This chapter gives the overview about the basic concepts of Requirement Engineering, Natural Language Processing, and Ambiguities. This chapter helps the readers to be familiarized with the idea of Requirement Engineering, its importance in the development of a software project. It also discusses the phenomena of Natural Language processing and how ambiguities play a vital role in the SRS documentation.

This chapter also gives the overview about the past work and the literature survey done to know the state of the art in the field of disambiguation of documents represented in natural languages. This chapter helps the readers to be familiarized with the idea of methods and techniques proposed by various researchers in the past to resolve the ambiguities present in a document.

2.1 Background Study

2.1.1 Natural Language

Natural language is the most common means of everyday communication among humans for example Hindi, English, & Urdu. They have evolved together with humankind. However, Natural languages are inherently ambiguous. Shannon & Weaver, (C E Shannon, 1949), has developed a communication model (Figure 2.1).

![Figure 2.1: Communication model by Shanon](image-url)
Suppose sender ‘A’ have some idea and he wants to communicate it to Receiver ‘B’. ‘A’ uses Natural Language (English) to express his idea.

Ideally in case of successful communication, ‘B’ interprets exactly what ‘A’ want to communicate, so both ‘A’ & ‘B’ shares the same idea. However in real world situation the communication may fails sometimes due to ambiguities. The receiver ‘B’ may interpret the concept differently as intended by the sender as the words may not refer to same idea.

2.1.2 Ambiguity

According to IEEE Recommended Practice for Software Requirements Specifications “An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation.”

A statement is ambiguous when it has more than one distinctive interpretation/meaning. Many SRS documents suffer from ambiguity. Ambiguous requirements lead to confusion, wasted efforts and rework.

Anderman and Rogers (2005), identifies the cause of ambiguities as:

i) A linguistic or syntactic hitch such as Polysemy words, erroneous word order, logical connectives (such as “AND” and “OR”), use of quantifiers (such as “someone”, “all”), synonyms, articles (such as “the”, “a”, “an”), number, use of tenses and speech act rules.

ii) A conceptual dilemma such as vagueness.
iii) An invariant figurative interpretation that requires the general knowledge to understand the concepts of events, properties and associations from the source domain to the target domain.

For instance, “I am going to bank”. Here the word “Bank” has 18 senses (according to Wordnet 2.1). Like a Financial Institution and a sloping land besides a water body. Moreover “bank” could be a noun (10 senses) or a verb (8 senses). For humans it is easy to extract the correct meaning of Polysemic words, intended by the writer, by considering the context in which it is used. For example “I am going to deposit money in the bank”. Here out of two senses discussed above, the financial institution is possibly more accurate given the context and world knowledge. However, for a machine it is still a challenge to disambiguate such statements.

2.1.2.1 Types of Ambiguity

We can distinguish following basic types of ambiguity in Natural Languages:

a) Lexical Ambiguity:

Lexical ambiguity occurs when a word in a sentence has multiple meaning. For example “bank”. Lexical ambiguities are caused by Polysemic words or homonyms.

a.1) Polysemic words: Polysemic words are those that have multiple but related meanings. For example “paper”. The word “paper” has meanings such as a report, a newspaper, a scholarly article, a medium for written communication. All these meanings are related to each other and are having the same etymological background.
a.2) **Homonyms:** in contrast with Polysemy words, Homonyms are the words having multiple meanings which are clearly distinct. For example “lie”. The word “lie” can be interpreted as ‘a statement that deviates from the truth’. It can also mean ‘a position or manner in which something is situated’. These meanings are clearly not related to each other.

b) **Syntactic Ambiguity:**

Syntactic Ambiguity, also called Structural Ambiguity, occurs when a given sequence of words can be given more than one grammatical structure, and each has a different meaning. In the terminology of compiler construction, syntactic ambiguity occurs when a sentence has more than one parse tree. We can distinguish following types of Syntactic Ambiguity:

b.1) **Analytical Ambiguity:** Analytical Ambiguity occurs due to presence of compound nouns. For example, ‘The Indian history teacher’.

The above example can be read as The (Indian history) teacher.

Or The Indian (History teacher).

b.2) **Attachment Ambiguity:** It occurs due to Prepositional Phrase attachment. For example, ‘The police shot the rioters with guns’.

Here the two possible interpretations are:

i) Police shot the rioters with the help of guns.

ii) The police shot the rioters, who are having guns.
b.3) **Coordination Ambiguity:** This type of ambiguity occurs due to the presence of more than one Conjunction or when one conjunction is used with a modifier. For example, ‘I saw Peter and Paul and Marry saw me’.

The possible interpretations of the above sentence can be:

i) I saw (Peter and Paul) and Marry saw me.

ii) I saw Peter and (Paul and Marry) saw me.

An example for second case can be, ‘intelligent boy and girl’.

The above sentence can be interpreted as:

i) Intelligent (boy and girl).

ii) (Intelligent boy) and girl.

b.4) **Elliptical Ambiguity:** This type of ambiguity occurs due to an ellipse (a gap in a sentence due to the omission of a necessary constituent). For example, ‘Ram knows an intelligent man than Shyam’.

