Journal of Theoretical and Applied Information Technology
30th April 2018. Vol.96. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN APPROACH FOR DETECTING SYNTAX AND SYNTACTIC AMBIGUITY IN SOFTWARE REQUIREMENT SPECIFICATION

[1] ALI OLOW JIM'ALE SABRIYE, [2*]WAN MOHD NAZMEE WAN ZAINON
[1]Faculty of Computing, SIMAD University, Mogadishu, Somalia
[2]School of Computer Sciences, Universiti Sains Malaysia, Malaysia
E-mail: [1]colowyare2@simad.edu.so, [2]nazmee@usm.my

ABSTRACT

Software requirements are considered to be ambiguous if the requirements statement could have more than one interpretation. The ambiguous requirements could cause the software developers to develop software which is different from what the customer needs. The focus of this paper is to propose an approach to detect syntax and syntactic ambiguity in software requirements specification. In this paper, Parts of speech (POS) tagging technique has been used to detect these ambiguities. A prototype tool has been developed in order to evaluate the proposed approach. The evaluation is done by comparing the detection capabilities of the proposed tool against human capabilities. The overall results show that the humans do have some difficulties in detecting ambiguity in software requirements, especially the syntactic ambiguity and software requirements that contains both syntax and syntactic ambiguity in one sentence. The proposed tool can definitely help the analyst in detecting ambiguity in Software requirements.

Keywords: *Part of speech tagging, Syntax ambiguity, Syntactic ambiguity, Software requirements specification.*

1. INTRODUCTION

Software requirement specification (SRS) document is an important document with a full description about functional and non-functional requirements of a software sys-tem to be developed [1]. SRS helps the analyst to understand the customer needs and it is the base document for other software development activities [2].

Around 87.7 % of software requirements are documented using natural language (NL) [5]. SRS quality has the highest contributing factor of the success or failure for the project to be developed. There are 9 quality measures of SRS, and one of these quality measures is that SRS must be unambiguous [6]. SRS document is unambiguous if and only if the requirements contained in this document has only one interpretation, however, most of NL SRSs contain ambiguous requirements which can negatively affect in all the software development process [7].

Ambiguity in SRS can cause some big issues that can affect the software development process because different interpretations can turn into bugs (such as design, functional, logical, performance, requirement or user interface bugs) if not detected and solved at the early phases of software development [8]. NL SRS ambiguity can also cause negative significances for the whole software development, this problem can lead to a project failure, highly maintenance costs, delayed product releases or developing a software which does not fit to the user requirements.

The scope of this paper is limited to detecting syntax and syntactic ambiguity in natural language requirement specification. This study is different from other works where it focus on the type of ambiguity that can be detected using POS tagging techniques. Other works are using different detection

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

techniques and caters for different type of ambiguity. Details about related works can be found in section 3 where we will discuss all relevant works related to detecting ambiguities.

2. AMBIGUITY IN NL SRS DOCUMENT

Ambiguity can be defined as a statement which has more than one interpretation [9]. The definition of ambiguity in software requirements perspective is a requirement which have more than one interpretation despite the knowledge of the reader about requirements engineering context. Based on a study by Sandhu, G. & Sikka, S.[10], there are five different types of NL SRS ambiguity;

2.1 Lexical Ambiguity

Lexical Ambiguity refers to a situation where there is a single word that can have several meanings. Lexical ambiguity can be subdivided into two- homonymy lexical ambiguity and polysemy lexical ambiguity. Homonymy is a word which has distinct meanings and etymologies as well. For example, the word bank means a depository financial institution; sloping land; a flight maneuver. Polysemy word has several interrelated meanings but etymology i.e.: the word green may mean color green or unripe [12]. For example, the requirements "*The users of the system are customers and administrators. They login to the system*." The word "*They*" in the second sentences is a lexical ambiguity because of unclear reference. The reference of the word They can be either They (the customers) login to the system, or they (The administrators) login to the system or both the customers and administrators' login to the system.

2.2 Syntactic or Structural Ambiguity

Syntactic or Structural Ambiguity is a type of ambiguity that occurs if a sentence can be parsed in several ways with a different meaning. For example: the sentence "*I saw the girl with the telescope*" can be parsed into two different ways:

1) the girl has a telescope with her.
2) I used a telescope to see the girl.

