System inserts information into database with default version value.
- 6.b. Proceed to step 7.

Table A.40: Defects detection results for steps of use case UC6  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	Y	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	Y
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	Y	N	Y	N	N	Y	N	N
7	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.7 UC7 Edit data file****Main Scenario:**

1. User select 'Edit data' option.
2. System displays form to select data.
3. User types file name.
4. System displays data in editing form.
5. User change data and confirm.
6. System updates information in database with new version signature.
7. System acknowledges operation.

**Extensions:**

- 3.a. User cancels operation.
- 4.a. System returns to previous page.
- 5.b. User cancels operation.
- 6.b. System returns to previous page.

Table A.41: Defects detection results for steps of use case UC7  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	Y	N	N	N	N	N
5	N	N	N	N	N	Y	N	N	N	N	Y
6	N	N	N	Y	N	Y	N	N	Y	N	N
7	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.8 UC8 Database search****Main Scenario:**

1. User open start page.
2. System displays start page with search box form (Search box form is a part of start page).
3. User types regular expression in edit forms and confirm.
4. System search database with regular expression.
5. System displays result of search in browser as links to information in database.
6. User select link.
7. System opens link.

**Extensions:**

- 3.a. User closes browser.

Table A.42: Defects detection results for steps of use case UC8  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
2	N	N	N	Y	N	Y	N	N	Y	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	Y	N	Y	N	N	Y	N	Y
5	N	N	N	Y	N	Y	N	N	Y	Y	N
6	N	N	N	N	N	Y	N	N	N	N	Y
7	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.9 UC9 Upload data

#### Main Scenario:

1. User logs on to the user panel.
2. System displays user panel options.
3. User selects the ‘Data management’ option.
4. System displays the users’ data management menu.
5. User selects UploadData options.
6. System displays the users’ UploadData menu.
7. User selects the file and confirm.
8. System displays the users’ data management menu.

#### Extensions:

- 7.a. User cancels action.

Table A.43: Defects detection results for steps of use case UC9  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	Y	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	Y	N	N	N	N	Y	N	N
7	N	N	N	N	N	Y	N	N	N	N	Y
8	N	N	N	Y	N	Y	N	N	Y	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.10 UC10 Delete data

#### Main Scenario:

1. User logs on to the user panel.
2. System displays user panel options.
3. User selects the ‘Data management’ option.
4. System displays the users’ data management menu.
5. User selects DeleteData options.
6. System displays the users’ DeleteData menu.
7. User selects the file and confirm.
8. System displays the users’ data management menu.

**Extensions:**

- 7.a. User cancels action.

Table A.44: Defects detection results for steps of use case UC10  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	UTJ	Spike solution				Exsperts					
			TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS	
UC10	1	N	N	N	N	N	N	N	N	N	N	N
UC10	2	N	N	N	N	N	N	N	N	N	N	N
UC10	3	N	N	N	N	N	N	N	N	N	N	N
UC10	4	N	N	N	Y	N	N	N	N	Y	N	N
UC10	5	N	N	N	N	N	N	N	N	N	N	N
UC10	6	N	N	N	Y	N	N	N	N	Y	N	N
UC10	7	N	N	N	N	N	N	N	N	N	N	Y
UC10	8	N	N	N	Y	N	N	N	N	Y	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS	

**A.3.11 UC11 View data****Main Scenario:**

1. User logs on to the user panel.
2. System displays user panel options.
3. User selects the ‘Data management’ option.
4. System displays the users’ data management menu.
5. User selects ViewData options.
6. System displays the users’ files.

**Extensions:**

- 6.a. Proceed to step 1.

Table A.45: Defects detection results for steps of use case UC11  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	UTJ	Spike solution				Exsperts					
			TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS	
UC11	1	N	N	N	N	N	N	N	N	N	N	N
UC11	2	N	N	N	N	N	N	N	N	N	N	N
UC11	3	N	N	N	N	N	N	N	N	N	N	N
UC11	4	N	N	N	Y	N	N	N	N	Y	N	N
UC11	5	N	N	N	N	N	N	N	N	N	N	N
UC11	6	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS	

### A.3.12 UC12 Download file

#### Main Scenario:

1. User selects the 'Download' option.
2. System displays the Download menu.
3. User chooses the file.
4. System displays the questionnaire.
5. User fills the questionnaire and confirm.

#### Extensions:

- 5.a. User cancels form fill.