Here, the above statement can have following meaning:

i) Ram knows a man, who is intelligent than Shyam is.

ii) Ram knows a man, who is intelligent than any person Shyam knows.

c) **Semantic Ambiguity:**

It is also called Scope ambiguity. Semantic Ambiguity arises when a sentence has more than one way of reading it within its context although it contains no Lexical or
Structural ambiguity. Syntactic ambiguity causes when a sentence has different structure i.e. parse tree, whereas Semantic ambiguity, the structure of the sentence remain same but the sentence can be interpreted differently. Semantic ambiguity is ambiguity with respect to logical form. For example, ‘Ram kisses his son, and so did Shyam’.

Now the possible interpretations are:

i) Ram kisses his son and Shyam also kisses Ram’s son.

ii) Ram kisses his own son and Shyam kisses his own son.

d) Pragmatic Ambiguity:

Pragmatic ambiguity occurs when a sentence has more than one interpretation in the context in which it is uttered. That is pragmatic ambiguity is ambiguity due to context dependent meaning. For example, ‘Every student thinks she is genius.’

Here the possible interpretations are:

i) Every student thinks she (herself) is genius.

ii) Every student thinks she (referred to someone) is genius.

e) Referential Ambiguity:

It is also called Anaphoric Ambiguity. Anaphor is an element (a phrase) that refers to something previously mentioned, possibly a different previous sentence. A referential ambiguity occurs, when an anaphor can take it reference from more than one element. For example, ‘How can you drop a raw egg onto a concrete floor without cracking it?’
Here ‘it’ can refer to egg or the floor, which produces two different meanings of the sentence.

2.1.3 Vagueness

A statement is vague, if it lacks clarity. Vagueness is closely related to ambiguity. For example, ‘it is cold outside’. Here different persons can assume different outside temperature, which can create confusion.

Vagueness should be avoided specially in the case of software requirements. An SRS statement is vague, if there is no precise way of describing the requirement and measuring whether it is fulfilled or not.

For example, ‘The system should have fast Response time’

Here, it is very difficult and objective to precisely measure the value of ‘fast’ and hence there is no way to validate that the requirement is fulfilled or not.

It is recommended that, requirement statement should specifically state the precise requirement. For example, instead of stating that ‘the system should operate at normal temperature’, the unambiguous statement will be like, ‘the system should operate at temperature between 20°C to 30°C, including 20°C & 30°C."

Vague or incomplete requirement documents have missing information the developers need to develop an efficient system. If the developers/testers can’t think of test cases to verify whether the requirements are fulfilled or not, that means the requirements are vague and SRS document is not well defined.
2.1.4 Requirement Engineering

Requirement Engineering is the process of defining, documenting and maintaining the expectations of all the stakeholders for a new or modified product. In the waterfall model, requirement engineering is presented as the first phase of the software development life cycle. Later on due to importance of requirement engineering, the development models like extreme programming, Scrum and Rational Unified Process recommends that requirement engineering should continues through the life time of the system.

In industrial requirement engineering (RE), the most frequently used representation to state the requirements that are to be met by information technology products or services is natural language. Natural language is a global and flexible medium of communication, but the drawback of Natural Language as a communication medium is that Natural Language is intrinsically ambiguous. Even worse, in case of formal communication like requirement elicitation and documentation, often neither customers nor software developers identify an ambiguity as each of the stakeholders derives an interpretation that is different from that of others without observing this. As a result, the software developers design and implement a system that does not perform as anticipated by the customers.

The use of Semi-formal and formal representation techniques, such as Unified Modeling Language (UML) or Software Cost Reduction, have been proposed to counter the weakness of natural language. However, they only budge the problem; an ambiguous requirement statement simply becomes an unambiguous wrong specification statement, which must be discovered by the stakeholders in reviews.
**Steps in Requirement Engineering:** the various activities involved in requirement engineering include:

i) Requirements elicitation: It is the process of discovering the needs of all the stakeholders and system constraints.

ii) Requirements identification: In this process all the needs of the stakeholders are identified.

iii) Requirements analysis: It is the process of reviewing and finalizing the concrete and unambiguous requirement specifications.

iv) Requirements documentation: A Software requirement specification (SRS) document is prepared.

v) System modeling: The system to be developed is modeled using some modeling techniques like UML.

vi) Requirements validation: Checking that whether the modeled system is consistent with the documented requirements and stakeholders need or not.

vii) Requirements management: Managing changes to the requirements as the system develops.
2.1.5 Techniques for dealing with ambiguity in Requirement Engineering

Traditional techniques to reduce the intensity of ambiguities in software requirement specifications documents can be divided into three categories depending in which requirement engineering activity it is performed (Berry, 2000).

a) **Requirement elicitation:** In this phase, two distinguish strategies can be followed to minimize the ambiguity in SRS documents.

i) As language is always interpreted in context, it is recommended to establish a context. If context is not established explicitly, on which all the stakeholders are agreed, then it is more likely that confusion of interpretation may occur.