This type of ambiguity is generated when the sentence contains vague words. adjectives or adverbs which can be considered as vague words [13]. The requirements "*The Software must be reusable*." and "*The Software will display the map quickly*." are example of structural ambiguities because the words reusable (adjective) and normally (adverb) are vague words that can have different interpretations.

2.3 Semantic/Scope Ambiguity

Semantic/Scope ambiguity is an ambiguity that occurs when the sentence has several interpretations within its context without containing lexical, structural and syntactic ambiguity. For example, the requirement "*All users enter a password code*" is scope ambiguity because when the scope of all includes the scope of a, the meaning of the requirements becomes all users enter the same password code, When the scope of *a* includes the scope of *all,* the sentence meaning is each user enters a password code.

2.4 Pragmatic Ambiguity

Pragmatic ambiguity is an ambiguity type that focuses on the relationship between the sentence meaning and its context. It depends of the requirement's context including the knowledge of the requirement's reader. For example, two readers that have different backgrounds can interpret a single requirement into two different ways.

2.5 Syntax Ambiguity

Syntax ambiguity is an ambiguity occurs if the sentence is in passive voice form which the user element is not specified inside the sentence or the sentence not end with a full stop/period ".". For example, the sentence "*Student records should be documented*." is a syntax ambiguity, because it is not clear the user who will document the student records [14].

3. RELATED WORK

Computer scientists have proposed different methods and techniques can be applied to solve ambiguity in SRS. Three main categories can be summarized the techniques applied to solve ambiguity, among these categories Natural Language Processing(NLP) based techniques for solving ambiguity [11].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

An example of NLP based technique for solving ambiguity is [16] work. They have analyzed NL SRS document using NLP Standard POS tagger and parser to generate activity and sequence diagrams. Their method allows the user to reduce ambiguity. They used POS tagging like this research, but with different result: It reduces ambiguity by generating activity and sequence diagrams. The drawback of their method is lack of automatically highlighting the ambiguous sentences in NL SRS document.

The work of [13] can be an example of automatic ambiguity detection in NL SRS document. They developed a tool that detects ambiguities and gives explanation about the ambiguity source. The researchers used regular expressions, part of speech tagging (POS) and list of ambiguous words from ambiguity handbook to mark ambiguous words in NL SRS document and give explanation about the ambiguity sources. The drawback of their method was lack of calculating the percentage of the detected ambiguity.

Another example of using POS tagging for ambiguity detection is [5]. These re-searchers developed a technique that checks the validity of the requirements and detect only lexical ambiguity. In this method, a dictionary was used to compare with the words of one line in NL SRS document and store it in a data structure. They detected lexical ambiguity by checking the word in a single sentence that has more than one type of part of speech. The limitation of this approach is that it can work efficiently if the NL SRS document contains not more than six words.

The researcher [17] illustrated many NLP tools used for finding defects and deviations in SRS documents. These tools aim to check the Quality of requirements, ambiguity, uncertainty, quality of user stories and quality of use cases. Among these tools Dowser and HEJF that focus on ambiguity in SRS document has been selected for further discussion.

Dowser tool is a tool designed to identify ambiguities in SRS document using parsing technique. Initially, Dowser parse the requirements using constraining grammar. In addition to that, object oriented analysis model of the system will be developed by creating classes, methods, variables and associations. Lastly, the model will be presented for the reviewers to detect the ambiguity [18]. However, this technique does not consider detecting ambiguity automatically; the human makes the final decision of the ambiguity.

Qualicen (Formally known as HEJF) is a commercial tool that detects the possible quality defects Such as slash, ambiguous adverbs and adjectives, negative words, non-verifiable term, subjective language, Imprecise phrase, requirements, comparative requirements, Vague pronouns, Loophole, UI detail and long sentence [19]. Qualicen detects software requirements mismatch certain requirements engineering principles using POS tagging, morphological analysis and dictionaries. This tool dis-plays warning messages that contains description of the detected smell to the user.

