Automatically Retrieving of Software Specification Requirements Related to Each Actor

Jantima Polpinij[1](✉) and Khanista Namee[2]

[1] Intellect Laboratory, Faculty of Informatics, Mahasarakham University, Mahasarakham 44150, Thailand
jantima.p@msu.ac.th

[2] King Mongkut's University of Technology North Bangkok, Bangkok 10800, Thailand
khanista.n@fitm.kmutnb.ac.th

Abstract. It is well-known that success or failure of any software is dependent upon requirement analysis. The most significant problem hidden in natural language software requirements may be ambiguity. This can lead to poor design and performance of the final software product and can be time-consuming at the system analysis stage. Previous studies applied natural language processing (NLP) techniques to identify these ambiguities and reduce them to improve the requirement quality. However, this problem has not yet been fully resolved. This study applies NLP to automatically extract the software system requirement specification and visualize an overview of the software that is being developed. The proposed method clusters relevant software requirement sentences of each system user by text mining technique, and then extracts 'actors' and their 'actions' from sentences using NLP techniques. The recall technique is used to evaluate the efficacy of the proposed method. Response time and relevancy of the results are significant factors for software product satisfaction.

Keywords: Requirement analysis · Ambiguity · Text mining · BM25 · Natural language processing

1 Introduction

Requirement engineering (RE) has become a popular research topic in the field of software engineering (SE), where success or failure of any software is dependent upon requirement analysis [1–4]. As a result, since 1960, requirement analysis has become an increasingly trending topic in SE.

The requirement state is a major process in the software development life cycle (SDLC) that includes the phases of planning, analysis, design, development, testing and maintenance [2]. Requirement analysis describes what the stakeholders require or expect from the system, and encompasses those tasks that determine the needs or conditions that must be met for new or altered software products. The results of requirement analysis are called software requirement specification (SRS). This describes what the software will do, how it will be expected to perform, and also the functionality that the product needs to fulfill all stakeholders' (business, users) needs. Therefore, the software requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

In general, when analyzing small software, it is not too difficult to find and extract SRSs from software requirements gathered from many sources such as users' old system and existing related documents. Unfortunately, when working on a large software, this may not be an easy process [1, 2]. The most significant problem hidden in natural language software requirements may be ambiguity [1, 2, 4–13]. This problem has been confirmed as a key factor of software development success. Any ambiguity present in the requirement specification is considered as a defect or failure [13], leading to poor design and performance of the final software product.

Several automatic analysis methods have been proposed to improve the requirement quality. Currently, no technique is available to completely reduce ambiguity caused by lexical, syntactic and syntax ambiguities [13]. Previous studies applied natural language processing (NLP) techniques to identify these ambiguities and reduce them to improve the requirement quality. However, this problem has not yet been fully resolved.

As mentioned, the results of requirement analysis called SRS, often use case diagrams to summarize the details of a system's users (also known as '*actors*') and their interactions with the system. The interactions, often called '*actions*', represent use cases. Automatically generating '*actors*' and '*actions*' from software requirements documents is not easy to perform on a large scale. One possible solution to address this problem is to assemble relevant software requirements sentences in software requirements documents. This may help to see the overall picture of each sub-system domain, and the best way to generate a user case diagram.

As mentioned above, this study aims to automatically extract software specification requirements from software requirements documents. We apply text mining techniques to assemble relevant software requirement sentences, and then extract '*actors*' and their '*actions*' from these sentences using NLP techniques. This study does not show results in the format of diagrams. We only show the actors extracted and their actions in the system.

The structure of this paper is as follows. Section 2 describes literature review. Section 3 describes our dataset and Sect. 4 is the proposed method. Experimental results are presented in Sect. 5. Finally, conclusions are made in the last section.

2 Literature Review

2.1 Related Work

Requirement engineering is the first and crucial phase in the development of software [4]. Success or failure of any software is dependent upon requirement analysis. Manually analyzing software requirements is a time-consuming and labor-intensive task. Therefore, automated systems have been developed to gather user requirements and bridge communication gaps between users and analysts. Unfortunately, the most significant problem hidden in natural language software requirements may be ambiguity [1, 2, 4–13]. Many automatic analysis methods have been proposed to identify these ambiguities and reduce them to improve software requirement quality, but this problem has not been fully resolved. Some studies related to automatic requirement analysis using NLP, text mining, and machine learning techniques to handle language ambiguity in software requirements are illustrated as follows.

