Mining Architectural Responsibilities and Components from Textual Specifications Written in Natural Language

Agustin Casamayor, Daniela Godoy, Marcelo Campo

ISISTAN Research Institute, UNICEN University
Campus Universitario, Paraje Arroyo Seco
B7001BBO, Tandil, Bs. As., Argentina
CONICET, National Council for Scientific and Technical Research
C1033AAJ, Bs. As., Argentina
Te: +54 (2293) 439682 Ext. 42 - Fax: +54 (2293) 439681
{acasamay,dgodoy,mcampo}@exa.unicen.edu.ar

Abstract. Given the enormous growth and complexity of modern software systems, architectural design has become an essential concern for almost every software development project. One of the most challenging steps for designing the best architecture for a certain piece of software is the analysis of requirements, usually written in natural language by engineers not familiar with specific design formalisms. The Use Case Map (UCM) notation can be used to map requirements into proper design concerns, usually known as responsibilities. In this paper, we introduce an approach for mining candidate architectural responsibilities and components from textual descriptions of requirements using natural language processing (NLP) techniques, in order to relieve software designers of this complex and time-consuming task. High accuracy and precision rates achieved by applying part-of-speech (POS) tagging with domain rules and semantic clustering to textual requirement documents, suggest a great potential for providing assistance to software designers during early stages of development.

Keywords: software design, architectural responsibilities, architectural components, requirements engineering, text mining techniques, part-of-speech tagging

1 Introduction

During the last few years, the growing number of processes involved in Software Engineering activities has led to the introduction and popularization of several standards to measure and certify quality, not only of the system being developed, but also of the development process itself [11]. Multiple approaches have been proposed for aiding analysts and engineers in the definition and application of an active development process [19], including automated or semi-automated tools for modeling business processes and the application of re-engineering techniques. Modern Software Engineering is characterized by the use of several models that establish and show the different states a software product goes through, from its

initial conception to its end, covering its development, setup and maintenance amongst others. Requirements Engineering plays a critic role in this process since it is concerned with one of the most important stages of software development: the definition of the product we want to build, that is to say, the generation of correct and compact specifications that clearly and unambiguously describe the system's behavior. The failure of a high percentage of software projects is often caused by the lack of proper requirement analysis [24], followed by low user involvement and participation, incomplete specifications, and changing requirements [8], among others.

Software architectures are system models with such a high level of abstraction that allow different stakeholders to correctly handle the great distance between requirements and implementation. The growing adoption of architecture-centered development is mainly because the most important design decisions and their consequences are properly documented in an early stage of the software development process. This allows a better understanding of the system as a whole, taking into consideration every relevant quality attribute.

One of the most challenging steps for designing the best architecture for a certain software system is the analysis of requirements, usually written in natural language by engineers not familiar with specific design formalisms. Automatic processing and knowledge extraction from requirement documents can be performed with text mining. Text mining involves a set of techniques to organize, classify and extract relevant information from text collections. These practices are part of a much general process of Knowledge Discovery in Databases (KDD), which is the semi-automated process of extracting relevant knowledge from databases (that may be textual), aiming at discovering valid knowledge, previously unknown and potentially useful [5].

In this paper we propose the application of text mining techniques to requirement and use-case documents, in order to help designers in bridging the complex gap between requirement analysis and architectural software design through detection of responsibilities and components. The reminder of the article is organized as follows. Section 2 introduces the proposed approach to automatically detect candidate responsibilities and components from textual requirements using part-of-speech tagging and a set of custom rules for NL analysis. The empirical evaluation of this approach is explained and summarized in Section 3. Section 4 discusses related works in the use of Information Retrieval (IR) and linguistic techniques for analyzing textual requirements and aiding software designers in the complex task of bridging the gap between specifications and architectures. Finally, conclusions and ongoing work are stated in Section 5.

2 Proposed Approach

Software architectures are engineering artifacts that provide designers and developers with high-level descriptions of complex systems. Architectures are composed of several views that allow a better understanding of the system and can be represented using different types of diagrams. Use Case Map (UCM) is a visual standard notation for the materialization of architectural scenarios [18], which focuses on both static and dynamic aspects of a system. These diagrams

are useful for modeling architectures from early design stages, as they can be easily exploded into similar diagrams with further degrees of detail.

