Tool for Automatic Discovery of Ambiguity in Requirements

Ayan Nigam[1], Neeraj Arya[2], Bhawna Nigam[3] and Deepika Jain[4]

[1] Quality and Process Lead, Ideavate Solutions
Indore, Madhya Pradesh 452001, India

[2] Institute of Engineering and Technology, Devi Ahilya Vishwavidhyalaya
Indore, Madhya Pradesh 452017, India

[3] Institute of Engineering and Technology, Devi Ahilya Vishwavidhyalaya
Indore, Madhya Pradesh 452017, India

[4] Institute of Engineering and Technology, Devi Ahilya Vishwavidhyalaya
Indore, Madhya Pradesh 452017, India

Abstract

Requirements are the foundation for delivering quality software. Often it is found that the short development cycle lead teams to cut short the time they will spend on Requirement Analysis. In this work we developed a tool which can quickly review requirements by identifying ambiguous words and provide us the possible sources of wrong interpretation. Currently tool supports identification of Lexical, Syntactic and Syntax ambiguities. The tool will assist requirement analysis personnel while reviewing specifications, highlighting ambiguous words and providing graphical snapshot to gauge the correctness of documents.

Keywords: *Software Requirements Specification, Ambiguity, Requirement Engineering, Lexical Ambiguity, Syntactic Ambiguity, Syntax Ambiguity.*

1. Introduction

One of the important phases of Software Development process is Requirement gathering. Requirements (functional as well as non functional) are managed in a document called as Software Requirement Specification (SRS), which is referred by development team to understand requirements. If there is short development cycle of project, then team members don't spend more time on Requirement Analysis. Hence the outcome is an improper SRS document. Another reason for inappropriate SRS document is that, if requirements are frequently changing or incomplete requirements are provided from customer's side, then document designer may use inexact words or statements while preparing the SRS. When stakeholders refer such document, they can interpret the sentences of SRS in various ways which ultimately results in "Ambiguity" and affects the quality of the system to be built.

Researchers have already shown the importance of SRS and areas of SRS, which are responsible for success or failure of a software project. For e.g. Don Gause [11] lists five most important sources, including SRS, that are responsible for failure of requirements. [24] shows the roles of SRS document in large systems, and its importance in coordinating team of multiple persons to ensure that right system is going to be built. Bertrand Mayer [12] shows areas of SRS document, where document writer is more likely to make mistakes. His study presented a thorough description of such mistakes by classifying them into seven distinct categories named as "seven sins". All these sins deteriorate the quality of an SRS document. Here Ambiguity is presented as one of the sins.

Ambiguity in Requirement Specification causes numerous problems that affect the system to be built, because ambiguity becomes a bug if not found and resolved at early stages. Common types of bugs are Design Bug, Functional Bug, Logical Bug, Performance Bug, Requirement Bug and UI Bug [16]. If these bugs or other types of bugs are not found until testing, then they are approximately fourteen times costlier to get fixed [3]. For example, in early 1970's, software for payroll system was designed that uses last two digits for representing a year rather than 4 digits so as to save memory space. But in Year 2000, Y2K bug arose, that threatened the major industries. Hundreds

of dollars were spent to upgrade this failure [15]. If such types of bugs are detected in early phase of development, then it would be easier to fix it. Fig. 1 shows that 56% of bugs were identified in requirement analysis phase. Analysis [3] shows that if these issues are not settled at early stage then cost and development time will be affected. [4] Shows that the cost of repairing a requirement error during other phases could cost 10 to 20 times more than that of repairing the error during requirement and early design phases. Table 1 shows that relative cost to fix an error is comparatively less in requirement phase [3].

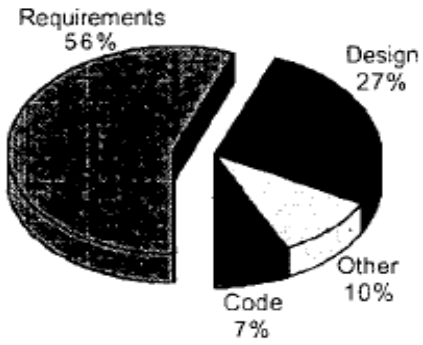

Fig 1: Distribution of bugs in different phases of development cycle [3]

Therefore testing of requirements is very important task in Software Engineering. Requirement Testing means verification and validation of software requirements [1]. The basic objective of verification and validation of software requirement is to identify and resolve the software problems and high-risk issues early in the software life cycle [2].