ii) It is recommended that, requirement engineer should paraphrase, what he understood from the stakeholders’ statement in her own words. This is an effective practice for the requirement engineer to get the stakeholders to pinpoint their own ambiguities.

b) **Requirement documentation:** In this phase there are three different strategies which can be followed to minimize the ambiguities in SRS documents.

i) Selection of precise statements to represent the requirement specifications in natural languages.
ii) Use of world knowledge (general knowledge) can help to reduce ambiguities present in a statement. So it is recommended that more contextual information should be provided to the users.

iii) Conventions on how a statement should be interpreted in case of ambiguous statement/phrases shall be negotiated between all the stakeholders.

c) **Requirement validation:** The common strategies used in this phase of software engineering to reduce ambiguities are:

i) Use of formal languages, as formalization of informal natural language statements enforces precision. It helps in exposing the possible ambiguities.

ii) Use of indicators and checklist. There are some tools available which can identify or indicate the possible occurrence of ambiguities based on some keywords.

iii) Get the interpretations of the SRS document by different stakeholders. If these interpretations differ from one another, that implies there is ambiguity in the document.

iv) Communicating the understanding of the document back to the author, after which she can point out misinterpretations effortlessly.

### 2.1.6 WordNet

WordNet® is a large lexical database of English language (About WordNet, 2010). It is a system for bringing together different lexical and semantic relations between the words.
Nouns, verbs, adjectives and adverbs are grouped into synsets. Synsets are cognitive groups, each representing a distinct concept. WordNet® thus can be seen as a combination of dictionary and thesaurus. WordNet® provides relationships among these synsets and their members. This structure makes WordNet® a useful tool for computational linguistics and natural language processing. WordNet can be downloaded or accessed online through web browser. The latest online version of WordNet® is 3.1. It contains 155287 words organized in 117,659 synsets for a total of 206,941 word-sense pairs.

The main relation among different words in WordNet® is synonymy. Synonyms are those words which represent the same concept and can be used interchangeably in a context.

2.2 Study of related work

Word Sense disambiguation (resolving lexical ambiguity) is an open problem of natural language processing. It is a bottleneck of many NLP applications such as machine translation, information retrieval systems, and sentiment analysis. It is a more serious issue in software development. Practically, the initial documentation of requirement specifications is done in any natural language. An online survey of business requiring software (Mich Luisa, 2004) shows

i) 71.8% of the requirement documents are written in natural languages.

ii) 15.9% of the requirement documents are written in structured language.

iii) 15.3% of these SRS documents are written in formalized language.
The results produced by the survey, clearly indicates that the primary mode of expressing requirements is natural languages. As natural languages have intrinsic property of being ambiguous, the SRS documents produced contains ambiguity most of the time.

Researchers have worked a lot to solve this problem. Many systems have been developed for word sense disambiguation with sufficiently high level of accuracy and precision. A variety of techniques have been developed from knowledge based approaches or dictionary based approaches where dictionaries (Lexical resources) are used to disambiguate the word, to supervised machine learning approaches, where machine learn to classify the senses from manually labeled training sets, to completely supervised approaches that are based on unlabeled corpora.

2.2.1 Approaches to deal with Lexical Ambiguity (WSD)

Main approaches to deal with word sense disambiguation or Lexical ambiguity can be categorized as follows:

i) **Supervised WSD**: These approaches use machine learning techniques to train a classifier for each word on a corpus of manually sense annotated examples. A training set of examples encoded in terms of a number of features together with their correct class (sense label).

ii) **Unsupervised WSD**: These methods do not exploit any tagged corpus that is these approaches are based on unlabeled corpora to provide proper sense of a word in context.

Generally, supervised techniques for WSD have obtained better results than unsupervised approaches.
Another criterion that can be used to classify WSD methods is use of dictionary.

i) **Knowledge based WSD**: They are also called dictionary based or knowledge rich techniques. These approaches exploit the use of external Lexical resources, such as thesauri, machine readable dictionaries and ontologies.

ii) **Corpus based WSD**: They are called knowledge poor approaches. These techniques do not make use of such knowledge resources to provide the appropriate sense of a word.

Another classification of WSD techniques can be:

i) **Token based WSD**: These techniques relate a specific sense with each token i.e. with each occurrence of a word in the text, depending on the context in which it appears.

ii) **Type based WSD**: These techniques are based on assumption that a word consensually referred with the same meaning within a single text. These techniques have a propensity to surmise a sense for a word by the analysis of the entire document and associate the sense, which is predominant sense among all the possibilities, to each occurrence of the word in the document. Token based techniques can always be used to perform type-based word sense disambiguation by allocating the majority sense throughout the document to each incidence of the target word.
2.2.2 Supervised Word Sense Disambiguation

Supervised WSD techniques make use of machine learning methods to train a classifier from manually sense-annotated data sets. It is based on the classification task in order to provide the most appropriate sense to each instance of the target word. The training set is a manually labeled dataset in which each word is tagged with a sense from a sense inventory of a reference lexicon.