As can be seen in Figure 1, UCMs consist of a set of components and responsibilities associated to them, joined by the execution path of the quality scenario they materialize. The basic notation is very simple and consists of: a set of start-points, filled circles, which represent preconditions; responsibilities, crosses that represent the tasks to be executed; end-points, bars that represent post-conditions; and components, boxes that represent a software entity and contain responsibilities. The execution path is represented by curved lines, from the start point to the end, passing through all the responsibilities associated with the scenario. One of the advantages of using the UCM notation is that it is easily understandable by any stakeholder of the system, allowing designers to reduce the gap between clients' needs and requirement analysts. Some studies have particularly focused on the integration of UCM with UML and XML document formalization [3].

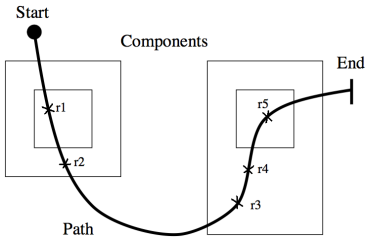


Fig. 1: UCM basic elements

Considering the expressiveness of this notation and the ease to create and understand these diagrams, we propose a method based on the application of text mining techniques to requirement documents written in natural language, for semi-automatic detection of architectural responsibilities and components. Using this method, responsibilities can be assigned to software components, which can aid designers to establish candidate architectures for a system.

Figure 2 depicts the proposed approach. Initially, a set of requirement documents and use case specifications provided by the requirements engineering team are processed with a classifier [4] that splits them into two groups, functional and non-functional (NFR). Afterwards, each document is analyzed with a part-of-speech (POS) tagger and a set of rules to determine whether if it is a candidate responsibility or not, weighted with information obtained in the previous stage. Then, responsibilities are grouped using semantic clustering to infer the components they may belong to. Finally, an analyst supervises these results and provides feedback to improve the tagging and detection process in further stages. In the following subsections we explain each of the above mentioned parts.

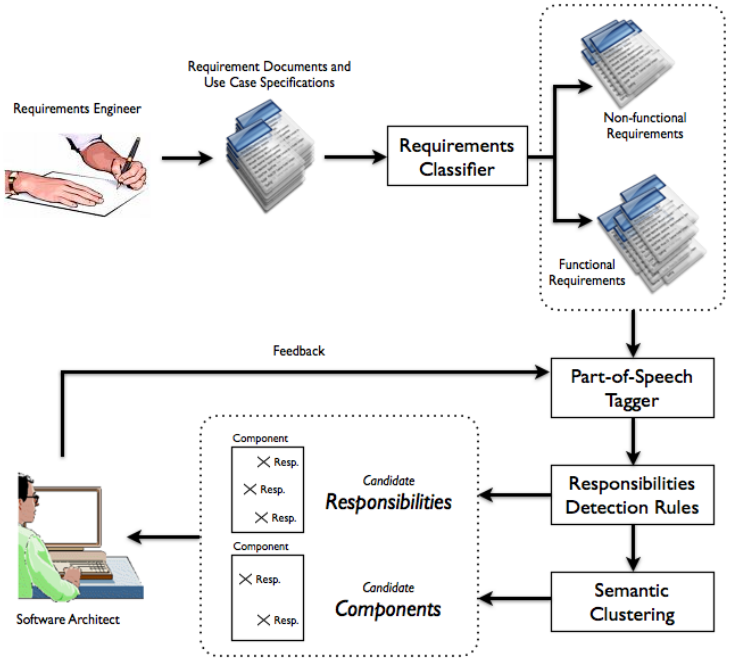


Fig. 2: Responsibilities and components detection approach

2.1 Part-of-Speech Tagging

Part-of-Speech (POS) tagging is the process of assigning a lexical class marker to every word in a sentence. The usual markers are noun, verb, pronoun, preposition, adverb, adjective, among others [17]. A POS tagging algorithm receives a sentence as input and returns the corresponding tags for every word within a specified tagset, which is a finite list of part-of-speech tags. When given out of context, many words have several senses or meanings, causing an ambiguity about how they should be interpreted. The task of disambiguation is to determine which of the senses of an ambiguous word is invoked in a particular use of the word, which is often performed by looking at the context of the word's use.