Phase in which error found	**Cost Ratio**
Requirements	1
Design	3-6
Coding	10
Unit/ Integration Testing	15-40
System/ Acceptance Testing	30-70
Production	40-1000

Table 1: Relative cost to fix an error [3]

For documentation of software requirement, Software Engineering Standard Committee of IEEE computer society presents a guideline using IEEE 830:1998 format [5]. [6] Proposed alpha-beta procedure to cut off the branches of requirement tree and reduce the complexity of tree traversal. Antonio Bertilino discusses different types of challenges and achievements in software testing [7]. [8] Develops an algorithm to generate test cases that verify the requirement of developing a GUI.

The main objective of this paper is to describe a tool named "Ambiguity Detector" that will assist in finding the words or sentences, responsible for three types of ambiguities i.e. Lexical ambiguity, Syntactic ambiguity and Syntax ambiguity. For this purpose, Parts of Speech Tagger and Corpus of ambiguous words is used.

2. Architecture of Ambiguity Detector

The architecture of Ambiguity Detector is shown in fig. 2. This tool contains four main components i.e. SRS document, Algorithm for detecting Ambiguous Sentences, Corpus of different ambiguous words and Parts of Speech Tagger which are explained as below:

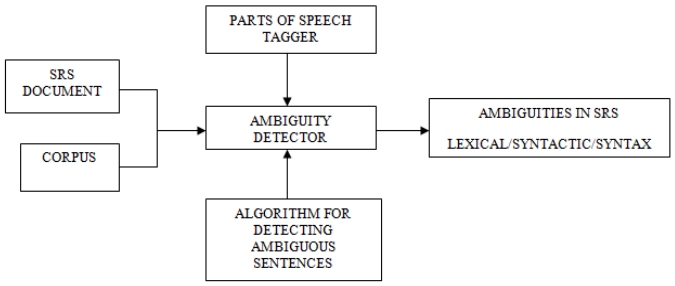

Fig 2: Architecture of Ambiguity Detector Tool

2.1 SRS Document

The goal of requirement specification is to create a SRS document, describing what system is to be built. SRS captures the results of problem analysis and characterizes the set of acceptable solutions for the problem [24]. SRS can play many roles:-

2.1.1 The SRS is primary vehicle for agreement between the developer and customer on exactly what is to be built. It is a document reviewed by the customer or his representative and often is the basis for judging fulfillment of contractual obligations.
2.1.2 The SRS records the result of problem analysis. Documenting the result of analysis allows question about the problem to be answered only once during development.
2.1.3 The SRS defines what properties the system must have and constraints on its design and implementation. It helps in ensuring that requirement decision is made explicitly during the requirement phase not implicitly during programming.

2.1.4 The SRS is basis for estimating cost and schedule. It is a primary management tool for tracking development progress and ascertaining what remains to be done.
2.1.5 The SRS is basis for test plan development. It is used like a testers tool for determining the acceptable behavior of software.
2.1.6 The SRS provide the standard definition of expected behavior for the system maintainers and is used to record engineering changes.

2.2 Corpus

Corpus is the main component of ambiguity detector. Ambiguous words that result in misinterpreted requirements are analyzed and stored into the corpus. The major concern of this tool is to check and validate whether the data which is a part of SRS document is ambiguous or not. So SRS is matched with the vague words that are stored in corpus [9] [10] [15]. Some of the ambiguous words are introduced here:-

2.2.1 Always, Every, None, Never: This word denotes something as certain or absolute, make sure that it is indeed, certain, find out these words and think of cases that violate them.
2.2.2 Certainly, Clearly, Therefore, Obviously: These words tend to persuade accepting something as given.
2.2.3 Good, Fast, Small, Cheap, Stable and Efficient: These are unquantifiable. If they appear in a specification, they must be further defined to explain exactly what they want.
2.2.4 Some, Sometime, often, usually, Ordinarily, Customarily, Most, Mostly: These words are too vague. It's impossible to test a feature that operates sometime.
2.2.5 Handled, Processed, Rejected, Skipped, Eliminated: These terms can hide large amounts of functionality that need to be specified.
2.2.6 And So Forth, And So On, Such As: Lists that finish with words such as these aren't testable. If they appear in a specification, they must further be defined to explain so that there's no confusion as to how the series is generated and what appears next in the list.
2.2.7 It, They, That, Those: These words contain vague subjects that can refer to multiple things.

Table 2 shows some other ambiguous words.