The authors [20] carried out systematic review about NLP tools for resolving ambiguities in SRS document. This review contains 8 NLP tools used to deal with ambiguity problem in SRS document. From these 8 identified tools, RESI, SR-Elicitor and NL2OCL has been selected to discuss in further since the selected tools are most related tools to the research area.

RESI is a tool developed by [21] which was designed to help software. It provides a dialog system that alerts the user when the SRS document is ambiguous, faulty, or inaccurate. It offers the possible interpretations of each word in the SRS document, so that the software analyst can change the word. RESI tool detects the possible nominalizations contained in SRS document and suggests verbs that can be used instead of nominalizations. Also, RESI avoids incomplete process words, similar meanings, nouns without reference index and wrongly used universal quantifiers. How the RESI tool works is like this: first, RESI imports the SRS document as a graph, Second, each word in the SRS document is tagged with part of speech (POS) to check either it is noun or verb. After POS tagging done automatically, the system user can adjust the tags manually if wanted. Finally, RESI applies the ontologies WordNet, ResearchCyc, ConceptNet and YAGO to detect ambiguous, faulty, and inaccurate.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SR-Elicitor is a tool developed by [22] to automate the requirements elicitation process, solve ambiguous problem in SRS document and generate a controlled representation. The researchers of SR-Elicitor used Semantic of Business Vocabulary (SBVR) and Rules to capture NL SRS document. Fig. 1 Shows the approach used to translate NL software requirements into SBR requirements. After translating NL to SBR, SR-Elicitor parses NL SRS document, the parsing process includes lexical parsing using tokenization, sentence splitting, Parts-of-Speech (POS) Tagging and Morphological Analysis, Syntactic and Semantic Interpretation. The next phase of SR-Elicitor tool is the process of extracting SBVR vocabulary elements such as noun, verb and individual concepts and object types from the input. After that, SBVR rules are generated from SBVR vocabulary. This phase is important to extract SBVR requirements and apply semantic formulation. The final step of SR-Elicitor is applying structured English notation. In this step, object types are underlined, verb concepts will be in italic form, SBVR keywords will be bolded and individual concepts will be double underlined. Fig. 2 Shows SBVR rule representations of software requirements.

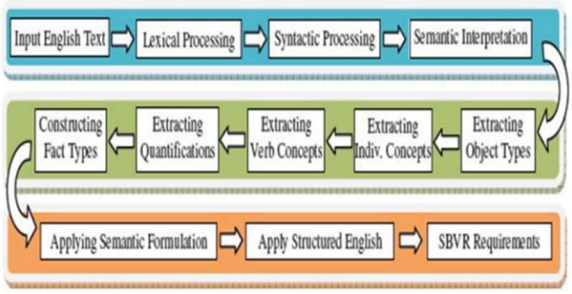

Fig. 1. Approach to translate NL SRS document into SBVR requirements [22]

Details
It is obligatory that the system *shall let*, each Patron who *is logged* into the Cafeteria Ordering System, *place* at least one order for at least one or more meals.
It is obligatory that the system *shall confirm* that the 'Patron' *is registered* for payroll deduction to *place* at least one order.
If the Patron *is* not *registered* for payroll deduction, It is obligatory that the system *shall give* the Patron options to *register* and *continue placing* at least one order, to *place* at least one order for *pickup* in the cafeteria, or to *exit* from the COS.
It is obligatory the system *shall prompt* the Patron for the *meal date*.
If the *meal date is* the current date and the current time *is* after the order *cutoff* time, it is obligatory that the system *shall inform* the Patron that it's too late to *place* at least one order for today.
It is possibility that the Patron *may change* the *meal date* or *cancel* the order.
It is obligatory the Patron *shall specify* whether the order *is* to *be picked* or *delivered*.
If the order *is* to *be delivered* and there still *are* available *delivery times* for the *meal date*, it is obligatory that the Patron *shall provide* at least one valid *delivery location*.