POS taggers can be rule-based or stochastic. Rule-based taggers use a set of predefined hand-written rules to distinguish the ambiguity of a tag. Stochastic taggers are either based on hidden Markov models (HMM), choosing the tag sequence which maximizes the product of word likelihood and tag sequence probability, or cue-based, using decision trees or maximum entropy models to combine probabilistic features.

A tagset encodes both the target feature of classification, telling the user the useful information about the grammatical class of a word, and the predictive features, encoding features that will be useful in predicting the behavior of other words in the context [21]. Table 1 shows a sample list of frequently used part-of-speech tags in English.

Tag	Part of Speech
AT	article
BEZ	the word *is*
IN	preposition
JJ	adjective
JJR	comparative adjective
MD	modal
NN	singular or mass noun
NNP	singular proper noun
NNS	plural noun
PERIOD	. : ? !
PN	personal pronoun
RB	adverb
RBR	comparative adverb
TO	the word *to*
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle, gerund
VBN	verb, past participle
VBP	verb, non 3rd person singular present
VBZ	verb, 3rd singular present
WDT	*wh-* determiner (*what, which*)

Table 1: Frequently used part-of-speech tags in English

Tagging is much easier to perform than parsing, and accuracy results are very high. Between 96% and 97% of tokens are disambiguated correctly by almost every approach [17]. The intermediate layer of representation that can be obtained from POS tagging can be used for information extraction, question answering, and shallow parsing amongst others.

In our approach, we parsed and grouped POS tags into an intermediate layer, identifying verb phrases, noun phrases, adverbs (of place, time, etc.), and direct objects for interesting and non-ambiguous verbs. Figure 3 shows an example of intermediate layer generation for a simple sentence.

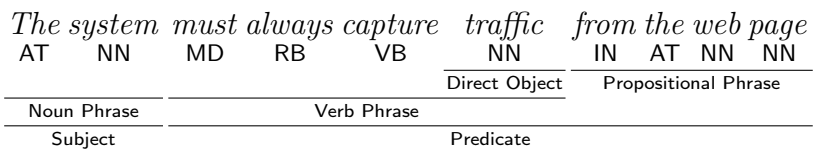


Fig. 3: Sample sentence with POS tags and intermediate semantic analysis

Initially, every verb phrase is selected as a candidate responsibility. Afterwards, a set of rules is applied to filter those phrases that do not contain verbs in the desired tenses (VB, VBN, VBP and VBZ tags), that is to say, verb phrases that contain a verb in the simple past or the simple participle form are dismissed. Next, verb phrases with incomplete or non-existent direct object are also discarded. Additional information such as prepositional phrases contained onto the verb phrase are saved for subsequent processing. Finally, verb phrases are rewritten in a "verb + direct object" form, converting the verb to its infinitive form if necessary.

2.2 Semantic Clustering

Clustering algorithms aim at reducing the amount of data by categorizing or grouping similar data items together [17]. The goal is to place similar objects in the same group and to assign dissimilar objects to different groups. Clustering is unsupervised, that is to say, it does not require training data and the result only depends on natural divisions in the data. These algorithms usually help in the automatic construction of categories or taxonomies, and can be divided into two basic types: hierarchical and partitional.

Hierarchical clustering works by iteratively merging smaller clusters into larger ones, or by splitting larger clusters. The key point is the rule used by the algorithm to decide which two small clusters are merged or which cluster is split. The result is a tree of clusters known as dendogram, which shows how clusters are related.

On the other hand, partitional clustering attempts to directly decompose the data set into disjoint clusters. The function that the algorithm tries to minimize depends on the local structure of the data. Usually the global criteria try to minimize some measure of dissimilarity in the objects within each cluster, while maximizing the dissimilarity of different clusters. Our approach uses a basic partitional clustering algorithm known as K-means [16]. In K-means clustering the criterion function is the average squared distance of the data items from their nearest cluster centroids.

