# 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, Syntax 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 2.1explains 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.

**Revised Manuscript Received 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)

## II. SUGGESTED SYSTEM

Figure 1 explains the system architecture of the ambiguity detector (suggested system). The main components of the ambiguity detector architecture are:

1. SRS (Software Requirement Specification) Document.
2. POS (Part of Speech) Tagger.
3. Regular Expressions.
4. Algorithm for detection of ambiguous sentences.
5. 
6. 

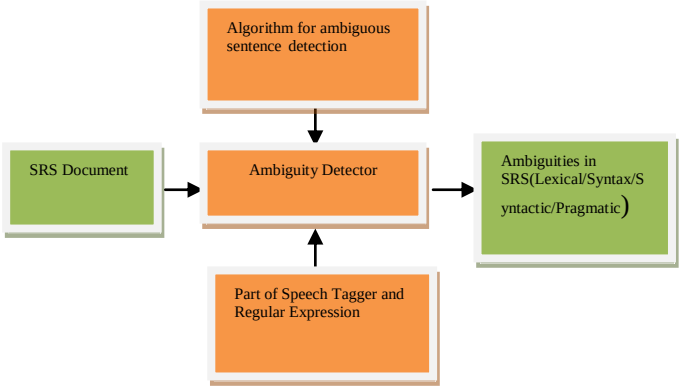


**Figure 1: Ambiguity Detection Architecture**

### 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(Software Requirement) document. Regular expression is basically used to detect pragmatic ambiguity.

### 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 1: Text Analysis Techniques Classification**

| Ambiguity Type | Analysis Task | Analysis Outcomes |
|---|---|---|
| Lexical | classify and authenticate the expressions | Set of expressions used in the manuscript |
| Syntactic | Identify the terms, build and authenticate the model | Set of expressions used in the manuscript and a model of the system defined in the text |
| Syntax | Construct a syntax depiction of sentences | Syntax depiction of each sentence |
| Pragmatic | Build a representation of text, including relationships between sentences | Valid depiction of each sentence, formulae |

## 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".

**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:**

Lexical Ambiguity = $\sum_{Scount=1}^{Scount=-1} Lexical$

Syntactic Ambiguity = $\sum_{Scount=1}^{Scount=-1} Syntactic$

Syntax Ambiguity = $\sum_{Scount=1}^{Scount=-1} Syntax$

Pragmatic Ambiguity = $\sum_{Scount=1}^{Scount=-1} Pragmatic$

Where Scount = sent_count

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

Percentage of lexical ambiguity detected

$= (\frac{Lexical\ Ambiguity}{Scount})*100$

Percentage of Syntactic ambiguity detected

$= (\frac{Syntactic\ Ambiguity}{Scount})*100$

Percentage of Syntax ambiguity detected

$= (\frac{Syntax\ Ambiguity}{Scount})*100$

Percentage of Pragmatic ambiguity detected

$= (\frac{Pragmatic\ Ambiguity}{Scount})*100$

Percentage of Non-ambiguous sentences detected

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

## V. EVALUATION

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

**Tickets are resubale within a limited time span.**
**System works untill 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 ambiguties (like lexical, syntax, syntactic and pragmatic), each sentence are marked with the Part of speech tagging and regular expression as well.

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

If any sentence having words like they, include, easy, until etc. It will be considered as lexical ambiguty and denoted in red color[22]. For example in line 2, word "until" does not specify the perticular time. So "until" word comes under lexical ambiguty.

If any sentence having the adjectives or adverbs [19]. It will considered as Syntactic ambiguty. In the example, words like "resubale", "possible", "norrmally", "minimum" are vague words(having more than one meaning) and reported as syntactic ambiguty. These are denoted in green color.

If any sentence having some missing information. It will considered as Syntax ambiguty[29]. For example, in line 6 sentence marked as syntax ambiguty due to word "both" and denoted in blue color.

If any sentences having the links or relationship between sentences. It will be considered as pragmatic ambiguty. For example, in line 7 word "they" can refer to both trucks and roads. It is denoted in purple color.

Calculating the percentage of ambiguities in above example:

Scount=7(Total number of statements)

Lexical Ambiguty = ( $\frac{Lexical\ Ambiguity}{Scount}$ )*100 = (1/7)*100=14.28%

Syntactic Ambiguity = ( $\frac{Syntactic\ Ambiguity}{Scount}$ )*100 = (4/7)*100= 57.14%

Syntax Ambiguity = ( $\frac{Syntax\ Ambiguity}{Scount}$ )*100 = (1/7)*100=14.28%

Pragmatic Ambiguity = ( $\frac{Pragmatic\ Ambiguity}{Scount}$ )*100= (1/7)*100 = 14.28%

Table 2 displays the calculation of ambiguities stated in example.

**Table 2: Percentage of ambiguities calculated in example**

| | **Lexical Ambiguity** | **Syntactic Ambiguity** | **Syntax Ambiguity** | **Pragmatic Ambiguity** |
|---|---|---|---|---|
| % of ambiguity | 14% | 57% | 14% | 14% |

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.

