Algorithm for Automatic Detection of Ambiguities from Software Requirements

Ashima Rani, Gaurav Aggarwal

Abstract: Software Systems are built by the Software engineers and must ensure that software requirement document (SRS) should be specific. Natural Language is the main representation of Software requirement specification document, because it is the most flexible and easiest way for clients or customers to express their software requirements [2]. However being stated in natural language, software requirement specification document may lead to ambiguities [28]. The main goal of presented work to automatically detection of the different types of ambiguities like Lexical, Syntactic and Pragmatic. Then an algorithm is proposed to early detection the different types of ambiguities from software requirement document. Part of Speech (POS) technique and regular expression is used to detect each type of ambiguities. An algorithm presented in this paper have two main goals (1) Automatic detection of different types of ambiguities. (2) Count the total number of each types of ambiguities found and evaluate the percentage of ambiguous and non-ambiguous statements detected from software requirement document. The suggested algorithm can absolutely support the analyst in identifying different kinds of ambiguities in Software requirements specification (SRS) document.

Index Terms: Lexical ambiguity, Syntax Ambiguity, Software Requirement Specification, Syntactic ambiguity, Pragmatic ambiguity, Part of Speech Tagging.

I. INTRODUCTION

Software Requirement Specification (SRS) document is the base document with a full description about functional or non-functional requirements for software development and helps the developer to understand the customer requirements [26]. Majority of the requirement document (87.7%) are in natural language [5]. Due to incomplete or frequently changing requirements submitted by customer’s side, Software requirement document can be inappropriate and ambiguous which affects the software system quality. Ambiguity is the main problem that occurred in the natural language [23].

The main purpose of this paper is to detect the different types of ambiguities from software requirement document. In this paper we have designed the algorithm which is different from other work where it focuses on the different types of ambiguities like lexical, syntactic, syntax and pragmatic that can be detected by regular expressions and POS Tagger technique.

Outline: The remnants of this paper are systematized as follow: Section 1 initiate the field of work. Section 2 defines suggested system (Architecture of Ambiguities Detection Algorithm). Sub Section 2.1 explains the Part of Speech (POS) Tagging and Regular expressions. Sub Section 2.2 defines the different kinds of ambiguities. Section 3 presents the summary of related work. Section 4 explains the algorithm for ambiguity detection and presentation. Section 5 explains the evaluation of proposed algorithm using example. Section 6 presents the conclusion and future work.

II. SUGGESTED SYSTEM

Figure 1 explains the system architecture of the ambiguity detector (suggested system). The main components of the ambiguity detector architecture are: SRS (Software Requirement Specification) Document. POS (Part of Speech) Tagger. Regular Expressions. Algorithm for detection of ambiguous sentences.

Figure 1: Ambiguity Detection Architecture

A. Part Of Speech Tagger

Part of Speech (POS) Tagger is the computational Linguistics Techniques which is used to mark each term of sentence of SRS document with pre-defined POS tags [27]. For Example, the words of the sentence “The System can avoid errors” are marked in following way: The\\DT system\\NN can\\MD avoid\\VB errors\\NNS. Here DT a determiner, NN a noun, MD means modal, VB a verb, NNS means proper noun. POS Tagger can deliver basic form of every term. The tags “VBZ”, “VBN”, “VBP”, “VBG”, “VBD”, “MD”, “VB”, “JJ” and “RB” are vital to detect lexical, syntax and syntactically ambiguity in NL SRS(Lexical/Syntax/Syntactic/Pragmatic).

Authorized on June 15, 2019.

Ashima Rani, Research Scholar, Department of Computer Science, Jagannath University, Jhajjar, Haryana, India (Email: ashima.ashugambhir@gmail.com)

Dr. Gaurav Aggarwal, Associate Professor, Department of Computer Science, Jagannath University, Jhajjar, Haryana, India (Email: gaurav.aggarwal@jagannathuniversityncr.ac.in)

Revised Manuscript Received on June 15, 2019.