The purpose of using clustering is to semantically categorize candidate responsibilities into groups, based on the noun phrases each verb relates to. These noun phrases may belong to the subject of the sentence or even to the verb phrase itself, functioning as direct object or as a propositional modifier. In any case, the name of the cluster is given by the noun of the noun phrase and corresponds to the component associated to the responsibilities of the cluster. At this point, an experienced software designer may intervene to correct the suggested components and rearrange the clusters.

3 Empirical Evaluation

Empirical evaluation of the approach was performed integrating text engineering techniques and algorithms into a standalone Java application, which provides

a convenient environment for developers to visualize and enhance the results. Figure 4 shows two snapshots of the developed tool, where plain text documents and use case specifications can be loaded and processed within the application, and once potential responsibilities and components are detected, they can be easily drawn, moved and linked throughout the design canvas. Experimental setting, evaluation metrics and results are detailed in this section.

3.1 Experimental Setting

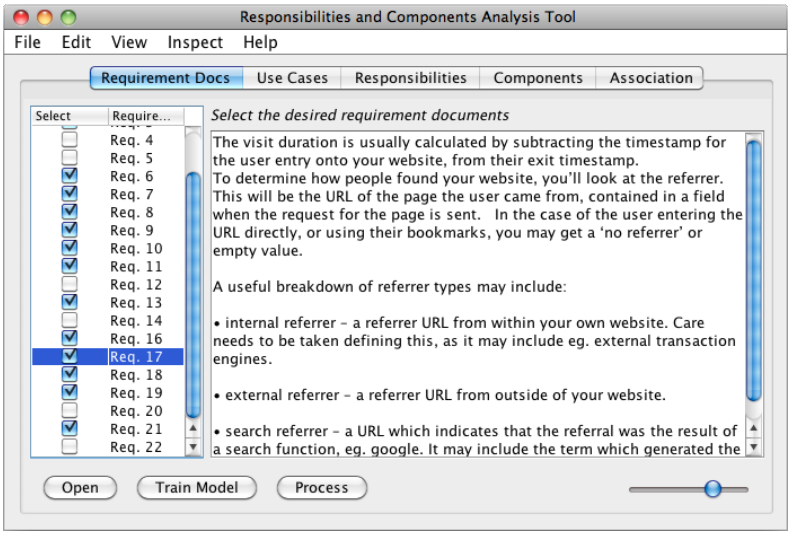
In order to evaluate the approach of semantic analysis for architectural responsibilities and components detection, several experiments were carried out using real data from IBM's support website for the Rational Suite[1] and also from projects developed in our institute[2]. For each of these case studies, requirement documents, UML-like use-cases and software architectures were available and validated by us.

The three case studies correspond to a simple movie rental system, a standard desktop email client application and a full-featured web-analytics system. A set of requirement documents ranging from 12 to 21 was available for the mentioned systems, and a manual analysis and count of real potential responsibilities was performed. A summary of the three case studies analyzed is shown in Table 2.

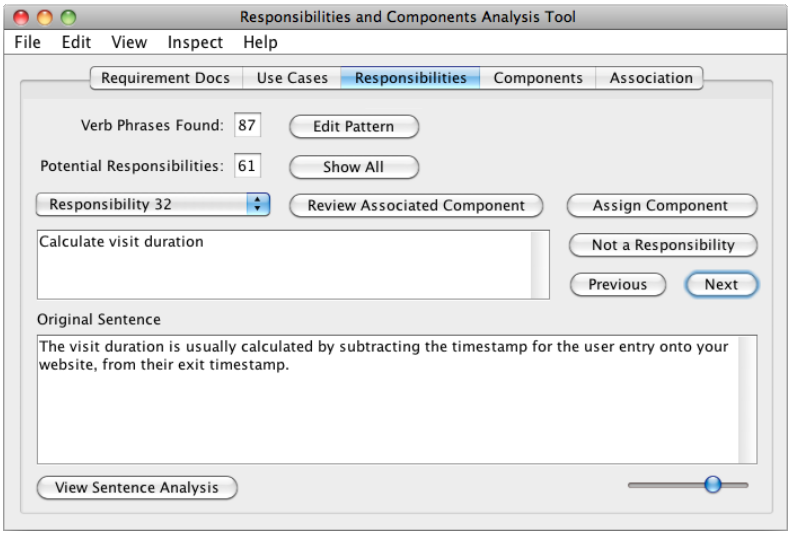
	Number of Requirement Documents	Number of Use Case Specifications	Real Potential Responsibilities	Real Potential Components	Actual Verb Phrases (potential responsibilities)	Actual Noun Phrases (potential components)
Project 1: *Rent System*	12	20	43	6	59	76
Project 2: *Email System*	17	32	50	6	64	81
Project 3: *Web-Analytics System*	21	35	57	8	72	95