Accommodate	Capability of	Normal
Adequate	Capability to	Not limited to
And	Easy	Provide for
As a minimum	Effective	Robust
As applicable	Etc.	Sufficient
As appropriate	If practical	Support
Be able to	Maximize	These
Be capable of	May	This
Can	Minimize	When necessary

Table 2: Words/Phrases that result in misinterpretation

2.3 Parts of Speech Tagger

Part-of-speech (POS) Tagger is very important component of Ambiguity Detector. POS Tagger tags every word of a sentence with one of the predefined parts-of-speech. For example, the words of the sentence "Failure of any other physical unit puts the program into degraded mode" are marked in the following way: Failure/NN of/IN any/DT other/JJ/ physical/JJ unit/NN puts/VBZ the/DT program/NN IN/into degraded/VBN mode/NN. Here, NN means a noun, DT a determiner, JJ an adjective, VBZ a verb, and IN a preposition. With the help of tagger tool, ambiguity i.e. lexical/syntactic/syntax ambiguity is detected.

2.4 Ambiguities in SRS

Ambiguity is the possibility to interpret a phrase/word in several ways. It is one of the problems that occur in natural language texts. An empirical study by Kamsties et al [12] depicts that "Ambiguities are misinterpreted more often than other types of defects. Ambiguities, if noticed, require immediate clarification". The Ambiguity Handbook [14] lists several types of ambiguities, namely lexical, syntactic, syntax and semantic ambiguities. This tool detects first three types of ambiguities, which are explained below-

Lexical ambiguity: - Lexical ambiguity occurs when a word has several meanings. For example "green" means "of color green" or "immature". Lexical ambiguity also occurs when two words of different origin come to the same spelling and pronunciation. For example "bank" means "river bank" or "bench".

Syntactic ambiguity: - Syntactic ambiguity, also called structural ambiguity, occurs when a given sequence of word can be given more than one grammatical structure, and each has different meaning. For example when the sentence allow different parse trees, like "Small car factory" that can mean both "(small car) factory" and "small (car factory)".

Syntax Ambiguity: - This ambiguity is particular to the tool developed. This error occurs if a sentence does not end with a period (.), second if user agent is not specified in the sentence, then it is regarded as syntax error.

2.5 Algorithm for Ambiguity Detection

Ambiguity Detector works on following algorithm. This algorithm is used to classify the ambiguities as Lexical, Syntactic or Syntax ambiguity. The steps of algorithm are as follows:-

Step-1: Read corpus of ambiguous words from a text file, and store it in data structure named as 'i'.

Step-2: Read the SRS document (that is to be tested) line by line.

Step-3: For each line, match all words against the corpus. If word/words are matched then store the sentence in another data structure named as 'j'. Continue this step for each line of SRS, till the end of SRS document is reached.

Step-4: Match each entry of j with POS Tagger, which classifies the sentences into Lexical, Syntactic or Syntax ambiguities, depending upon the types of ambiguous words/phrases.

Step-5: Count and store the total number of lexical, syntactic and syntax ambiguities.

Step-6: Calculate the percentage of ambiguities.

2.6 Ambiguity Detector

Ambiguity detector tool is designed to find ambiguities in SRS document. Fig 3 shows the interface of Ambiguity detector tool.

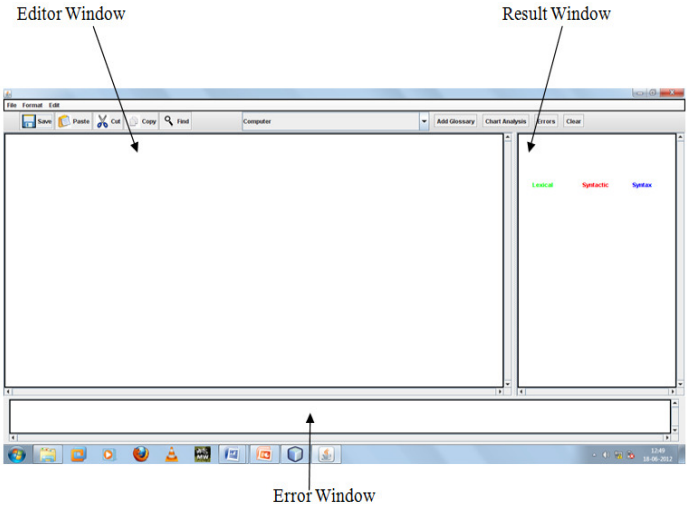

Fig 3: Interface of Ambiguity Detector

Ambiguity detector mainly contains three windows - first is "Editor Window" where SRS is selected and processed line by line for ambiguity testing. In this window, different colors are used to highlight the ambiguities in selected SRS document (green for lexical ambiguity, red for syntactic ambiguity and blue for syntax ambiguity). Apart from SRS documents, sentences can directly be written for testing ambiguities. Second window is "Error Window". This window specifies the ambiguities in the statements that are explicitly written by users where line number and related ambiguity of complete SRS is displayed in this window.