Early studies for automated analysis of software requirements were conducted by Osborne and MacNish [6] and Wilson et al. [7]. The former mentioned that ambiguity in requirement specifications causes numerous problems; for example, in defining customer/supplier contracts, ensuring the integrity of safety-critical systems, and analyzing the implications of system change requests. A direct appeal to formal specification does not solve these problems, partly because of the restrictiveness and lack of habitability of formal languages. An alternative approach, described in the paper, is to use natural language processing (NLP) techniques to aid the development of formal descriptions from requirements expressed in controlled natural language. While many problems in NLP remain unsolved, we show that suitable extensions to existing tools provide a useful platform for detecting and resolving ambiguities. Their system was demonstrated through a case study on a simple requirements specification. Wilson et al. [7] mentioned that poorly written requirements can result in software with the wrong functionality. Therefore, the Goddard Space Flight Center's (GSFC) Software Assurance Technology Center (SATC) developed an early life cycle tool for assessing requirements that are specified in natural language. The tool searches the document for terms that the SATC has identified as quality indicators, e.g. weak phrases. Reports produced by the tool are used to identify specification statements and structural areas of the requirements specification document that need to be improved.

Later, in 2005, MacDonell et al. [8] addressed the use of NLP in the requirements analysis and systems design processes. They developed a prototype toolset that can assist the systems analyst or software engineer to select and verify terms relevant to a project. Also, they described the processes employed by the system to extract and classify objects of interest from requirements documents. These processes are illustrated using a small example.

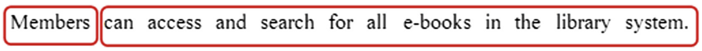
In 2015, Btoush and Hammad [11] examined the problem of extracting entity-relationship (ER) elements from natural language specifications using NLP. Their approach provided the opportunity of using natural language documents as a source of knowledge for generating an ER data model. A structural approach was used to parse specification syntactically, based on a predefined set of heuristics rules. Extracted words with their Part of Speech (POS) mapped into entities, attributes, and relationships are the basic elements of ER diagrams.

In 2016, Wang [12] proposed an approach to automatically analyze software requirement specifications (SRSs) and extract the semantic information. This approach used a machine learning and ontology based semantic role labeling (SRL) method. First, some common verbs were calculated from SRS documents in the E-commerce domain, and then semantic frames were designed for these verbs. Based on the frames, sentences from SRSs were selected and labeled manually, and the labeled sentences were used as training examples in the machine learning stage. Besides the training examples labeled with semantic roles, external ontology knowledge was used to relieve the data sparsity problem and obtain reliable results. Based on their proposed method and WordNet corpus, the senses of nouns and verbs were identified in a sequential manner through the K-nearest neighbor approach. Then, the senses of the verbs were used to identify the frame types. After training the SRL labeling classifier with the maximum entropy method, some new features were also added based on word sense, such as hypernyms and hyponyms of word senses in the ontology. Experimental results showed that their approach for automatic functional requirements analysis was effective.

In 2018, Li [14] presented an automated technique for extracting features and variability to provide reliable solutions to simplify the work of domain analysis. The main mechanism of the proposed method was NLP and machine learning techniques.

2.2 Understanding of Software Requirement Sentence

Software requirements are often written with a simple language structure and should be easy to understand. A software requirement sentence consists of two parts as ‘*actor*’ and ‘*action*’. See in Fig. 1. The actor performs the action. This structure is called ‘*Actor-Action-Object*’ [15], which is used in the Object-Oriented Software Processes through the Use Case Diagram.



Members can access and search for all e-books in the library system.

Fig. 1. An example of the position of “*actor*” and “*action*” in a requirement sentence

This structure is similar to English sentence structure that consists of two main parts: subject and predicate. The subject is the person or thing (who/what) that performs the action, while the predicate is the part of the sentence that contains a verb or a verb phrase and its complements. Therefore, if comparing between the English sentence and the software requirement sentence, the subject in the English sentence is comparable to the ‘*actor*’ of the software requirement sentence, while the predicate in the English sentence is comparable to the ‘*action*’ of the software requirement sentence.

3 Dataset

The dataset used for our experimental stage is a set of software requirements related to a library website written in English and contains over 50 pages. Each page contains about 40 lines. This dataset has been analyzed by system analysts, with a final SRS case diagram that details a system’s users and their interactions. For a legal policy, we cannot show details of the software requirements used in this study.