2.2.2.1 Decision List

Ronald L Rivest (Rivest, 2001), defined decision list as, a decision list is an ordered set of rules (list of weighted if-then-else rules), which are used to allocate the most appropriate sense to the target word. A training set is used for inducing a set of features. The resultant rule are like: (feature-value, sense, score). The decision list is created based on these scores.

According to D. Yarowsky (Yarowsky, 1994), the score of sense $S_i$ is computed as the maximum among all the feature scores. Where the score of a feature $f$ is calculated as the logarithm of the probability of sense $S_i$ given feature $f$ divided by the sum of the probabilities of the other senses given feature $f$:

$$
\text{score}(S_i) = \max_f \log \left( \frac{P(S_i | f)}{\sum_{j \neq i} P(S_j | f)} \right)
$$

2.2.2.2 Decision Tree

Pedersen Ted (Ted, 2001), described decision tree as, a decision tree, perform a search of feature space from general to specific case. The goal of this approach is to find out
a minimal set of features that can effectively classify the feature space into classes of manually sense-tagged examples of a target word. A decision tree represents classification rules with a tree-like structure, where each internal node represents a test on a feature value and each branch represents an outcome of the test.

### 2.2.2.3 Naïve Bayes classifier

Naïve Bayes classifier, based on Bayes theorem of conditional probability, is a simple classifier. It relies on the calculation of the conditional probability of each sense $S_i$ of a target word $W$ given the features $F_j$ in the context. The sense which has the maximum probability is chosen as the most appropriate sense of the target word in a given context.

### 2.2.2.4 Instance based learning

Instance based learning; also called Exemplar-based learning or memory-based learning, is a supervised algorithm in which the classification model is built by training of the system from examples. The model retains examples in memory as points in the feature space and, as new examples are subject to classification, they are progressively added to the model. One of the highest performing Exemplar-based methods is KNN for WSD.

### 2.2.2.5 WSD method using Support Vector Machines

This method is based on the idea of learning a linear hyperplane from the training dataset that separates positive examples from negative examples. The hyper
plane is located in that point of the hyper space which maximizes the distance to the closest positive and negative examples, called support vectors.

Table 2.1: Comparison of supervised WSD methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Avg. Precision</th>
<th>Avg. Recall</th>
<th>Corpus</th>
<th>Avg. Baseline Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision List</td>
<td>96%</td>
<td>--</td>
<td>Tested on a set of 12 highly polysemous</td>
<td>63.9%</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>64.13%</td>
<td>--</td>
<td>Senseval-3 All Word Task</td>
<td>60.9%</td>
</tr>
<tr>
<td>Instance Based</td>
<td>68.6%</td>
<td>--</td>
<td>WSJ6 containing 191 content words</td>
<td>63.7%</td>
</tr>
<tr>
<td>SVM</td>
<td>72.4%</td>
<td>72.4%</td>
<td>Senseval-3 Lexical sample task, used for disambiguation of 57 words</td>
<td>60.9%</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>79%</td>
<td>63%</td>
<td>Arabic Corpus</td>
<td>72%</td>
</tr>
<tr>
<td>KNN</td>
<td>--</td>
<td>--</td>
<td>Bengali Corpus</td>
<td>71%</td>
</tr>
</tbody>
</table>
2.2.3 Un-Supervised Word Sense Disambiguation

Unsupervised WSD identifies patterns in a large sample of data, without the benefit of any manually tagged examples. These patterns are used to divide the data into clusters; the members having more common features are clubbed in a cluster.

Unsupervised methods have the impending to surmount the knowledge acquisition bottleneck, that is, the lack of large manually tagged dataset resources with word senses.

The idea behind these approaches is that, the same sense of a word will have similar neighboring context words. They are able to induce word senses from input text by clustering word occurrences, and then classifying new occurrences into the induced clusters. They do not make use of any machine readable resources like thesauri, dictionaries etc.

However, the main drawback of these approaches is they cannot rely on a shared reference inventory of senses, as they do not use any dictionary.

The main methods for Unsupervised WSD are as follows:

2.2.3.1 Context clustering

This approach is based on representing each occurrence of a target word in a corpus as a context vector. These vectors are then clustered into groups, each group identifying a sense Si of the target word.

This method uses the concept of word space, that is, a vector space whose dimensions are words. A word w in a corpus can be represented as a vector whose jth component counts the number of times that word wk occurs with w within a fixed context. A context vector is built as the centroid, the normalized average, of the vectors
of the words occurring in the context. Finally sense discrimination can be performed by grouping the context vectors of a target word using any clustering approach.