Table A.46: Defects detection results for steps of use case UC12  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	Y	N	N	N	N	N	N	N
2	N	N	N	Y	N	N	N	N	Y	N	N
3	N	N	N	Y	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	Y	N	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.13 UC13 RegisterUser

#### Main Scenario:

1. User selects the 'Register User' option.
2. System displays the questionnaire.

3. User fills questionnaire and confirm registration.
4. System sends questionnaire to administrator.
5. Administrator creates user account.

**Extensions:**

- 3.a. Incorrect data.
- 4.a. User cancels action.
- 5.b. Administrator rejects user request.
- 6.b. Proceed to step 1.

Table A.47: Defects detection results for steps of use case UC13  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	Y
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.14 UC14 Log on****Main Scenario:**

1. User writes down login and password into edit forms and confirm.
2. System displays default page.

**Extensions:**

- 2.a. System displays failure info page.

Table A.48: Defects detection results for steps of use case UC14  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.15 UC15 Log off****Main Scenario:**

1. User selects 'Logoff' option.
2. System closes window.

**Extensions:**

Table A.49: Defects detection results for steps of use case UC15  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	Y	N	N	N	N	Y	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.16 UC16 Run simulation****Main Scenario:**

1. User goes to start simulation panel.
2. System shows form with simulation parameters to write down.
3. User gives starting parameters for simulation, model and algorithm localization for execute.
4. User confirms operation.
5. System starts JSARA simulation.
6. After simulation system shows results.

**Extensions:**

- 3.a. User cancels operation.
- 4.a. Return to main page.
- 5.b. Simulation is started by another user.
- 6.b. System queue the simulation and shows information about it.

Table A.50: Defects detection results for steps of use case UC16  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	Y	N	Y	N	N	N	Y	N
6	N	Y	N	N	N	N	Y	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.17 UC17 Simulation time****Main Scenario:**

1. User selects 'Timeout' option.
2. System displays form with timeout.
3. User fills in the form new timeout and confirm.
4. System updates information about timeout.

**Extensions:**

- 2.a. User cancels operation.
- 3.a. Proceed to step 1.

Table A.51: Defects detection results for steps of use case UC17  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	Y	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	Y
4	N	N	N	N	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.18 UC18 Run applet****Main Scenario:**

1. User goes to start applet panel.
2. System shows form with applet parameters to write down.
3. User gives starting parameters for execute.
4. User confirms operation.
5. System starts applet.

**Extensions:**

- 3.a. User cancels operation.
- 4.a. Return to main page.

Table A.52: Defects detection results for steps of use case UC18  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	Y	N	N	N	N	Y	N	N
2	N	N	N	Y	N	N	N	N	Y	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	Y	N	N	N	N	Y	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.19 UC19 Create publication

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the 'Content management' option.
3. System displays content management form.
4. Administrator selects the 'Add information' option.
5. System displays information editing form.
6. Administrator composes information in editing form and associates it to specific level and confirm.
7. System acknowledges operation.
8. System returns to administration panel.

#### Extensions:

- 4.a. Administrator cancels operation.
- 5.a. Proceed to step 1.
- 6.b. Administrator cancels operation.
- 7.b. Proceed to step 3.

Table A.53: Defects detection results for steps of use case UC19  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	N	N	Y	N	N	N	Y	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.20 UC20 Modify publication

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the ‘Content management’ option.
3. System displays content management form.
4. Administrator selects the ‘Modify information’ option.
5. System displays information list in tree hierarchy form
6. Administrator chooses information and click ‘Select’ button.
7. System displays information in editing form.
8. Administrator changes information in editing form and confirm.
9. System acknowledges operation.
10. System returns to administration panel.

#### Extensions:

- 4.a. Administrator cancels operation.
- 5.a. Proceed to step 1.
- 6.b. Administrator cancels operation.
- 7.b. Proceed to step 3.
- 8.c. Administrator cancels operation.
- 9.c. Proceed to step 5.

Table A.54: Defects detection results for steps of use case UC20  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	Y	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
8	N	N	N	N	N	Y	N	N	N	N	Y
9	N	N	N	N	N	N	N	N	N	N	N
10	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.21 UC21 Delete publication

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the ‘Content management’ option.
3. System displays content management form.
4. Administrator selects the ‘Delete information’ option.
5. System displays information list in tree hierarchy form
6. Administrator selects information and click ‘Select’ button.
7. System acknowledges operation.
8. System returns to administration panel.

#### Extensions:

- 4.a. Administrator cancels operation.
- 5.a. Proceed to step 1.
- 6.b. Administrator cancels operation.
- 7.b. Proceed to step 3.