Fig 4 shows the ambiguity of sample sentence written in Editor Window. The sentence is "This system must be reusable". The ambiguous word is "reusable" which is highlighted in Editor Window, whose description (Ambiguous Adjective) is displayed in error window along with line number (line no. 1).

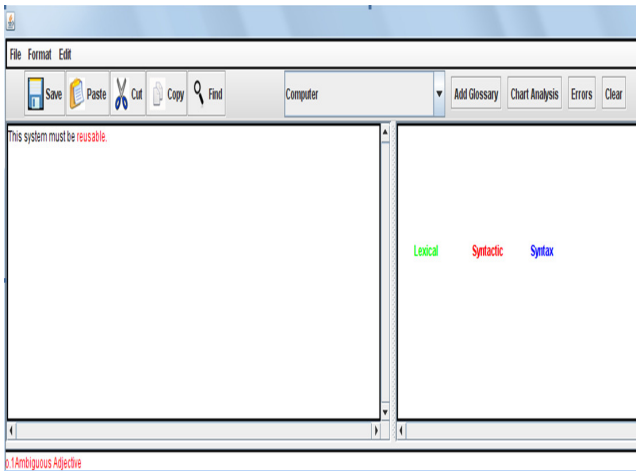

Fig 4: Snapshot of Sample statement and output ambiguity in Error Window

For SRS document, different error window is designed, which will be displayed after clicking the error button. Third window is the "Result Window", which shows the total ambiguities and count of individual ambiguities in the SRS document using bar graph. Same color schemes (green, red and blue) are used for graphical representation of ambiguities.

An option is also given in the tool for Chart Analysis in which proportion of different ambiguities are shown in the form of pie chart in a separate window.

Example: Six ambiguous sentences (in bold) are taken from a sample SRS, which are already matched against corpus.

The System shall be easy as possible.
Both should be documented.
It must be reusable.
The system should avoid errors normally.
The system provides maximum output.
System works until deadline.

For finding ambiguities, all sentences are tagged one by one according to parts of speech, using Parts of speech

Tagger. After testing these six sentences, Ambiguity detector gives following types of results -

1. The system shall be easy as possible.
2. Both should be documented.
3. It must be reusable.
4. The system should avoid errors normally.
5. The system provides maximum output.
6. System works until deadline.

Lexical ambiguity (in green color) arises due to some unidentified references. For example in sixth line the word "until" has been reported as lexical ambiguity because "until" does not specify a particular time.
Syntactic ambiguity (in red color) arises due to use of vague words. Adjectives and adverbs are considered as vague words, because the words are unclear i.e. these words can have different interpretations. So in the example, words like "reusable", "normally" and "maximum" are reported as syntactic ambiguities.
Syntax ambiguity (in blue color) arises due to some missing information. In above example, second line is marked as syntax ambiguity because of the word "both". This sentence does not contain complete information.

Fig. 5 shows the percentage of ambiguities present in the example in form of pie chart. It shows that highest percentage of ambiguities present (67%) are syntactic ambiguities, where as only 16% Lexical and 16% Syntax errors are found in the example.

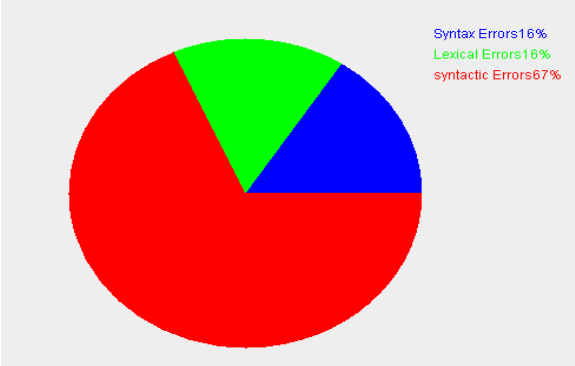

Fig 5: Result of Example in form of Pie Chart
Table 3 shows the percentage of error reported in example.