4 Research Method

4.1 Searching for the System’s Users (or Actors)

This section describes the process of finding and extracting all ‘actors’ in a software requirements collection. An ‘actor’ can be compared to the ‘subject’ in the English sentence structure. In general, the ‘subject’ should be a noun, pronoun, or noun group.

To find all actors, we then apply Stanford NLP Parser that works out the grammatical structure of sentences [16] to generate a parse tree of rules. Then, we extract only the sets of 'noun phrase (NP)' or 'word' that are located at the position of the sentence subject. An example is illustrated as Fig. 2.

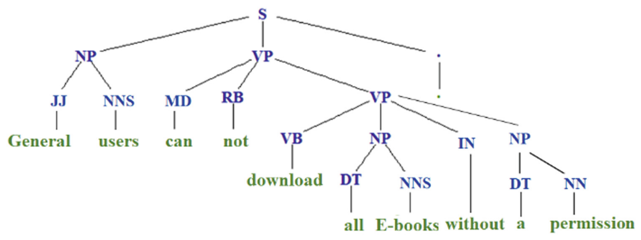


Fig. 2. An example of parsing requirement sentence to extract 'NP' considered as 'actor'

After obtaining all 'NPs' or 'words' that may be considered as actors, these NPs and word groups can make distinctions as four different actors of the system. These are administrators, librarians, members, and general users.

4.2 Using 'Actors' for Searching Their Relevant Software Requirement Sentences

After obtaining four different actors of the system, these actors are used to search their relevant software requirement sentences by keyword search technique. This is a process of preliminary grouping for relevant software requirement sentences. The overview picture of the expected results of this stage is shown as Fig. 3.

4.3 Retrieving Relevant Software Requirement Sentences of Each Actor

After some relevant requirement sentences of each actor are retrieved (mentioned in Sect. 4.2), these sentences become queries that are used to find other relevant requirement sentences of the actor. This solution helps the system analyst team to visualize a preliminary overview of the software that is being developed. Conse quently, this may increase the chances of completely analyzing and designing the software. Each process of retrieving all relevant software requirement sentences can be described as follows.

The software requirement sentences are first segmented to a meaning unit. In this study, the meaning unit of language is words. afterward, the program performs a process of stop-word removal. Finally, the software requirement sentences are represented as a vector space model, called bag of words (BOW). Later, this study applies BM25 [17] which is a ranking function for retrieving relevant software requirement sentences of each actor. BM25 is often used as a text mining technique to retrieve relevant documents based on similarity analysis.

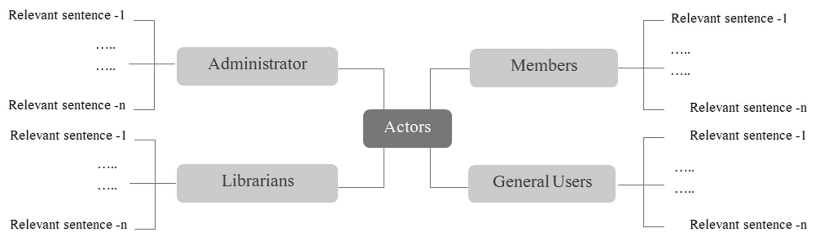


Fig. 3. The expected results of preliminary grouping of relevant software requirement sentences for each actor

In BM25 formula, it requires only *document frequency* (*df*). The *df* is number of documents where the term *t* appears. Instead of regarding the inter-relationship between the query terms within a software requirement sentence. The BM25 formula is:

$$BM25(r_1, r_2) = \sum_{i=1}^{|r_1|} idf(q_i) \times \left(\frac{tf(q_i, r_2) \times (k+1)}{tf(q_i, r_2) + k \times (1 - b + b \times \frac{|r_2|}{r_2 l_{avg}}} \right) \quad (1)$$

Let r_1 be software requirement sentence of an actor and r_2 be software requirement sentence that may be related to that r_1. Therefore, $tf(q_i, r_2)$ is the term frequency, where it is defined as the number of occurrences that the terms q-th of $r_1$appears in r_2. Indeed, $|r_2|$ is the length of $r_2$in words, while $r_2 l_{avg}$ is the average software requirement sentence length for all the software requirement sentences in the corpus. For, k and b, they are free parameters. They are used to control the weighting between $tf(q_i, r_2)$ and the normalized software requirement sentence length. Generally, the values of k and b should be in the range of $1.2 < k < 2.0$ and $0.5 < b < 0.8$ [17, 18]. This study uses the values of k and b at 2.0 and 0.8 respectively. They are the same values used in [18].