Table 2: Case Studies Summary

[1] http://www-01.ibm.com/software/rational/
[2] http://isistan.exa.unicen.edu.ar



(a) Requirement Documents Selection



(b) Responsibilities Detection and Validation

Fig. 4: Responsibilities and Components Analysis Tool

For linguistic analysis, we used the Stanford Parser and the Stanford POS Tagger, a set of efficient Java tools developed by the Stanford Natural Language Processing Group[3]. The Stanford Parser is an implementation of a probabilistic natural language parser. This package implements a highly optimized probabilistic context-free grammar parser (PCFG) based on a factored product model, with separate PCFG phrase structure and lexical dependency, in which preferences are combined by efficient exact inference using an A* algorithm [13,12]. The Stanford POS Tagger is the materialization of the maximum-entropy (CMM) POS tagger described by Tautanova and Manning in [22,21]. The instance of the POS tagger was trained with a modified version of the Penn Treebank corpus[4], adding specific software-related annotations. The tool also allows the user to train the model with any custom corpus. Clustering was performed using an implementation of the K-means algorithm from the Java Machine Learning Library (Java-ML)[5]

3.2 Evaluation Metrics

The purpose of the responsibilities detection approach is to identify if a certain part of a requirement can be mapped to a responsibility and further associated to a component. The results of this mapping process were evaluated using the standard definitions of accuracy, precision, recall, and F-measure metrics [23].

For responsibilities identification, given a test set of documents expressing system requirements, a contingency table is constructed for each binary classification, that is to say, whether a verb phrase is a responsibility or not. Knowing beforehand the real number of potential responsibilities and the real number of verb phrases N, tables were constructed relying on the count of true positives *(TP)* or number of correctly verb phrases detected as responsibilities, false positives *(FP)* or number of verb phrases incorrectly suggested as candidate responsibilities, true negatives *(TN)* or number of verb phrases correctly not detected as responsibilities and false negatives *(FN)* or number of verb phrases incorrectly not suggested as responsibilities. Using these values, the metrics are defined as follows:

$$Accuracy = \frac{TP + TN}{N} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

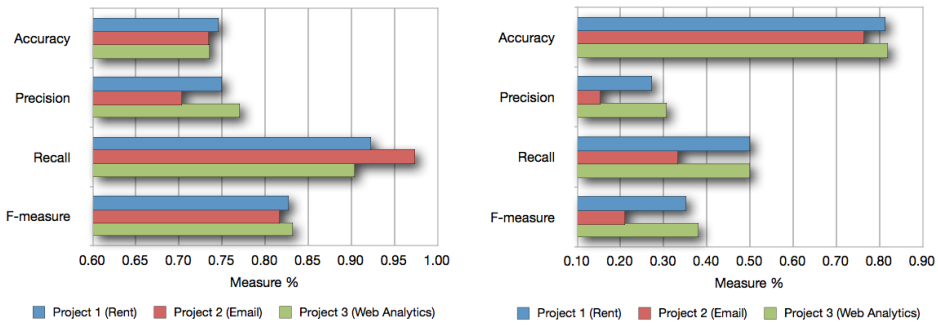
$$F - measure = \frac{2 \times precision \times recall}{precision + recall} \tag{4}$$

[3] http://nlp.stanford.edu/software/index.shtml
[4] http://www.cis.upenn.edu/~treebank/
[5] http://java-ml.sourceforge.net/

3.3 Experimental Results

As mentioned before, the approach was validated using Java implementations of machine learning algorithms, integrated into a support tool. To evaluate the effectiveness of the responsibilities and components detection scheme, we calculated each metric for the three case studies. In a first attempt, the POS tagger used was trained with the Penn Treebank corpus as it is, yielding accuracy and precision rates that were below 55% for responsibilities detection and below 40% for component detection. Afterwards, the model was retrained with a custom version of the same corpus, adding specific domain software-related sentences and annotations. The results obtained outperformed previous values, reaching accuracy rates near to 80% for responsibilities and components detection. However, precision figures differed and did not perform as good as expected for components detection. Figure 5 summarizes the results regarding measures of detection.