Published By:
Blue Eyes Intelligence Engineering & Sciences Publication
B. Ambiguity and Its Types

Ambiguity can be defined as a statement which has more than one meaning (interpretation). In this Paper, Ambiguity lists various categories of ambiguities specifically Lexical, Syntactic, Syntax and Pragmatic ambiguity.

a) Lexical Ambiguity: Lexical ambiguity occurs when a word has many meanings [21]. For example “Young” means “young of age” or “inexpert”, “Bat” means “flying mammal” or “wooden club”.

b) Syntactic Ambiguity: Syntactic ambiguity occurs when a sentence (sequence of words) has multiple parse trees or more than one grammatical structure. Syntactic ambiguity also known as Structural ambiguity. For example: “I saw the men with the telescope” and “time flies like an arrow”. The syntactic ambiguity generated when the sentence contains vague words (adjective or adverbs).

c) Syntax Ambiguity: Syntax ambiguity occurs when a sentence does not end with the full stop (”.”) or if user is not mentioned in the sentence.

d) Pragmatic Ambiguity: Pragmatic ambiguity mainly focuses on relationship or links between the sentences [15]. For example, “they” in “The trucks shall treat the roads before they freeze” can refer both to the roads and to the trucks.

Table 1 explains the classification of text analysis techniques of listed four types of ambiguities.

<table>
<thead>
<tr>
<th>Ambiguity Type</th>
<th>Analysis Task</th>
<th>Analysis Outcomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexical</td>
<td>classify and authenticate the expressions</td>
<td>Set of expressions used in the manuscript</td>
</tr>
<tr>
<td>Syntactic</td>
<td>Identify the terms, build and authenticate the model</td>
<td>Set of expressions used in the manuscript and a model of the system defined in the text</td>
</tr>
<tr>
<td>Syntax</td>
<td>Construct a syntax depiction of sentences</td>
<td>Syntax depiction of each sentence</td>
</tr>
<tr>
<td>Pragmatic</td>
<td>Build a representation of text, including relationships between sentences</td>
<td>Valid depiction of each sentence, formulae</td>
</tr>
</tbody>
</table>

Table 1: Text Analysis Techniques Classification

III. RELATED WORK

Many Researchers have already shown the SRS importance and area of SRS for success or failure of software project. They have applied different techniques and methods to resolve the different types of ambiguities from software requirements. In this section, we study the different approaches for the same.

The work of [6] can be example of solving ambiguity using NLP based technique. In this paper, they have created the activity and sequence diagram using NLP standard POS tagger and parsing technique. By this technique, it reduces the ambiguities in NLSRS document. The main drawback of this technique to absence of automatically prominence the ambiguous statements in NLSRS.

The work of [7] can be another example of POS tagging technique for ambiguity detection. In this method, it detects only the lexical ambiguity and matches the words of one line in NLSRS by the part of speech tagger. The main drawback of this method is that it can work properly if the NLSRS document does not contain more than six words.

The authors [8] developed a tool that detect the ambiguity in NLSRS and mentioned the ambiguity sources. In this paper, researcher used the Part of speech (POS) tagger and regular expression to identify the uncertainties. The main disadvantage of this research is to lack of computing the percentage of ambiguous and non-ambiguous statements in NLSRS.

The researcher [9] explains the different natural language Processing (NLP) tools used for finding ambiguity, uncertainty and quality of use cases.

The work of [4] be example of detection of syntactic, lexical and syntax ambiguities from software requirement specification document. In this paper, they used the POS tagger and corpus. They match the each entry of words of sentence with the POS tagger and identify the syntax, Lexical and syntactic ambiguity. The limitation of this paper is to only detect three types of ambiguity not cover all types.

The authors [5] developed a tool to detect the syntactic and syntax ambiguity using POS tagger. In this method, they implement the algorithm to detect the ambiguities and also compute the percentage of ambiguous and unambiguous statements from software requirement document. The limitation of this paper is only focus to syntactic and syntax ambiguities.

The researchers [3] can be another example to detect the different types of ambiguities at early stages of SDLC. In this research, they designed a tool to detect the lexical, syntactic and pragmatic ambiguity using POS tagger and regular expressions.

IV. ALGORITHM

In this Section, we have proposed an algorithm to detect different types of ambiguities which described in section 2.2. This algorithm consists of 7 steps to detect ambiguities using Regular expressions matching and POS tagging [24],[ 25].

Step 1: Read the NL SRS document (that is to be tested) line by line:

NL SRS document is group of paragraphs [17]. In this step, the paragraph is divided into gradient of sentences and stores it in data organization “splitted_sentence”. Then the total numbers of verdicts are counted from “splitted_sentence” file and stored it in a data organization called “sent_count”.