For $idf(q_i)$, it is the *inverse document frequency* of the term q-th of r_1. It can be calculated by following formula.

$$idf(q_i) = \log\left(\frac{N - df(q_i) + 0.5}{df(q_i) + 0.5}\right) \quad (2)$$

where N is the total number of sentences in the corpus, while $df(q_i)$ is the number of sentences containing the term q-th of r_1.

In general, the similarity score should be between 0 and 1. However, when using the BM25 technique to estimate the similarity score, it is possible that this technique can return a score greater than 1.0. Similarity scores should be normalized to allow a comparison of different similarity values using a single scale. Normalizing similarity scores helps to remove the mean and scale to the similarity score variance. To normalize the BM25 similarity scores in the range [0, 1], the following function also applies in this case.

$$f(x) = x/(1 + x) \quad (3)$$

where x is the similarity score generated by BM25.

4.4 Extracting of 'Actors' and Their 'Actions'

After completely retrieving all relevant software requirement sentences into their specific clusters, the software requirement sentences in each cluster are performed by Stanford NLP Parser [16] to generate their parse trees. Afterward, we extract '*verb phrases (VP)*' as actions of each actor. Some examples of the expected results are shown as Table 1.

Table 1. Some examples of the expected results.

Actors	Actions
Members	Need to register as member to get the authorization
	Need to login for access the system
	Can search e-books in the system
	…

5 The Experimental Results

This section evaluates the proposed method using recall, precision, and *F*-measure techniques. Recall gives a measure of how accurately our proposed method can identify the relevant data, while precision is the closeness of the measurements to each other. The *F*-measure is a single score that balances both the concerns of precision and recall as one number.

In this study, there are two evaluations. First, the performance of extracting all actors from the software requirement collection is evaluated, while the second evaluation measures the results of retrieving relevant software requirements of each actor.

5.1 The Evaluation of Extracting All Actors from the Software Requirement Collection

This evaluates the efficiency of extracting '*actors*' from the software requirement documents. After performing this process, the NPs or word groups extracted are distinct, and all actors of the system can be obtained. They include *administrators*, *librarians*, *members*, and *general users*. Results are shown in Table 2.

Table 2. The results of extracting actors.

Results	Recall score
Extracted actors	1.00

In Table 3., the results are good in terms of recall. However, some errors can be found. For example, consider the following phrase, 'the processes can be initiated'. The word ‘processes’ is not an actual actor but it is extracted because it is an NP that is located at the position of the subject of the sentence. Therefore, some constraints should be provided to handle this issue and improve the quality of extracting actors from the software requirement documents.

5.2 The Results of Retrieving Relevant Software Requirements of Each Actor

After testing via the recall, precision, and F-measure, the results of retrieving the relevant software requirements relating to each actor are presented in Table 3.

Table 3. The results of retrieving relevant software requirements of each actor

Actors	Recall	Precision	F-measure
Administrators	1.00	0.77	0.87
Librarians	1.00	0.81	0.89
Members	1.00	0.71	0.83
General users	1.00	0.74	0.85

Consider Table 3. Although the results are satisfactory, they still have some errors. Consider the following software requirement sentence.

“*The members can search e-books from the system, and they can also download these e-books.*”

In the example sentence above, ‘*the members*’ is an ‘*actor*’ in the system, while the pronoun ‘*they*’ indicates ‘*the members*’. Unfortunately, our method cannot recognize the meaning of the word ‘*they*’ and to which ‘*actor*’ the word ‘*they*’ refers. Consequently, the proposed method of retrieving relevant software requirements of each actor can lead to failure analysis.

The failure rate recognized in this study requires improvement. Two solutions are suggested for further analysis. A large number of language patterns is also necessary. However, developing language patterns with only the training set is not sufficient, and a variety of sentence patterns is required.