2.2.3.2 Word clustering

Word clustering methods are based on clustering words which are semantically similar and can thus convey a specific meaning. D Lin (Lin, 1998), proposes a well known word clustering approach which is based on the idea of identification of words $W=\{w_1, w_2, ...\}$ which are possible synonyms/senses of a target word $w_0$. The similarity between the target word and its sense $w_i$ is evaluated by the information content of their single features, given by the syntactic dependencies which occur in a corpus (such as, e.g., subject-verb, verb-object, adjective-noun, etc.). The information content will be high if the two words have more dependencies. To differentiate between the senses, an approach called word clustering is applied. Let $W$ be the list of related words structured by degree of similarity to the target word. To start with, a similarity tree $T$ is created containing target word as a single node. Next, for each word in the list, a word is added as a child of word $w_i$ in the tree $T$ such that it is the most similar word to the parent node word. After the pruning is done, each subtree rooted at target word is considered as a distinct sense of the target word.

A different word clustering approach called Clustering by Committee (CBC) algorithm is proposed by Lin and Pantel (P Pantel, 2002). As above, initially it computed a list of all similar words for each target word. To evaluate the similarity, each word is represented as a feature vector, where each feature represents a syntactic context in which the word occurs. Given a set of target words, a similarity matrix $S$ is built such that $S_{ij}$ contains the pair-wise similarity between words $w_i$ and $w_j$. 
In second phase, given a set of words $W$, a recursive method is used to determine sets of clusters, known *committees*, of the words in set $W$. For this, a standard clustering approach, called, average-link clustering is implemented. In each step, remaining words which are not similar enough to the centroid of the any committee and not covered by any committee are recognized. More committees are discovered from residue words recursively. It is observed that, committees combine senses as each word $w_i$ belongs to a single committee.

Lastly, as a sense discrimination step, every target word $w_0 \in W$, characterized by a feature vector, is recursively allotted to its most similar committee, based on its similarity measure to the centroid of every committee. After a word is allotted to a committee, the overlapped features between the word and other elements in the committee are removed from the representation of the word, so that at later iterations, identification of senses of the target word which are less frequent can be performed.

### 2.2.3.3 Co-occurrence Graph

These approaches are based on the notion of a cooccurrence graph, that is, a graph $G = (V, E)$ whose vertices $V$ corresponds to words in a text and edges $E$ connect pairs of words which co-exists in a context.

Given an ambiguous word (target word) $w$, a local graph $G_w$ is built for the target word $w$. By normalizing the adjacency matrix related with $G_w$, we can interpret the graph as a Markov chain. The Markov clustering algorithm is then applied to determine word senses, based on an expansion and an inflation step, aiming, respectively, at inspecting new more distant neighbors and supporting more popular nodes.
Subsequently, V’eronis proposed an ad hoc approach called *HyperLex*. First, a cooccurrence graph is built such that the nodes are the words occurring in the paragraphs of a corpus in which the target word occurs. An edge is added between a pair of words if they co-exist in the same paragraph. Each edge is assigned a weight according to the relative co-occurrence frequency of the two words connected by the edge. As a result, words having high frequency of co-occurrence are allotted a weight close to zero, while words which rarely exist together in a paragraph will assign weights close to 1. The edges having weight above the threshold value are discarded.

In second step, an iterative algorithm is applied to the co-occurrence graph created in above step: at each iteration, the node having highest relative degree in the graph is designated as a *hub*. The algorithm terminates when the relative frequency of the word related to the selected hub is less than a fixed threshold value. The whole set of words selected as hub represents the possible senses of the target word. Words selected as hubs are then connected to the target word generating a graph with edges having zero weight. A minimum spanning tree (MST) of the generated graph is calculated which is used to disambiguate specific instances of our target words.

An alternative approach to identify word senses based on graph is *PageRank*. PageRank is a well-known algorithm developed for generating the web pages ranking retrieved from a search engine, and is the main ingredient of the Google search engine. In the adaptation of PageRank to unsupervised WSD, (Agirre & Edmonds, 2007) (2007) defined $w_{ji}$ is, as for HyperLex, a function of the probability of cooccurrence of words $wi$ and $wj$. PageRank algorithm provides the vertices of the graph sorted by their PageRank value. Those vertices are selected as a hub of the target word which has the best ranking.
Table 2.2: Comparison of Unsupervised WSD methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Avg. Precision</th>
<th>Avg. Recall</th>
<th>Corpus</th>
<th>Avg. Baseline Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin’s Approach</td>
<td>68.5%</td>
<td>Not reported</td>
<td>Trained using WSJ corpus containing 25 million words. Tested on 7 SemCor files containing 2832 polysemous nouns</td>
<td>64.2</td>
</tr>
<tr>
<td>Hyperlex</td>
<td>97%</td>
<td>82%</td>
<td>Tagged on a set of 10 highly polysemous French words</td>
<td>73%</td>
</tr>
</tbody>
</table>

### 2.2.4 Knowledge Based Disambiguation

The aim of knowledge based or dictionary based approaches for word sense disambiguation is to make use of knowledge resources like dictionaries, ontologies, collocations, thesauri etc to infer the most appropriate sense of the target word in the given context.

#### 2.2.4.1 Overlap of sense definitions

A simple and intuitive knowledge-based approach relies on the calculation of the word overlap between the sense definitions of two or more target words. This method is known as *gloss overlap* or the *Lesk* algorithm. Given a two word context \((w1, w2)\), the senses
of the target words whose definitions have the highest overlap (i.e., words in common) are assumed to be the correct ones.