Retrieval Number: I1410789S19/19©BEIESP
DOI: 10.35940/ijitee.I1410789S19
Published By:
Blue Eyes Intelligence Engineering & Sciences Publication

879
Step 2: Match every entry of “splitted_sentence” through the POS tagger:

Tagged sentences kept in a data organization called or named “tag_sentence”. In this step, mark each entry of “splitted_sentence” with the entry of “tag_sentence” which will be used to classify the sentences into different types of ambiguities (Lexical, syntactic and syntax).

Step 3: Detect Lexical, syntactic and Syntax ambiguity and kept in a data organization called “Lexical”, “Syntactic” and “Syntax” respectively.

(a) Checking if some condemnation in the “tags_sentence” having words like they, include, minimum, easy etc. it will be measured as lexical ambiguity.
(b) Checking if some condemnation in the “tag_sentence” does not contain full stop (“./.” tag) at the end or is passive voice. It will be measured as syntax ambiguity.
(c) Checking if some condemnation in the “tag_sentence” is an adjective or adverb (adjective tag “JJ” and adverb tag “RB”). It will be considered as syntactic ambiguity.

Step 4: Identify pragmatic ambiguity and stored in data organization called “Pragmatic”:

Firstly build a representation of text, include links between sentences from “splitted_sentence”. Checks whether the analyzed line matches certain regular expression. If matches then stored in a data structure “pragmatic”.

Step 5: Continue step 3 and 4 for every line of NLSRS till the completion of NLSRS document.

Step 6: Compute the aggregate number of all types of ambiguities detected by using formulas:

$$\text{Lexical Ambiguity} = \frac{\sum_{\text{Scount} = 1}^{\text{Scount}} \text{Lexical}}{\text{Scount}}$$

$$\text{Syntactic Ambiguity} = \frac{\sum_{\text{Scount} = 1}^{\text{Scount}} \text{Syntactic}}{\text{Scount}}$$

$$\text{Syntax Ambiguity} = \frac{\sum_{\text{Scount} = 1}^{\text{Scount}} \text{Syntax}}{\text{Scount}}$$

$$\text{Pragmatic Ambiguity} = \frac{\sum_{\text{Scount} = 1}^{\text{Scount}} \text{Pragmatic}}{\text{Scount}}$$

Where Scount = sent_count

Step 7: Compute the percentages of ambiguous and unambiguous sentences detected by using formula:

Percentage of lexical ambiguity detected

$$= \left(\frac{\text{Lexical Ambiguity}}{\text{Scount}}\right) \times 100$$

Percentage of Syntactic ambiguity detected

$$= \left(\frac{\text{Syntactic Ambiguity}}{\text{Scount}}\right) \times 100$$

Percentage of Syntax ambiguity detected

$$= \left(\frac{\text{Syntax Ambiguity}}{\text{Scount}}\right) \times 100$$

Percentage of Pragmatic ambiguity detected

$$= \left(\frac{\text{Pragmatic Ambiguity}}{\text{Scount}}\right) \times 100$$

Percentage of Non-ambiguous sentences detected

$$= 100 - \left(\frac{\text{Lexical Ambiguity} + \text{Syntactic Ambiguity} + \text{Syntax Ambiguity} + \text{Pragmatic Ambiguity}}{\text{Scount}}\right)$$

V. EVALUATION

In this section, we will execute the algorithm which we have discussed in section 4. Following are some ambiguos statements which are taken from sample SRS:

- Tickets are resubale within a limited time span.
- System works until deadline.
- The software must be easy as possible.
- The system provides minimum output.
- The system should avoid errors normally.
- Both should be documented.
- The trucks shall treat the roads before they freeze

For detection of four different categories of ambiguities (like lexical, syntactic, syntactic and pragmatic), each sentence are marked with the Part of speech tagging and regular expression as well.

- Tickets are resubale within a limited time span.
- System works until deadline.
- The software must be easy as possible.
- The system provides minimum output.
- The system should avoid errors normally.
- Both should be documented.
- The trucks shall treat the roads before they freeze

If any sentence having words like they, include, easy, etc. It will be considered as lexical ambiguity and denoted in red color. For example in line 2, word “until” does not specify the particular time. So “until” word comes under lexical ambiguity.

If any sentence having the adjectives or adverbs. In the example, words like “resubale”, “possible”, “normally”, “minimum” are vague words(having more than one meaning) and reported as syntactic ambiguity. These are denoted in green color.