Fig. 2. SBVR rule representation of software requirements

NL2OCL is a tool designed to solve syntactic ambiguity [23]. This project aims to translate NL SRS document of software constraints to formal constraints. This translation requires two inputs: English specification of a constraint and UML class model. The developers of the tool used Stanford POS tagger and the Stanford Parser for syntactic analysis of English specification. After that, the output of syntactic analysis is passed to the semantic analyzer to do detailed semantic analysis. The UML class model is important to do syntactic analysis, notably to resolve syntactic ambiguity.

4. PROPOSED APPROACH

To achieve the objectives of this research, an ambiguity detection approach that can automatically detect syntax ambiguity and syntactic ambiguity has been pro-posed. Both of these ambiguities can be detected using POS tagging technique. The proposed approach of this research takes NL SRS document as an input, processes the NL SRS document by using POS tagging to detect ambiguity and highlights the detected ambiguity as an output.

The proposed approach consists of three major components, which are the pre-processing, processing, and post processing components.

Pre-processing component contains NL SRS document that will be uploaded into the proposed approach. This component makes the raw data (NL SRS document) to be ready to be processed in the next phase.

Processing component contains the operations performed to detect syntax and syntactic ambiguous software requirements. It consists of two sub parts, POS tagging and ambiguity detector. Both part works together to detect syntax and syntactic ambiguity in the document.

Part of speech (POS) tagging tags words of a sentence to its English parts of speech equivalents, for example: “The system can avoid errors.” is tagged as “The/DT sys-tem/NN can/MD avoid/VB errors/NNS.”. Also, POS gives the basic form of every word. This form is an important for detecting syntax ambiguity, especially passive voice. The tags “VBZ”, “VBN”, “VBP”, “VBG”, “VBD”, “MD”, “VB”,“JJ” and “RB” are very important to detect

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

syntactic and syntax ambiguity in NL SRS document.

Ambiguity detector steps are new proposed steps to classify the ambiguity into syntactic and syntax ambiguities. It consists of 7 steps to detect ambiguity using POS tagging. Each step has specific task and role in the ambiguity detection process.

Step1: Retrieve the browsed NL SRS document line by line.

This step implies the process of extracting and attaching the NL SRS document to make it ready for processing. The attached document can be a collection of paragraphs. In this stage, the paragraphs will be split into a list of sentences and saved in a data structure called "*Sentence_Splitter*". Then, the sentences are counted by counting the number sentences contained by *Sentence_Splitter*. The counted sentences are stored in a data structure called "*Sentence_Counter*"

Step-2: Mark each sentence with POS tagger.

This step is the process of tagging each sentence contained in *Sentence_Splitter* to its equivalent English eight parts of speech. The tagged sentences will be kept in a data structure named "*Tagged_Sentence".* This is the core of the research which will be used for ambiguity detection, notably syntax and syntactic ambiguity.

Step-3: Detect syntax ambiguity and store in a data structure called 'syntax' by:

(a) Checking if any sentence in the Tagged_Sentence not contains full stop

This technique is being done by ensuring that each sentence in *Tagged_Sentence* does not contain the tag "./." at the end. If a sentence that does not have this tag is detected, it will be considered as syntax ambiguity.

(b) Checking if any sentence in the Tagged_Sentence is passive voice.

This technique is very important for detecting passive voice sentences which causes ambiguity in SRS, particularly syntax ambiguity. In this process, the passive voice formulas and its POS tagging are matched. Table 1 shows passive voice formulas and its POS equivalent. If *Tagged_Sentence* contains the tags in column 3 (POS passive voice detection formula POS equivalent) of Table 1, it will be considered as syntax ambiguity.

Step-4 Detect syntactic ambiguity and store in a data structure called 'syntactic' by:

Checking if any sentence in the Tagged_Sentence is an adjective or adverb. This technique can be done by detecting any sentence in Tagged_Sentence that contains the adjective tags "JJ" and adverb tag "RB".

Table 1 Passive voice formulas and its POS equivalent

Tense	Passive voice Formula	passive voice detection formula POS equivalent
Simple present	am, is, are + past participle	VBZ +VBN VBP+VBN
Present continuous	am being, is being, are + past participle	VBZ+VBG+VBN VBP+VBG+VBN
Simple past	was, were + past participle	VBD+VBN VBZ +VBN
Past continuous	was being, were being + past participle	VBD+ VBG+VBN
Present perfect	has been, have been + past participle	VBZ+ VBN+ VBN VBP+ VBN+ VBN
past perfect	had been + past participle	VBD+ VBN+ VBN
Future	will be + past participle	MD+ VB+ VBN
Future perfect	will have been + past participle	MD+ VB+ VBN+ VBN

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Step-5 Continue step 3, and step 4 for each sentences of NL SRS up to the end of SRS document.