Table A.55: Defects detection results for steps of use case UC21  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	UTJ	Spike solution				Experts					
			TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS	
UC21	1	N	N	N	N	N	N	N	N	N	N	N
UC21	2	N	N	N	N	N	N	N	N	N	N	N
UC21	3	N	N	N	N	N	Y	N	N	N	N	N
UC21	4	N	N	N	N	N	N	N	N	N	N	N
UC21	5	N	N	N	N	N	Y	N	N	N	N	N
UC21	6	N	N	N	Y	N	N	N	N	Y	N	N
UC21	7	N	N	N	N	N	N	N	N	N	N	N
UC21	8	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS	

### A.3.22 UC22 View content

#### Main Scenario:

1. User open start page.
2. System displays menu of content and default messages.
3. User chooses branch from menu.
4. System displays menu of content and information from branch.
5. See Step 3.

**Extensions:**

- 3.a. User closes browser.

Table A.56: Defects detection results for steps of use case UC22  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	Y	N	N	N	N	Y
2	N	N	N	Y	N	Y	N	N	Y	N	N
3	N	N	N	Y	N	N	N	N	Y	N	N
4	N	N	N	Y	N	N	N	N	Y	N	N
5	N	Y	N	N	N	Y	Y	N	N	N	Y
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

**A.3.23 UC23 Create level****Main Scenario:**

1. Administrator logs on to the administration panel.
2. Administrator selects the 'Content management' option.
3. System displays content management form.
4. Administrator selects the 'Create level' option.
5. System displays level tree.
6. Administrator chooses place to create level, puts level name and confirm.
7. System acknowledges operation.
8. System returns to administration panel.

**Extensions:**

- 4.a. Administrator cancels operation.
- 5.a. Proceed to step 1.
- 6.b. Administrator cancels operation.
- 7.b. Proceed to step 3.

Table A.57: Defects detection results for steps of use case UC23  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	N	Y	N	N	N	N	Y	N
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.24 UC24 Modify level

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the 'Content management' option.
3. System displays content management form.
4. Administrator selects the 'Modify level' option.
5. System displays level tree.
6. Administrator chooses level and click 'Select' button.
7. System displays level name in editing form.
8. Administrator changes level form in editing form and confirm.
9. System acknowledges operation.
10. System returns to administration panel.

#### Extensions:

- 4.a. Administrator cancels operation.
- 5.a. Proceed to step 1.
- 6.b. Administrator cancels operation.
- 7.b. Proceed to step 3.
- 8.c. Administrator cancels operation.
- 9.c. Proceed to step 5.

Table A.58: Defects detection results for steps of use case UC24  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	Y	N	N	N	N	Y	N	N
7	N	N	N	N	N	Y	N	N	N	N	N
8	N	N	N	N	N	Y	N	N	N	N	Y
9	N	N	N	N	N	N	N	N	N	N	N
10	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.25 UC25 Delete level

#### Main Scenario:

1. Administrator logs on to the administration panel.
2. Administrator selects the 'Content management' option.
3. System displays content management form.
4. Administrator selects the 'Delete level' option.
5. System displays level tree.
6. Administrator selects level and click 'Select' button.
7. System acknowledges operation.
8. System returns to administration panel.

#### Extensions:

- 4.a. Administrator cancels operation.
- 5.a. Proceed to step 1.
- 6.b. Administrator cancels operation.
- 7.b. Proceed to step 3.

Table A.59: Defects detection results for steps of use case UC25  
(*Web JSARA*) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Exsperts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	Y	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	Y	N	N	N	N	Y	N	Y
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	N	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.26 UC26 Add thread

#### Main Scenario:

1. User logs on the system.
  2. User selects the 'Forum' panel.
  3. System displays forum panel.
  4. User selects 'Add Thread' option.
  5. System displays the thread form.
  6. User fills the thread form.
  7. User confirms send the thread.
  8. System displays the tree of thread.
- 6.a. User cancels the action.

Table A.60: Defects detection results for steps of use case UC26  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	N	N	N
5	N	N	N	Y	N	N	N	N	N	N	N
6	N	N	N	Y	N	N	N	N	N	N	N
7	N	N	N	Y	N	N	N	N	N	N	N
8	N	N	N	Y	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.27 UC27 AddPost

#### Main Scenario:

1. User logs on the system.
  2. User selects the 'Forum' panel.
  3. System displays forum panel.
  4. User selects the thread on the tree.
  5. User selects 'Add Post' option.
  6. System displays the post form.
  7. User fills the post form.
  8. User confirms send the post.
  9. System displays the tree of thread.
- 7.a. User cancels the action.