With respect to association of responsibilities and components, Figure 6 depicts that this first naïve approach reached an average of 70% for every metric. However, this values might improve significantly providing expert feedback during the very first step of detection, that is to say, the selection of responsibilities, followed by the same procedure for components.



(a) Responsibilities (b) Components

Fig. 5: Measures of Detection

4 Related Work

Natural language descriptions transformed into textual specifications is a common means for capturing requirements in early stages of software development.

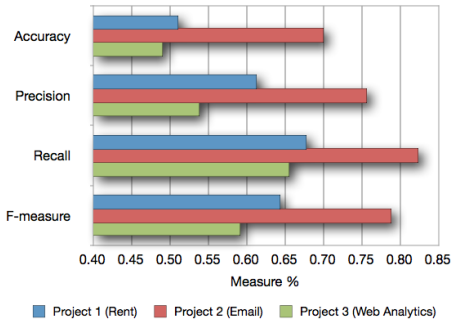


Fig. 6: Association of Responsibilities and Components

On the other hand, architectural software design helps to analyze properties of complex systems, comprising a major issue in the building of new systems with solid foundations, probably based on proven successful experiences. However, in real life scenarios, requirements engineering and architecture modeling may not be as close as they seem to be in theory. Interdependencies and constraints between architectural elements and requirements elements are difficult to understand and trace during software development.

Some approaches have been proposed to bridge the gap between software requirements and architectural design. Grünbacher et al. [6] introduced the Component Bus System and Properties (CSBP) approach as a lightweight model to provide a systematic way of reconciling requirements and architectures. In this work, a simple set of architectural concepts such as components, connectors, overall systems and their properties are used to map requirements to architectures in a straightforward process. In a later article [7], a case study is analyzed with full tool support, showing a possible transition between an EasyWinWin [2] requirements negotiation into a C2-style architectural model [20]. Kof [15] argues on how existing text analysis approaches for ontology extraction can be combined to produce better results than each one on its own. This position paper concludes that natural language processing is mature enough to be applied to ontology extraction in the context of requirements engineering, in spite of the necessary manual intervention, also showing two case studies [14].

The idea of extracting knowledge from text and represent it with formal models has also been approached throughout these years. Ilieva and Ormandjieva [10] proposed an automatic method for the transition of natural language software requirements specifications to formal representations, typically into object oriented designs using intermediate models. Their method consists of three main processing parts, in which firstly the sentences in the text are analyzed, then a semantic network is built by the formal NL presentation and finally, an OO model is deduced. More recently, Ilieva and Boley [9] proposed a method for representing textual requirements with UML diagrams using a similar process.

The Use Case Map notation has been used to assist engineers during software design. In a short paper, Amyot and Mussbacher [1] introduce the idea of bridging the requirements/design gap in dynamic systems with UCMs due to

their versatility and ease of understanding. Later, Mussbacher et al. [18] describe how scenario-based aspects can be modeled at the requirements level unobtrusively and with the same techniques as for non-aspectual systems with the help of UCMs, allowing the visualization of early aspects with these diagrams.

Our proposal combines the need for assistance in the transition between requirements written in natural language and architectural software design, with the ease of use and understandability of the UCM notation. This approach also counts with the appropriate tool support for usability purposes, combined with the possibility to learn from feedback provided by a human designer during the analysis and design process.