Step-6 Calculate the total number of syntactic and syntax ambiguities by using the following formulas:

$$\textit{Syntactic Ambiguity} = \sum_{sc=1}^{sc-1} \text{syntactic}$$

$$\textit{Syntax ambiguity} = \sum_{sc=1}^{sc-1} \text{syntax}$$

where SC= *Sentence_Counter* for all of the three formulas

Step-7 Calculate the percentage of ambiguities detected and non-ambiguous sentences by using the following formulas: -

Percentage of detected syntactic ambiguity

$$= \frac{\text{Syntactic Ambiguity}}{\text{Sentence\_Counter}} * 100$$

Percentage of detected syntax ambiguity

$$= \frac{\text{Syntax Ambiguity}}{\text{Sentence\_Counter}} *100$$

None ambiguity

$$= 100 - \left(\frac{\text{syntax Ambiguity}}{\text{Sentence\_Counter}} + \frac{\text{Syntactic Ambiguity}}{\text{Sentence\_Counter}} \right)$$

Post processing component is the result generation component. The detected syntax ambiguity will be colored as red, syntactic as blue and both syntax and syntactic as yellow. Also, a percentage of detected ambiguity will be displayed in a chart.

5. EVALUATION

The evaluation method used in this research is adopted from a study by Nigam et al. [8]. A repository that consists of 20 English software requirements sentences was created in order to evaluate the proposed approach in this research. These sentences consist of 5 syntax ambiguous software requirements, 5 syntactic ambiguous software requirements, 5 sentences that contain both syntax and syntactic ambiguity and 5 sentences that do not contain neither syntax ambiguity nor syntactic ambiguity-ty. These sentences are not created by our own but the sentences were taken from real software requirements in SRS documents from scribd.com website. Two experiments have been conducted in order to evaluate the proposed approach. These experiments are human detection experiment and automatic ambiguity detection experiment.

5.1 Human Detection Experiment

Human detection experiment was conducted to test and check if the human has difficulties in detecting syntax and syntactic ambiguities in SRS document. 63 participants from the School of Computer Sciences at Universiti Sains Malaysia (USM) were taking part in this experiment. The participants come from 3 main groups of students:

1. 35 undergraduates second year full time requirements engineering students whom are currently taking CPT243-Requirements analysis and modeling course.
2. 18 Computer Science Master Students
3. 10 PhD Computer Science Students

Evaluation forms were given to all the participants. The evaluation form contains 20 sentences in the Dataset selection. Also, the evaluation form contains the explanation and examples of syntax ambiguity and syntactic ambiguity that can be found in SRS document. The participants were asked to detect and mark syntax and syntactic ambiguities in the form. The subjects were given instructions to mark the ambiguity.

5.2 Automatic Ambiguity Detection Experiment

The aim of this experiment is to evaluate the performance of ambiguity detecting tools that has been developed based on the proposed approach. It consists of three min windows: editor window, graphical window and textual result window. Editor window is the window which allows the users to browse and process SRS document for detecting ambiguity. Also the users can directly write the requirements inside the editor window.

Textual result window displays detected ambiguous word in colored form. The syntax ambiguity will be colored as red, syntactic as yellow

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and lexical as blue. Graphical result window shows the detected ambiguity percentage in a graphical representation. The tool works as followings: first, the user uploads NL SRS document as a plain text format. This browsed document can contain ambiguity or not. Once the user browsed NL SRS document into the system, the user clicks on check ambiguity button. Then, the system generates the result by syntax ambiguity as red, syntactic as blue and both syntax and both syntax and syntactic ambiguity in one sentence as yellow. Also, a percentage of detected ambiguity will be displayed as a pie chart in this phase.