Table A.61: Defects detection results for steps of use case UC27  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	Y	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	N	N	N	N	N	N	N	N
9	N	N	N	N	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.28 UC28 Moderate

#### Main Scenario:

1. Administrator logs on the system.
2. Administrator selects the 'Forum' panel.
3. System displays forum panel.
4. Administrator selects the 'Moderate' options.
5. Administrator changes the forum.
6. Administrator confirms the changes.
7. System displays forum panel.

8. System displays the tree of thread.

6.a. Administrator cancels the action.

Table A.62: Defects detection results for steps of use case UC28  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N	N
4	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	Y	N	Y	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

### A.3.29 UC29 Quote

#### Main Scenario:

1. User logs on to the user's forum panel.
  2. System displays user's forum panel options.
  3. User selects the chosen thread.
  4. User goes to post which wants to quote.
  5. User selects 'Quote' option.
  6. System displays the quote panel.
  7. User inserts comment to quoted post.
  8. User selects 'Confirm' option.
  9. System adds new post with quote to the thread.
- 6.a. User cancels operation.
  - 7.a. System backs to the last choose thread.
  - 8.b. User cancels operation.
  - 9.b. System backs to the last choose thread.

Table A.63: Defects detection results for steps of use case UC29  
(Web JSARA) (N - defect not found, Y - defect found).

Step	TLS	Spike solution					Experts				
		UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS
1	N	N	N	N	N	N	N	N	N	N	N
2	N	N	N	N	N	N	N	N	N	N	N
3	N	N	N	Y	N	N	N	N	N	N	N
4	N	N	N	N	Y	N	N	N	N	Y	N
5	N	N	N	N	N	N	N	N	N	N	N
6	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N	N
8	N	N	N	N	N	N	N	N	N	N	N
9	N	N	N	Y	N	N	N	N	N	N	N
Step	TLS	UTJ	TCSS	LOA	MTVF	CS	UTJ	TCSS	LOA	MTVF	CS

## Appendix B

# Attachments

### B.1 CD

The CD contains:

- source code of the developed spike solution
- libraries necessary for running the developed spike solution
- master thesis in the pdf format
- master thesis in the latex format

## Appendix C

### Division of the performed tasks

Tasks		Alicja Cierniewska	Jakub Jurkiewicz
Reading literature concerning Natural Language Processing		✓	✓
Reading literature concerning requirements elicitation in a form of use cases		✓	✓
Choosing NLP tool ( <i>Stanford parser</i> )		✓	✓
Developing, implementing and testing methods of defects detection	Too long or too short use cases.	✓	
	Inconsistent style of naming	✓	
	Too Complex Sentence Structure	✓	
	Misusing Tenses and Verb Forms	✓	
	Using Technical Jargon	✓	
	Conditional Steps	✓	
	Duplicated use cases		✓
	Complicated Extensions		✓
	Repeated actions in neighboring steps.		✓
	Lack of the Actor		✓
Misusing Tenses and Verb Forms		✓	
Evaluation of the spike solution for the Software Requirements Specification for <i>Web jSara</i> project		✓	
Evaluation of the spike solution for the Software Requirements Specification for <i>dLibra CRM</i> and <i>Mobile News</i> projects			✓
Implementation and testing following parsers	PartOfSentenceParser	✓	
	PartOfSpeechParser	✓	
	JargonParser		✓
	NamingParser	✓	
	StructureParser	✓	
	ActorParser		✓
	BusinessObjectParser		✓
Implementation of GUI elements			✓