As the Lesk algorithm is computationally complex due to exponential number of steps required, a variation of the Lesk algorithm is currently used. This method recognize the sense S of a target word w by identifying the sense whose lexical definition has the maximum overlap with the context words of w. Formally, given a target word w, the following score is computed for each sense S of w:

$$\text{score}_{\text{LeskVar}}(S) = |\text{context}(w) \cap \text{gloss}(S)|,$$

Where context (w) is the bag of all content words in a context window around the target word w.

The original Lesk’s algorithm achieved 50–60% accuracy (depending on the word), using a comparatively fine set of sense features found in a usual learner’s dictionary. However, as Lesk’s algorithm is very susceptible to the exact wording of lexical definitions, so the lack of a certain word can drastically change the results. Another limitation of Lesk’s approach is that it only considers the overlaps between the glosses of the senses. The performance of the approach may degraded in many cases as glosses are generally short and do not provide enough vocabulary to compare fine-grained sense characteristic.

Walker proposed an algorithm for WSD. The idea is to allocate each word, one or more subject categories in the thesaurus. If the word is assigned to several subjects then it is assumed that they correspond to different sense of the word.

Another approach for word sense disambiguation is based on conceptual density. This approach selects a sense of a target word based on the relatedness of that sense of the target word to the context. Relatedness is measured in the terms of how close the concept
represented by the word and the concept represented by its context words are, that is called conceptual distance. This method make use of a structured hierarchical semantic net (WordNet) for finding the conceptual distance.

2.2.4.2 Selectional Preferences

A historical type of knowledge-based algorithm is one which makes use of Selectional preferences to control the number of senses of a target word occurring in particular context. Selectional preferences are constraints on the semantic type that word sense inflicts on the context words with which it come together in sentences having some grammatical relationships. For example, the verb drink anticipates an animate individual as subject and a drinkable entity as its direct object. We can differentiate between Selectional restrictions and preferences, as Selectional restriction disregard the senses among all the possible cases that violate the constraint, whereas in Selectional preferences, we select those senses out of all the possible cases that better satisfy the requirements.

The easiest way to perform Selectional preferences is to identify the semantic suitability of the association provided by a word-to-word relation. The simplest way to measure this is frequency count. For a pair of words w1 and w2 and a syntactic relation R (e.g., subject-verb, verb-object, etc.), in this technique we counts the number of cases (R, w1, w2) in a corpus of parsed text, obtaining a figure Count(R, w1, w2). Another estimation of the semantic appropriateness of a word-to-word relation is the conditional probability.
Table 2.3: Comparison of Knowledge based WSD methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lesk algorithm</td>
<td>50%-60%</td>
</tr>
<tr>
<td>Walker's approach</td>
<td>50% when tested on 10 highly polysemous English words</td>
</tr>
<tr>
<td>WSD using conceptual density</td>
<td>54% on Brown corpus</td>
</tr>
<tr>
<td>WSD using Selectional Preferences</td>
<td>44% on Brown corpus</td>
</tr>
<tr>
<td>Adaptive Lesk algorithm</td>
<td>79%</td>
</tr>
</tbody>
</table>

2.2.5 State of the art in WSD

M.B. Minai (Minai, 2014), proposes Genetic algorithm based approach for word sense disambiguation applied on the Arabic corpus. The approach attains 79% precision and 63% recall.

M Hausman (Hausman, 2011), proposed a method based on Genetic algorithm to resolve word sense ambiguity. It exploits the semantic relations for word sense disambiguation. The method performs fairly well.

R pandit and S K Naskar (R Pandit, 2015), applied KNN based approach to deal with word sense ambiguity for Bengali language. The result shows accuracy over 71%.

C Zhang et.al (Zhang Chunxiang, 2015), presented an approach for WSD based on Bayesian model. It exploits the morphology information of word units to guide the word sense disambiguation process. They claim good performance of the system.
B S Rintyarna and R Sarno (B S Rintyarna, 2016), have proposed an adaptive Lesk algorithm for WSD which uses adaptive weighted graph to disambiguate the words. They claim 19% improvement in the proposed approach compare to original lesk algorithm.

P Sachdeva et al (P Sachdeva, 2014), proposes an algorithm which compute intersection (number of common words to calculate similarity) using not only the glosses but also by including the related words. Also the intersection is computed for the entire hierarchy of the target word and its contextual words. It also includes a third factor. This parameter ‘distance’ is the distance between the target words and the words in the same context. The proposed method is evaluated on SemCor and results shows average precision of 58%.

A Golkar et al (Ali Golkar, 2015), have optimize conceptual density’s context for WSD by applying pruning of nouns with negative impact. The results shows precision of 59.6% as compared to 52.8% of original conceptual density method.

J Sarmah and S K Sarma (J Sarmah, 2016), proposed a Naive Bayes classifier for WSD applied to Assamese. The result shows average accuracy of 78%.