This pie chart displays the graphical representation of the processed text such detected syntax and syntactic ambiguity in NL SRS document and non-ambiguous requirements. It contains four sections: none ambiguity section, syntax ambiguity section, syntactic ambiguity section and both syntax and syntactic ambiguity section. Each section represents the percentage of the processed data. Fig. 3 shows an example of the results with ambiguities highlighted in different colors and Fig. 2 display the charts.

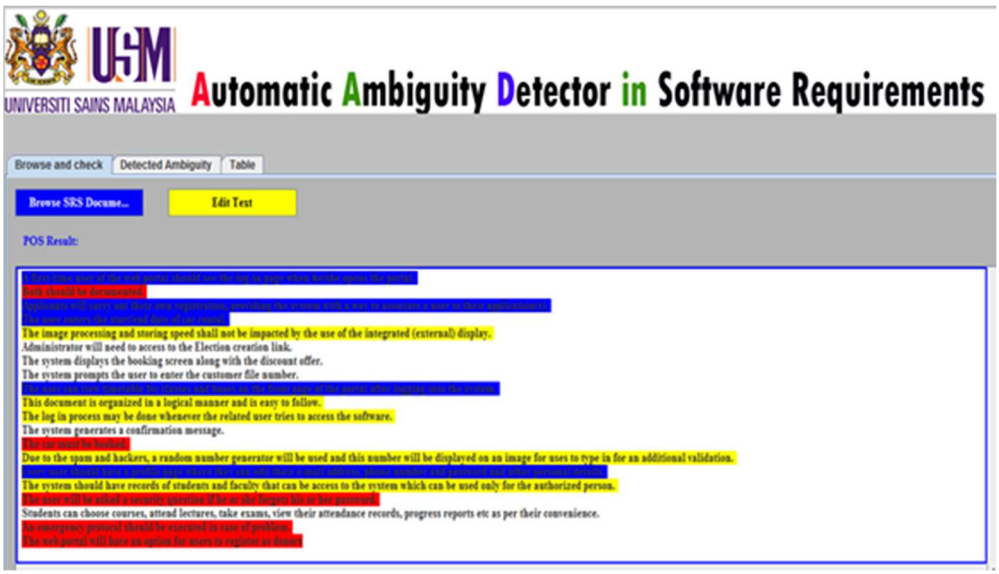

Fig. 3. Example of the results with the Ambiguities Highlighted

6. RESULTS AND DISCUSSION

In Automatic Detection Experiment the results of the tested tool show that the tool able to detect syntax ambiguity and syntactic ambiguity accordingly. It classified the tested data into syntax ambiguity, syntactic ambiguity, both syntax and syntactic ambiguity and none ambiguity. This classification is done by highlighting the processed SRS document in the tool. It can be observed that the tool has generated a proper statistical pie chart which divided the analyzed SRS document results into 4 slices. Moreover, the results of the experiment show that the tool can display the percentage of detected ambiguity in the tested data. This feature is important for the analyst to know the percentage of ambiguities in the document. Automatic Detection Experiment result indicates that that the proposed approach is able to detect syntax and syntactic ambiguity automatically. Moreover, this result has confirmed that POS tagging technique can be used to detect syntax and syntactic ambiguity in SRS document. It indicates that the tool managed to detect and identify the overall syntax and syntactic ambiguities in the Dataset sentences.

In Human detection experiment, the participants are requested to classify 20 given sentences into four groups namely: none-ambiguous group, syntax ambiguity group, syntactic ambiguity group and both syntax and syntactic ambiguity group. This result also confirmed that detecting syntactic

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ambiguity and both syntax and syntactic ambiguity groups are harder than detecting none ambiguous requirements and syntax ambiguity groups. It is not easy to know that a sentence can be parsed into more than 1 meaning. It requires a lot of knowledge. That is what made syntactic ambiguity detection harder than syntax ambiguity.

able to mark the ambiguous sentences and none ambiguous sentence with a different highlighting colors. This work gave us the chance to investigate and explore the detection of ambiguous software requirements in detail. The overall work has been completed successfully.