	Lexical Ambiguity	Syntactic Ambiguity	Syntax Ambiguity
Percentage of error	16	16	67

Table 3: Percentage of ambiguities in Example

3. Experimental Results

In experimental work, four open source SRS documents were taken and analyzed. Number of lines and source of sample SRS documents is presented in Table 4.

SRS	Number of Lines	Source
1.	165	www.scribd.com
2.	245	www.scribd.com
3.	357	www.scribd.com
4.	487	www.scribd.com

Table 4: Description of Datasets

An SRS Document (in .txt format) is selected in the tool for ambiguity testing. Fig 6 shows the snapshot of tool when SRS 2 is selected and tested.

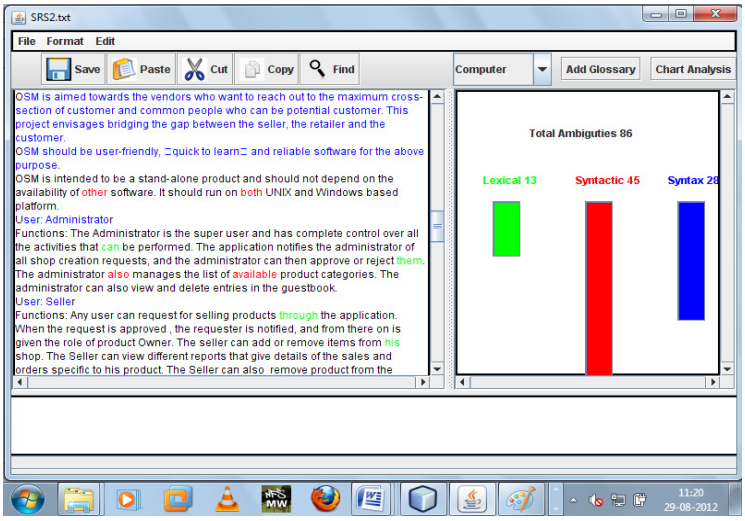

Fig 6: Snapshot of tool after selection of SRS 2

Table 5 shows the results of all the datasets. It shows the total ambiguities present in SRS documents and percentage of lexical, syntactic and syntax ambiguity for each SRS.

SRS	Total Ambiguities	Lexical Ambiguity	Syntactic Ambiguity	Syntax Ambiguity
1	61	21%	53%	26%
2	86	15%	53%	32%
3	60	15%	11%	74%
4	92	23%	35%	42%

Table 5: Results of Different ambiguities for Sample Datasets

Pie Chart representation of ambiguities for SRS 2 is shown in Fig 7.

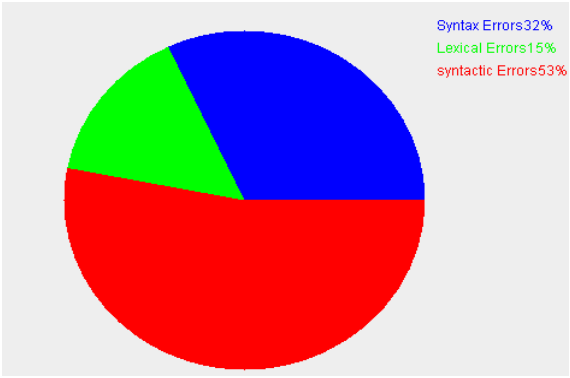

Fig 7: Snapshot for Result of SRS 2 (Chart Analysis)

4. Conclusion and Future Work

One of the most important stages of software development is requirement gathering. Rest of the project depends upon this initial step i.e. how requirements are understood, gathered and specified. If requirements are not properly understood, or SRS is not properly designed, then the outcome will be ambiguous SRS document. Ambiguities in SRS introduces conflicts in the software project, as different interpretations can be drawn by team members while understanding requirements, which ultimately affect the quality of system to be built. One way to solve this problem is to detect and resolve ambiguities early, i.e. in the requirement analysis phase. So a tool named Ambiguity Detector is designed that detects three types of ambiguities in SRS document namely lexical, syntax and syntactic ambiguity. This tool also determines the ambiguities in percentage basis that helps analysts to identify which ambiguity is present in the highest percentage. For e.g. experimental results in Table 5 shows that in all the datasets, percentage of syntactic ambiguity is highest, so provision will be made to improve the SRS document accordingly. As ambiguities can easily be found out and resolved at early stage by communicating again with the customer, therefore this tool is very helpful in saving cost and time.