5 Conclusions

One of the most critical phases of software engineering is requirements elicitation and analysis. The final success of a software project is influenced by the quality of requirements and their associated analysis since their outputs contribute to higher-level design and verification decisions. Architectural software design helps to analyze properties of complex systems, comprising a major issue in the building of new systems with solid foundations, probably based on proven successful experiences. However, this task is very hard to accomplish and requires a lot of effort and time from both requirement analysts and software architects. In this paper we introduced a first attempt for assistance in the detection of architectural elements within textual requirements specifications, making use of natural language processing techniques and a previous approach on semi-supervised classification of non-functional requirements [4]. High accuracy and precision rates achieved by applying part-of-speech (POS) tagging with domain rules and semantic clustering to textual requirement documents, suggest a great potential for providing assistance to software designers during early stages of development. However, it is very important to remark that writing style of requirement documents and the corpus used to train the POS tagger plays a critic role in this method.

We are planning to continue working on this field, aiming at improving the detection process by providing expert feedback during each step, before classification errors get propagated. Also, the information extracted during the analysis may be useful for inferring execution paths of the scenario represented in each UCM.

References

1. D. Amyot and G. Mussbacher. Bridging the requirements/design gap in dynamic systems with use case maps (ucms). In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 743–744, Washington, DC, USA, 2001. IEEE Computer Society.
2. B. Boehm, P. Grünbacher, and R. O. Briggs. Easywinwin: a groupware-supported methodology for requirements negotiation. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 720–721, Washington, DC, USA, 2001. IEEE Computer Society.

3. R. J. A. Buhr. Use case maps as architectural entities for complex systems. *IEEE Transactions on Software Engineering*, 24(12):1131–1155, 1998.
4. Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Identification of non-functional requirements in textual specifications a semi-supervised learning approach. *Information and Software Techonology*, 52(4):436–445, April 2010. ISSN 0950-5849.
5. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards an unifying framework. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996.
6. P. Grünbacher, A. Egyed, and N. Medvidovic. Refinement and evolution issues in bridging requirements and architecture - the cbsp approach. In *The First International Workshop on From Software Requirements to Architectures (STRAW' 01)*, pages 42–47, 2001.
7. P. Grünbacher, A. Egyed, and N. Medvidovic. Reconciling software requirements and architectures with intermediate models. *Software and System Modeling*, 3(3):235–253, 2004.
8. E. Hull, K. Jackson, and D. Jeremy. *Requirements Engineering*. SpringerVerlag, 2005.
9. M. Ilieva and H. Boley. Representing textual requirements as graphical natural language for uml diagram generation. In *SEKE*, pages 478–483. Knowledge Systems Institute Graduate School, 2008.
10. M. G. Ilieva and O. Ormandjieva. Automatic transition of natural language software requirements specification into formal presentation. In Andrés Montoyo, Rafael Muñoz, and Elisabeth Métais, editors, *NLDB*, volume 3513 of *Lecture Notes in Computer Science*, pages 392–397. Springer, 2005.
11. I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
12. D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Annual Meeting of the Association for Computational Linguistics*, volume 41, pages 423–430, 2003.
13. D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2003.
14. L. Kof. An application of natural language processing to domain modelling: two case studies. *International Journal on Computer Systems Science Engineering*, 20(1):37–52, 2005.
15. L. Kof. Natural language processing: Mature enough for requirements documents analysis? In Andrés Montoyo, Rafael Muñoz, and Elisabeth Métais, editors, *NLDB*, volume 3513 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 2005.
16. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.
17. C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.
18. G. Mussbacher, D. Amyot, and M. Weiss. Visualizing early aspects with use case maps. In *Transactions on Aspect-Oriented Software Development III*, pages 105–143. Springer, 2007.
19. L. Rising and N. S. Janoff. The scrum software development process for small teams. *IEEE Softwware*, 17(4):26–32, 2000.
20. R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. Whitehead Jr., J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow. A component- and message-based architectural style for gui software. *IEEE Transactions on Software Engineering*, 22(6):390–406, 1996.

21. K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 173–180, Morristown, NJ, USA, 2003. Association for Computational Linguistics.
22. K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora*, pages 63–70, Morristown, NJ, USA, 2000. Association for Computational Linguistics.
23. Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1-2):69–90, 1999.
24. Y. Yang, F. Xia, W. Zhang, X. Xiao, Y. Li, and X. Li. Towards semantic requirement engineering. In *WSCS '08: Proceedings of the IEEE International Workshop on Semantic Computing and Systems*, pages 67–71, Washington, DC, USA, 2008. IEEE Computer Society.