# Bibliography

- [ABCP02] S. Adolph, P. Bramble, A. Cockburn, and A. Pols. *Patterns for Effective Use Cases*. Addison-Wesley, 2002.
- [CB00] John Carroll and Ted Briscoe. Grammatical relation annotation. [on-line] <http://www.informatics.sussex.ac.uk/research/groups/nlp/carroll/grdescription/grdescription.html>, 2000.
- [Cho57] N. Chomsky. *Syntactic Structures*. The Hague-Paris: Mouton, 1957.
- [CJNO07] A. Ciemniowska, J. Jurkiewicz, J. Nawrocki, and Ł. Olek. Supporting use-case reviews. In *10th International Conference on Business Information Systems*, LNCS. Springer Verlag, April 2007.
- [CMB99] J. Carroll, G. Minnen, and T. Briscoe. Corpus annotation for parser evaluation. In *Proceedings of the EACL workshop on Linguistically Interpreted Corpora (LINC)*, 1999.
- [Coc01] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.
- [dMMM06] Marie-Catherine de Marneffe, B. MacCartney, and Ch. D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of the EACL workshop on Linguistically Interpreted Corpora (LINC)*, 2006.
- [eas] EasyMock project homepage. <http://www.easymock.org/>.
- [EMFa] EMF project homepage. <http://www.eclipse.org/modeling/emf/>.
- [Emfb] Emfatic project homepage. <http://www.alphaworks.ibm.com/tech/emfatic>.
- [EMFc] EMFTrans project homepage. <http://www.eclipse.org/modeling/emf/?project=transaction#transaction>.
- [Fag86] M. E. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, SE-12(7):744–751, 1986.
- [Faw03] T. Fawcett. Roc graphs: Notes and practical considerations for data mining researchers. 2003.
- [FGL03] A. Fantechi, S. Gnesi, and G. Lami. A relation-based approach to use case analysis. 2003.
- [F.P95] Jr. F.P.Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1995.
- [GHJV94] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [Gro04] Standish Group. *Chaos: A recipe for success*, 2004.
- [HP99] Hewlett-Packard. Applications of Software Measurement Conference, 1999.
- [Jac04] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

- [JFa] JFace project information. <http://en.wikipedia.org/wiki/jface>.
- [JM93] J.C.Knight and E. Ann Myers. An improved inspection technique. *Commun. ACM*, 36(11):51–61, 1993.
- [JUn] JUnit project homepage. <http://junit.org>.
- [KM02] D. Klein and Ch. D. Manning. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, 2002.
- [KS04] K. Krawiec and J. Stefanowski. *Uczenie maszynowe i sieci neuronowe*. Wydawnictwo Politechniki Poznańskiej, 2004.
- [LMI03] M. Luisa, F. Mariangela, and N. I.Pierluigi. Market research for requirements analysis using linguistic tools. 2003.
- [Mic06] Bartosz Michalik. Extending UCWorkbench with an asynchronous reviews module. Master's thesis, Poznan University of Technology, Poznań, Poland, 2006.
- [MP93] B. Macias and S. Pulman. *Natural language processing for requirements specification*. Chapman and Hall, 1993.
- [MS99] Ch. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.
- [NL03] C. J. Neill and P. A. Laplante. Requirements engineering: The state of the practice. *IEEE Software*, 20(6):40–45, 2003.
- [NO05] J. R. Nawrocki and Ł. Olek. Uc workbench: A tool for writing use-cases. In *6th International Conference on Extreme Programming and Agile Processes in Software Engineering*, volume 3556 of *LNCS*, pages 230–234. Springer, Jun 2005.
- [NS03] M. Niazi and S. Shastry. Role of requirements engineering in software development process: An empirical study. In *Multi Topic Conference, 2003. INMIC 2003. 7th International*, pages 402–407, 2003.
- [Och06] Mirosław Ochodek. Use cases based software effort estimation. Master's thesis, Poznan University of Technology, 2006.
- [OM96] M. Osborne and C.K. MacNish. Processing natural language software requirement specifications. In *Requirements Engineering Conference, 1996. Proceedings. 2nd IEEE International*, pages 229–236. IEEE, Apr 1996.
- [Pen] The Penn Treebank project homepage. <http://www.cis.upenn.edu/treebank/>.
- [sta] Stanford Parser project homepage. <http://www-nlp.stanford.edu/downloads/lex-parser.shtml>.
- [SWT] SWT project homepage. <http://www.eclipse.org/swt/>.
- [ucw] UC Workbench project homepage. <http://ucworkbench.cs.put.poznan.pl>.
- [Wei71] G. M. Weinberg. *The Psychology of Computer Programming*. Von Nonstrand Reinhold, 1971.
- [WNF64] H. Kucera W. N. Francis. Brown corpus manual. [on-line] <http://icame.uib.no/brown/bcm.html>, 1964.



© 2007 Alicja Cierniewska, Jakub Jurkiewicz

Poznan University of Technology  
Faculty of Computer Science and Management  
Institute of Computer Science

Typeset using L<sup>A</sup>T<sub>E</sub>X in Computer Modern.

Bib<sub>T</sub>E<sub>X</sub>:

```
@mastersthesis{ key,  
  author = "Alicja Cierniewska, Jakub Jurkiewicz",  
  title = "{Automatic detection of defects in use cases}",  
  school = "Poznan University of Technology",  
  address = "Pozna{\'}n, Poland",  
  year = "2007",  
}
```