Till now, the tool is detecting Lexical, Syntactic and Syntax ambiguity. In future work, other types of ambiguities such as Semantic and Pragmatic ambiguity will also be considered. Also, detailed description of word/phrases responsible for ambiguity needs to be provided. And apart from communication with the customer, a suggestion module will be designed in the tool to resolve ambiguities. The suggestion module will recommend a replacement word for the current ambiguous word in order to provide better clarity to the statements of SRS document.

References

[1]. Yonghua Li, Fengdi Shu, Guoqing Wu, Zhengping Liang "A Requirement Engineering for Embedded Real-Time Software-SREE," *Wuhan University Journal of Natural Science,* Vol. 11, No. 3, 2006, pp. 533-538

[2]. Bary W. Boehm, TRW, "Verifying and Validating Software Requirements and Design Specification", January 1984 IEEE.

[3]. Gery Mogyorodi, *Starbase Corporation,* "Requirement-Based Testing: An overview". 2001 IEEE.

[4]. Weider D. Yu"Verifying Software Requirement: A Requirement Tracing Mathodology and It's Software Tool- RADIX", 1994 IEEE.

[5]. Software engineering standard committee of IEEE Computer Society. IEEE Recommended practice for Software Requirement Specification, IEEE Inc. NY, USA, 1998

[6]. Gang Liu, Shaobin Huang, Xiufeng Piao, "Study on Requirement Testing Method Based On Alpha-Beta Cut-off Procedure" *Collage of computer Science and Technology, Harbin Engineering University, Harbin, Heilongjiang, China,* 2008 IEEE.

[7]. Antonio Bertolino, "Software Testing Research: Achievements, Challenges, Dreams" *Institute of Science and Information Technology, Pisa, Italy* .Future of Software Engineering 2007 IEEE.

[8]. 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.

[9]. Donald Firesmith "Specifying Good Requirements", Software Engineering Institute, U.S.A., Journal of Object Technology, Vol.2, No. 4, July-August 2003.

[10]. Ronald Kirk Kndt "Software Requirement Engineering: Practice and Techniques", Jet Propulsion Laboratory, California Institute of Technology, November 7, 2003.

[11]. Gause, D.C., "User DRIVEN Design—The Luxury that has Become a Necessity, A Workshop in Full Life-

Cycle Requirements Management", ICRE 2000 Tutorial T7, Schaumberg, IL (23 June 2000).
[12]. Meyer, B. (1985) "On Formalism in Specifications". *IEEE Software, 2(1)*, January 1985, 6–26.
[13]. Kamsties, E., Knethen, A.V., Philipps, J., Sch¨atz, B.: An empirical investigation of the defect detection capabilities of requirements specification languages. In: Proceedings of the Sixth CAiSE/IFIP8.1 International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design (EMMSAD'01). (2001) 125–136
[14]. Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: Linguistic sources of ambiguity (2003) http://se.uwaterloo.ca/˜dberry/handbook/ambiguityHandbook.pdf, accessed 27.12.2009.
[15].Software Testing by Ron Patton, Sams Publishing, July 26, 2005.
[16].Ayan Nigam, Bhawna Nigam, Chayan Bhaisare, Neeraj Arya : Classifying the Bugs Using Multi-Class Semi Supervised Support Vector Machine, Proceedings of the International Conference on Pattern Recognition, Informatics and Medical Engineering, March 21-23, 2012
[17]. Boehm, B. *Tutorial: Software Risk Management*(1989), IEEE Computer Society Press.
[18].Rupp, C.: Requirements-Engineering und - Management. Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (2004) .
[19]. Sawyer, P., Rayson, P., Cosh, K.: Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans. Softw. Eng. 31 (2005).
[20]. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich (1999).
[21]. Schiller, A., Teufel, S., St¨ockert, C., Thielen, C.: Guidelines f˙ur das Tagging deutscher Textcorpora mit STTS. Technical report, Institut fur maschinelle Sprachverarbeitung, Stuttgart (1999).
[22]. Goldin, L., Berry, D.M.: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Eng. 4 (1997).
[23]. Ronald Kirk Kndt "Software Requirement Engineering: Practice and Techniques", Jet Propulsion Laboratory, California Institute of Technology, November 7, 2003.
[24]. Stuat R. Faulk "Software Requirements: A Tutorial", Software Requirement Engineering 2nd Edition, R. Thayer. M. Dorfman, Eds., IEEE Computer Society press, 1997.