6 Conclusion

Requirement analysis is a major process in the software development life cycle that describes the functionality the product needs to fulfill all stakeholders’ (business, users) needs. Requirement analysis encompasses tasks that determine the needs or conditions needed to meet new or altered software products. On a small software, it is not difficult to find and extract SRSs from software requirements. By contrast, for large software, this is never an easy process. The most significant problem hidden in natural language software requirements may be ambiguity. This problem has been confirmed as a key factor of software development success because ambiguity can lead to poor design and performance of the final software product. Previous studies applied NLP techniques to identify these ambiguities and reduce them to improve the quality of software requirements. This issue is the challenge in this study. We present a method of automatically retrieving software requirements of each actor. Doing this may be an easier way to perform requirement analysis tasks. The main mechanisms of the proposed method are NLP and text mining techniques. Then, we apply NLP techniques to extract all actors of the system, and text mining techniques to find the relevant software requirements of each actor. Recall, precision, and F-measure techniques are used to evaluate the efficacy of the proposed method. Response time and relevancy of the results are significant factors regarding satisfaction.

Acknowledgement. This research was funded by King Mongkut's University of Technology North Bangkok. Contract no KMUTNB-64-DRIVE-34. We would like to deliver our greatest appreciation for their support.

References

1. Kamsties, E.: Understanding Ambiguity in Requirements Engineering. Engineering and Managing Software Requirements, pp. 245–266 (2005)
2. Kerry, E., Delgado, S.: Applying software engineering practices to produce reliable, high-quality and accurate automated test systems. IEEE Autotestcon (2009)
3. Badariah, S., Shamsul, S., Ghani, A.A.A.: A new maturity model for requirements engineering process: an overview. J. Software Eng. Appl. **5**, 340–350 (2012)
4. Mishra, A., Awal, A., Elijah, J., Abdulg, A., Gana, Rabiu, I.: Automation of requirement analysis in software engineering. Int. J. Recent Innov. Trends Comput. Commun. **5**(5), 1173–1188 (2017)
5. Liu, G., Huang, S., Piao, X.: Study on requirement testing method based on alpha-beta cut-off procedure. In: International Conference on Internet Computing in Science and Engineering, vol. 1, pp. 396-402 (2008)
6. Osborne, M., Macnish, C.K.: Processing natural language software requirement specifications. In: Processing Natural Language Software Requirement Specifications (1996)
7. Wilson, W.M., Rosenberg, L.H., Hyatt, L.E.: Automated analysis of requirement specifications. In: Proceedings of the 19th International Conference on Software Engineering, pp. 161–171 (1997)
8. MacDonell, S.G., Min, K., Connor, A.: Autonomous requirements specification processing using natural language processing. In: 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (2005)
9. Verma, K., Kass, A.: Requirements analysis tool: a tool for automatically analyzing software requirements documents. In: Sheth, A., et al. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 751–763. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88564-1_48
10. Vinay, S., Shridhar, A., Orashanth, D.: An Approach towards Automation of Requirements Analysis (2009). https://www.researchgate.net/publication/44259706
11. Btoush, E., Hammad, M.: Generating ER diagrams from requirement specifications based on natural language processing. Int. J. Database Theory Appl. **8**(2), 61–70 (2015)
12. Wang, Y.: Automatic semantic analysis of software requirements through machine learning and ontology approach. J. Shanghai Jiaotong Univ. (Sci.) **21**(6), 692–701 (2016). https://doi.org/10.1007/s12204-016-1783-3

13. Vimalraj, J.T., Seema, B.: Identification of ambiguity in requirement specification using multilingual word sense. Int. J. Adv. Res. Comput. Commun. Eng. **5**(6), 386–388 (2016)
14. Li, Y.: Feature and variability extraction from natural language software requirements specifications. In: Proceedings of the 22nd International Systems and Software Product Line Conference, vol. 2, pp. 72–78 (2018)
15. Mishra, A., Sureka, A.: Automatic detection of semantic inconsistency between BPMN process model and SBVR rule model. Computer Science (2015)
16. Stanford Parser. https://nlp.stanford.edu/software/lex-parser.shtml. Accessed 01 Nov 2020
17. Lee, C.Y., Hu, D.D., Feng, Z.Y., Yang, C.Z.: Mining temporal information to improve duplication detection on bug reports. In: IIAI 4th International Congress on Advanced Applied Informatics, pp. 551–555 (2015)
18. Yang, C.Z., Du, H.H., Wu, S.S., Chen, X.: Duplication detection for software bug reports based on bm25 term weighting. In: Conference on Technologies and Applications of Artificial Intelligence, pp. 33–38 (2012)