S Vij et al (S Vij, 2016), have proposed an approach by fuzzyfying the semantic relations in WordNet to disambiguate a word. The method gives encouraging results.

2.2.6 State of the art to deal with Vagueness

Prateek Shrivastav and Akhilesh Tiwari (P Shrivastav, 2015), have used vague set theory with genetic algorithm to make use of hesitation information (the chances that a user hesitate to select a product) for profitability management. The results of the approach are fairly good, as they add new rules based on hesitations which are quite effective for decision making process.
Singh, A. K., & Tiwari, A. (A K Singh, 2015), have explored the use of vague concept for uncertainty and hesitation management. They have applied vague association mining rule to deal with uncertainty.

F A Lisi and U Straccia (F A Lisi, 2013), address the issue of incomplete and vague structured knowledge by providing Inductive Logic Programming (ILP) method for inducing fuzzy Description Logic (DL).

Alexopoulos, P., & Gómez-Pérez, J. M. (P Alexopouluos, 2012), have proposed a method which makes use of Fuzzy Ontologies to deal with vagueness in semantic business management.

2.2.7 Approaches to deal with ambiguity in informal SRS

To deal with ambiguity in SRS represented in informal language, one can convert the requirements in formal language representation. But the problem is an ambiguous informal requirement mostly lead to an ambiguous or incorrect formal representation.

To deal with ambiguous requirements, the most general approach is the use of Controlled Languages & Quality indicators.

**Controlled Languages**

Controlled languages are the natural languages with restricted vocabulary. By constraining the vocabulary usage, and allowing only predefined set of words to express the requirement, one can reduce the chances of ambiguity in a requirement specification document. For example, object constraint language (OCL), a declarative language to describe rules that applied to Unified Modeling Language (UML) models.
Quality Indicators

Many researchers have introduced some quality indicators like can, and, their etc which indicates possibility of ambiguity in the sentence.

Inspection Techniques

Many researchers have proposed the use of checklist to identify the ambiguity in a document. They suggested that once requirement documentation is ready, it should be inspected for the possible presence of ambiguity on the basis of checklist.

- Remarkable work done to deal with ambiguities in formal SRS using above techniques

Hayes et al. (Hayes, 1990), (J H Hayes, 2003), (J H Hayes A. D., 2004), have developed quite a few requirements tracing methods and tools for NASA, like keyword matching based Software Automated Verification and Validation and Analysis System (SAVVAS), Front End Processor (SFEP) and Information Retrieval (IR) based approach. This technique can be used for identifying whether a high level requirement is related to a low level requirement, to resolve inconsistencies.

Rolland and Proix (C Rolland, 1992), has proposed an algorithm to model a conceptual schema of the system to be developed by using linguistic method, which is used to model real world phenomena. They have built a tool called OICSI based on the natural language (for French) to mechanize the support for requirement engineering.

Ohnishi (Ohnishi, 1994), has developed a tool called X-JRDL analyzer, for Japanese language. This tool is based on the Requirements Frame model for the domain of file system. The Requirements Frame model differentiates among three different frames, i.e. Noun
Frame, Case Frame, and Function Frame. These frames direct the vocabulary and the context of the Requirement Statements. Ohnishi affirms that with Requirements Frame model, each Requirement Statement can be transformed into an internal representation called CRD (Conceptual Requirements Description). After that, X-JRDL automatically analyzes each requirement description given in CRD.

The concept of controlled language is used by many researchers for stating the software requirements. Controlled languages present a method to write a Software Requirement Specification document in a representation which is close to Natural Language with some restriction on vocabulary and uses simple sentences instead of complex one. The use of Controlled languages assists to represent requirement statements in a less ambiguous manner.

Fuch and Schwitter (N E Fuchs, 1996), has developed a tool known as Attempto. This tool makes use of a Controlled English language, which is almost Natural English Language with restricted syntax and semantics and a domain-specific vocabulary. This tool allows requirement engineers to put together Requirement Statements in domain specific concepts in an interactive way. The tool transform complete Requirement Specification in Controlled English into discourse representation structures using first-order predicate logic and optionally Prolog.

Wilson et al. (W M Wilson, 1997), presents by and large the quality characteristic of Requirement Specifications. The authors have grouped these quality factors in two aspects: the first one is quality attributes that define quality aspects of a requirement statement such as completeness, correctness, traceability, etc. and the second one is attributes of Requirement Statements that point out lack of quality. These are the presence of imperatives,
continuances, directives, options, and weak phrases. Software Assurance Technology Centre (SATC) has build a tool known as Automated Requirements Measurement (ARM) by incorporating these quality attributes and indicators proposed by Wilson et al. to assess the quality of Software Requirements Specification document represented in a natural language.

V. Barr (Barr, 1999), recognizes specification patterns or sentence patterns, which assist in the conversion of requirement statements specifies in unstructured and informal natural language into a formal representation. He affirms two dissimilar classes of patterns. The first class of pattern is: If-then patterns, expressed within the Rule-Scheme. The second class of patterns is those patterns which describe an overall valid fact expressed within schemes for consequences without conditions where the consequence part of a rule is realized if and only if the condition evaluates to true, which is also an overall valid fact.