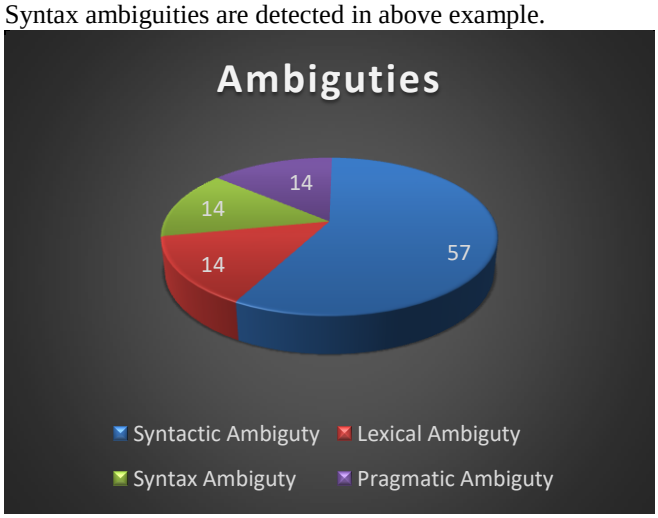


**Figure 2: Result of Example using Pie Chart**

## 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. Ambiguty 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 ambiguties like lexical, syntactic, syntax and pragamtic from NLSRS at the early stage. This algorithm also determines the type of amiguties and percentage of different types of amiguties using the formula. It also calculates the number of ambigous and non ambigous statements from Software requirement specification(SRS) document. The main purpose of algorithm is to detect main four types of ambiguties. In future work, we developed a automatic ambiguty detection tool with the implemention of proposed algorithm which highlights the ambigous statements with different colors according to type of ambiguty.

## REFERENCES


1. Ashima Gambhir, “Comparative Analysis of Ambiguities Resolving Tools in Natural language Software Requirements”, Journal of Emerging Technologies and Innovative research, Vol 6, Issue 5, May 2019, pp 133-137.
2. Ashima Rani, Gaurav Aggarwal, “Advanced Practices to Detect Ambiguities and Inconsistencies from Software Requirements”, IEEE, 7th international conference on System Modeling & Advancement in Research Trends, November 2018, pp. 17-21.
3. BenediktGleich, Oliver Creighton, Leonid Kof, “Ambiguity Detection: Towards a toll explaining ambiguity sources”, International Working Conference on Requirements Engineering: Foundation for Software Quality, 2010, pp 218-232.
4. Ayan Nigam, Neeraj Arya, Bhawna Nigam, Dipika Jain, “Tools for automatic discovery of ambiguity in requirements”, International Journal of Computer Science Issues, Vol 9, Issue 5, 2012, pp 350-356.
5. ALI OLOW JIM, WAN MOHD NAZMEE WAN ZAINON, “An approach for detecting syntax and syntactic ambiguity in software requirement specification”, Journal of Theoretical and applied Information Technology, Vol 96, Issue. 8, April 2018, pp 2275-2284.
6. Gulia, S. and T. Choudhury, “An efficient automated design to generate UML diagram from Natural Language specifications” , IEEE 6th International Conference Cloud System and Big Data Engineering (Confluence), 2016.
7. Beg, R., Q. Abbas, and A. Joshi, “A method to deal with the type of lexical ambiguity in a software requirement specification document”, ICETET’08 in Emerging Trends in Engineering and Technology, 2008.
8. 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.
9. Arendse, B., “A thorough comparison of NLP tools for requirements quality improvement”, 2016.
10. 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.
11. 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.
12. 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.
13. Popescu, D., Rugaber, S., Medvidovic N., & Berry, D.M. Reducing ambiguities in requirements specifications via automatically created object-oriented models. LNCS 5320, 2007. Springer.
14. 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.
15. 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.
16. Korner, S.J. and T. Brumm. Resi-a natural language specification improver. in Semantic Computing, 2009. ICSC'09. IEEE International Conference on. 2009. IEEE.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. Unnati S Shah, Devesh C. Jinawala, "Resolving Ambiguties in Natural language Software Requirements: A Comp rehensive Survey", ACM SIGSOFT Software Engineering Notes, Vol 40 No. 5, September 2015.
24. Gang Liu, Shaobin Huang, Xiufeng Piao, "Study on Requirement Testing Method Based On Alpha-Beta Cutoff Procedure" Collage of computer Science and Technology, Harbin Engineering University, Harbin, Heilongjiang, China, 2008 IEEE.
25. Ravi Prakash Verma, Bal Gopal, Md. Rizwan Beg, "Algorithm for Generating Test Case for Prerequisites of Software Requirement" Department of Computer Science and Engineering, Integral University. International Journal of Computer Application, September 2010 IEEE.
26. Ronald Kirk Kndt "Software Requirement Engineering: Practice and Techniques", Jet Propulsion Laboratory, California Institute of Technology, November 7, 2003.
27. MassilaKamalrudin, SafiahSidek, Sharifah Sakinah, Syed Ahmad, NadiahDaud, "A review of requirements engineering tools for requirements validation software engineering process",vol.1, no.1, International journal of software engineering and technology (IJSET), 2014.
28. Dr. Sohail Asghar, Mahrukhumar, "Requirement engineering challenges in development of software applications and selection of customer-offthe-Shelf (COTS) components", International journal of software engineering (IJSE), vol. 1, no. 2, August 2010.
29. Khtira, A.; Benlarabi, A.; El Asri, B. Detecting feature duplication in natural language specifications when evolving software product lines. In Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering, Barcelona, Spain, 29–30 April 2015.

## AUTHORS PROFILE

**Ashima Rani**is Research Scholar, Department of Computer Science, Jagannath University Jhajjar (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.