If any sentences having the links or relationship between sentences. It will be considered as pragmatic ambiguity. For example, in line 6 sentence marked as syntax ambiguity due to word “both” and denoted in blue color.

If any sentences having some missing information. It will considered as Syntax ambiguity. For example, in line 6 sentence marked as syntax ambiguity due to word “both” and denoted in blue color.

Calculating the percentage of ambiguities in above example:

$$\text{Scount} = 7$$ (Total number of statements)
Algorithm for Automatic Detection of Ambiguities from Software Requirements

Lexical Ambiguity = \( \left( \frac{\text{Lexical Ambiguity}}{\text{Scount}} \right) \times 100 = \frac{1}{7} \times 100 = 14.28\% \)

Syntactic Ambiguity = \( \left( \frac{\text{Syntactic Ambiguity}}{\text{Scount}} \right) \times 100 = \frac{4}{7} \times 100 = 57.14\% \)

Syntax Ambiguity = \( \left( \frac{\text{Syntax Ambiguity}}{\text{Scount}} \right) \times 100 = \frac{1}{7} \times 100 = 14.28\% \)

Pragmatic Ambiguity = \( \left( \frac{\text{Pragmatic Ambiguity}}{\text{Scount}} \right) \times 100 = \frac{1}{7} \times 100 = 14.28\% \)

Table 2 displays the calculation of ambiguities stated in example.

<table>
<thead>
<tr>
<th>% of ambiguity</th>
<th>Lexical Ambiguity</th>
<th>Syntactic Ambiguity</th>
<th>Syntax Ambiguity</th>
<th>Pragmatic Ambiguity</th>
</tr>
</thead>
<tbody>
<tr>
<td>14%</td>
<td>14%</td>
<td>14%</td>
<td>14%</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2 displays the calculation of ambiguities detected in above mentioned example in practice of pie chart. It explains that extreme percentage of existing ambiguities (57%) are syntactic, however 14% Lexical, 14% Syntactic and 14% Syntax ambiguities are detected in above example.

VI. CONCLUSION & FUTURE SCOPE

Requirement gathering is the most significant stage of software development life cycle. Software project mainly depends upon the first step of SDLC i.e. SRS(Software Requirement Specification) document. If SRS is not properly defined, then software project does not fulfill the users requirements. Ambiguity is the main thing to resolve the problems from Software requirement document. In this paper, we suggested a algorithm to detect the different types of ambiguities like lexical, syntactic, syntax and pragmatic from NLSRS at the early stage. This algorithm also determines the type of ambiguities and percentage of different types of ambiguities using the formula. It also calculates the number of ambiguous and non ambiguous statements from Software requirement specification(SRS) document. The main purpose of algorithm is to detect main four types of ambiguities. In future work, we developed a automatic ambiguity detection tool with the implementation of proposed algorithm which highlights the ambiguous statements with different colors according to type of ambiguity.

REFERENCES

language. in 2015 IEEE 23rd International Requirements Engineering Conference (RE), 2015. IEEE.

AUTHORS PROFILE

Ashima Rani is Research Scholar, Department of Computer Science, Jagannath University, Jhajhar (Haryana, India). She received her B.Sc. from Kurukshetra University, Kurukshetra in 2007 and MCA from GITM, Gurgaon (Maharishi Dayanand University, Rohtak) in 2010. She has done M.Tech(CSE) from MBU, Solan in 2013. She has been teaching UG and PG classes for well over 9 years. She has 18 Research papers in International Journals and Conferences. She has attended more than 30 workshop/Conferences/FDP/Seminar during her 9 years of experience of teaching. She has to her credit two books with International publisher. Her research interest includes Software Engineering, Algorithm Designs and Database.

Dr. Gaurav Aggarwal is Associate Professor and Head in Faculty Engineering & Technology with the Jagannath University, NCR, Haryana, India. He received his B.Tech in CSE from Maharishi Dayanand University, Rohtak in 2005 and M.Tech from VCE, Rohtak in 2008. He received his Ph.D. Degree from Faculty of Engineering & Technology, JNU, Jaipur in 2016. He has teaching experience of 14 years. He is active member of R. G. Education Society, Rohtak. He has more than 40 Research papers in National and International Journals and Conferences. He has guided 20 Post graduate students and 2 Doctorate student and currently 6 Doctorate students are doing their work under his supervision. His research interest includes Software Engineering, Software Reliability, Neural Networks and Data Mining.