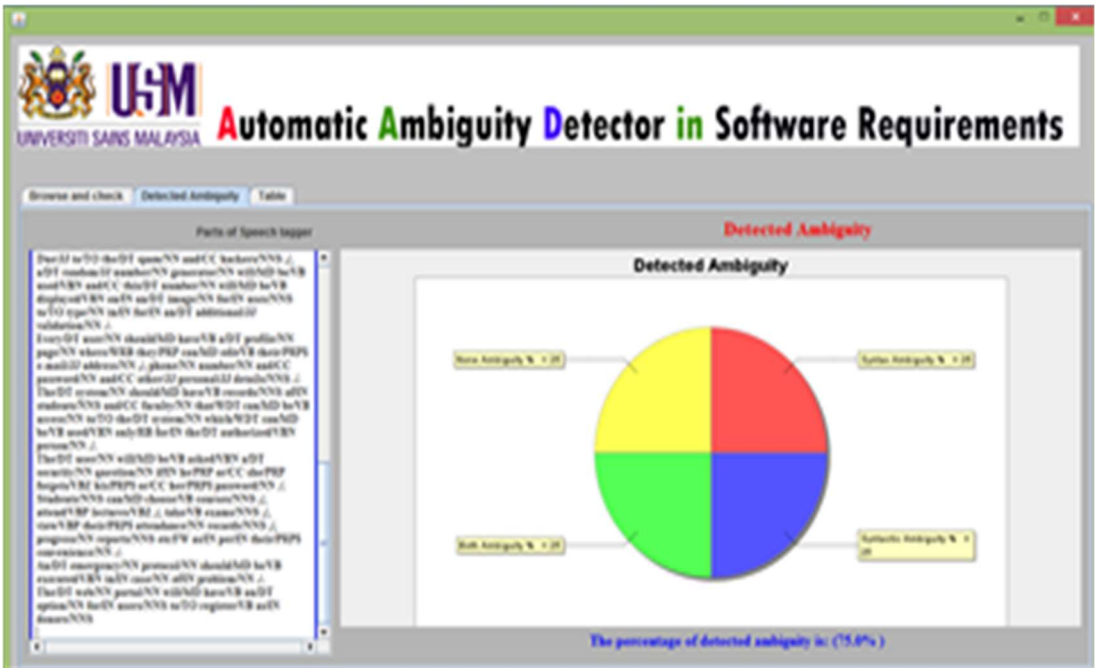

Fig. 4. Ambiguity detector chart

Moreover, Bot syntax and syntactic ambiguity group detection is not easy because it contains two types of ambiguities.

7 CONCLUSION

If the software requirements specified in SRS document are not properly understood by the software developers, the outcome will be ambiguous. Ambiguity in SRS document can lead to a project failure, highly maintenance costs, delayed product releases or developing software which is not fit to the user requirements. In order to solve this problem, a NLP technique called parts of speech tagging have been proposed to automatically detect syntax ambiguity and syntactic ambiguity. A prototype tool called Ambiguity detector in software requirements has been developed to check the effectiveness of the proposed approach. The tool

As currently implemented, the tool has several limitations; these limitations can be further improved in the future work.

1. Now, the automatic ambiguity detection is carried out only using txt format. The wok can be further improved to allow the other standard document formats such as DOC format and PDF format.
2. The ambiguity detector tool can detect two types of ambiguity only. It does not take into account the other types of ambiguity such as lexical ambiguity, scope ambiguity and programmatic ambiguity. In the future, these fragments can be added to the tool.
3. Currently, the tool does not support to save the processed document. it just displays the detected ambiguity without offering any saving option. In the future, this should be considered

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

to improve the quality of ambiguity detector tool.

The results and achievements of this research hopefully will help the software analysts to improve ambiguity detection in their SRS document.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Higher Education of Malaysia, under the Fundamental Research Grant Scheme (FRGS: 203/PKOMP/6711533).

REFRENCES:

[1]. Anuar, U., S. Ahmad, and N.A. Emran. A Simplified Systematic Literature Review: Improving Software Requirements Specification Quality With Boilerplates. 9th Malaysian Software Engineering Conference (MySEC), 2015. 2015. IEEE.

[2]. Soares, H.A. and R.S. Moura. A methodology to guide writing Software Requirements Specification document. in Computing Conference (CLEI), 2015 Latin American. 2015. IEEE.

[3]. Wanless, D., Securing our future health: taking a long-term view. 2002, HM Treasury London.

[4]. Fockel, M. and J. Holtmann. ReqPat: Efficient documentation of high-quality requirements using controlled natural language. in 2015 IEEE 23rd International Requirements Engineering Conference (RE). 2015. IEEE.

[5]. Beg, R., Q. Abbas, and A. Joshi. A method to deal with the type of lexical ambiguity in a software requirement specification document. in Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on. 2008. IEEE.

[6]. Takoshima, A. and M. Aoyama. Assessing the Quality of Software Requirements Specifications for Automotive Software Systems. in Software Engineering Conference (APSEC), 2015 Asia-Pacific. 2015. IEEE.

[7]. Umber, A. and I.S. Bajwa. Minimizing ambiguity in natural language software requirements specification. 2011 Sixth International Conference on Digital Information Management (ICDIM),. 2011. IEEE.

[8]. Nigam, A., Nigam, B., Bhaisare, C., & Arya, N. Classifying The Bugs Using Multi-Class Semi Supervised Support Vector Machine. 2012 International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME),. 2012. IEEE.

[9]. Gill, K.D., Raza, A., Zaidi, A.M., & Kiani, M.M. Semi-Automation For Ambiguity Resolution, 27th Canadian Conference on Open Source Software requirements. in Electrical and Computer Engineering (CCECE). 2014. IEEE.

[10]. Kamsties, E., D.M. Berry, and B. Paech. Detecting ambiguities in requirements documents using inspections. in Proceedings of the first workshop on inspection in software engineering (WISE'01). 2001.

[11]. Sandhu, G. and S. Sikka. State-of-art practices to detect inconsistencies and ambiguities from software requirements. 2015 International Conference on. Computing, Communication & Automation (ICCCA), 2015. IEEE.

[12]. de Bruijn, F. and H.L. Dekkers. Ambiguity in natural language software requirements: A case study. International Working Conference on Requirements Engineering: Foundation for Software Quality. 2010. Springer.

[13]. Gleich, B., O. Creighton, and L. Kof. Ambiguity detection: Towards a tool explaining ambiguity sources. International Working Conference on Requirements Engineering: Foundation for Software Quality. 2010. Springer.

[14]. Nigam, A., Arya, N., Nigam, B., & Jain, D., Tool for automatic discovery of ambiguity in requirements. International Journal of Computer Science Vol. 9, 5,.

[15]. Bano, M. Addressing the challenges of requirements ambiguity: A review of empirical literature. in 2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE). 2015. IEEE.

[16]. Gulia, S. and T. Choudhury. An efficient automated design to generate UML diagram from Natural Language Specifications. 2016 6th International Conference Cloud System and Big Data Engineering (Confluence),. 2016. IEEE.

[17]. Arendse, B., A thorough comparison of NLP tools for requirements quality improvement. 2016.

[18]. Popescu, D., Rugaber, S., Medvidovic N., & Berry, D.M. Reducing ambiguities in requirements specifications via automatically created object-oriented models. LNCS 5320,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2007. Springer.

[19]. Femmer, H., Fernandez, D.M., & Juergens, E., Rapid requirements checks with requirements smells: two case studies. in Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. 2014. ACM.

[20]. Shah, U.S. and D.C. Jinwala, Resolving ambiguities in natural language software requirements: a comprehensive survey. ACM SIGSOFT Software Engineering Notes, 2015. 40(5): p. 1-7.

[21]. Korner, S.J. and T. Brumm. Resi-a natural language specification improver. in Semantic Computing, 2009. ICSC'09. IEEE International Conference on. 2009. IEEE.

[22]. Umber, A., I.S. Bajwa, and M.A. Naeem. NL-based automated software requirements elicitation and specification. in International Conference on Advances in Computing and Communications. 2011. Springer.

[23]. Bajwa, I., M. Lee, and B. Bordbar, Resolving syntactic ambiguities in natural language specification of constraints. Computational Linguistics and Intelligent Text Processing, 2012: p. 178-187.