Götz and Rupp (R Gotz, 1999), proposed a rule base system that keeps all the rules necessary to identify deficiency, weak phrases and possible ambiguities in Requirement Specifications. They have used three main conversion schemes to model the real objective of a stake holder in specifying the requirement statements. These schemes are: • deletion that leads to the reduction of the perception of a person to a scope he or she can deal with, • generalization that leads to a detachment of an experience from its context and to assume that the experience is overall valid, and • distortion that is related to nominalization i.e. a noun stands for a complex process such as the recording, the playback, the take off.

N. Juristo et al. (N Juristo, 2000), have discriminate requirements into two types, namely, static requirements and dynamic requirements. They have used Static Utility Language (SUL) to convert any static requirements into a formal language and Dynamic Utility Language (DUL) to convert any dynamic requirements into a formal language. After
that, the statements specified in SUL and DUL can be transformed into predicate logic. They also advised some guidelines for static and dynamic requirements to be used in rewriting the Requirement statements.

Mich and Garigliano (L Mich, 2000), have explored the use of a set of ambiguity indices in the measurement of syntactic and semantic ambiguity. Implementing their approach, they have developed a system called LOLITA.

E. Kamsties et al. (Erik kamasties, 2001), provides an effective inspection technique based on checklist and scenario based reading for discovering RE-specific ambiguities.

Fabbrini et al. (F Fabbrini, 2002), build up a tool called QuARS (Quality Analyzer of Requirements Specifications). This tool supports the analysis and quality assessment of Requirements Specification documents. QuARS make use of Quality indicators for Requirement Sentence Quality (RSQ) and Requirement Document Quality (RDQ).

R. L. Smith et al. (R L Smith, 2002), have proposed a method and developed a toolset which is used for thoroughly elucidating system properties which are given in any informal natural language. This tool is known as PROPEL (Property Elucidator). The objective of PROPEL is to provide accurate and complete system specifications. This tool is very effective in handling possible ambiguities in specific types of requirements. PROPEL is based on the concept of “prevention is better than cure” i.e. it prevents the generation of requirement statements which may have ambiguity rather than identifying the ambiguities in the generated statements.

C. Denger and his team (Denger, 2002), (C Denger, 2003), developed an approach for reducing the problem of vagueness in informal Requirement Specifications by using natural language patterns, authoring rules, and document templates. He summarizes distinct
language patterns such as Functional Requirement Sentence Patterns, Event Patterns, Reaction Patterns, Computation Pattern, Relationship Patterns, Exception Patterns, Patterns for special aspects, and Nonfunctional Requirement Sentence Pattern.

F. Chantree (Chantree F., 2004), builds an adaptive ambiguity notification tool that automatically alerts the user to the presence of potentially dangerous ambiguities in a requirement specifications document. He states that in natural language generation, it is enviable to retain some ambiguities. So in his approach, a notification of possible ambiguities is given to the users with the indication of how seriously they can affect the interpretations of different readers of the document. The level of tolerance of an ambiguity is adjusted by the response of the users upon seeing any instance of that ambiguity. The system removes ambiguities which people infer easily, leaving the nocuous ones. The later ones can then be analyzed and rewritten manually.

A tool developed by Joseph E. Kasser (Kasser, 2004), known as TIGER Pro (Tool to InGest and Elucidate Requirements Professional). The tool scrutinizes each requirement statement and produces a summary of deficiency it comes across in the SRS document. The method is based on probing for five types of potential defects in the Requirement Statements.

Martínez, A., Pastor López, O., & Estrada, H. (Martinez, Lopez, & Estrada, 2005), provides a mechanism to create a controlled specification of requirements by using language patterns. He discover various classes of patterns such as architectural patterns that indicate the high level architectures of a software system, patterns that focus on the programming aspects known as design patterns etc.
Popescu et al. (D Popescu, 2007), developed a tool **Dowser** to discover ambiguities and inconsistencies in Rstats of a Natural Language SRS. However, the system takes input represented in a constrained language, which makes this approach impractical.

Tjong et al., (S F Tjong, 2007), (S F Tjong D. M., 2008), have suggested some directing rules that assist a document writer to write Natural Language Requirement Specifications which are less prone to ambiguity. These rules can also be used as an inspection checklist to find ambiguities in a Requirement Specification. He has designed a tool known as **SREE**. This is a lexical analyzer for finding out cases of probable ambiguity in a requirement specification document. When SREE discover any occurrence of probable ambiguity, it notify it to the user about it, the user then make decision whether the instance is truly ambiguous and to disambiguate the instance and rewrite the Rstat if desired. The guiding rules used in SREE are based on inference logic. However, a reassessment of the language patterns indicates that Logic Rules are not very efficient for fully resolving coordination ambiguity in a Requirement Specification document.

A tool for commercial use to identify ambiguity, incompleteness and inconsistency in a software requirements document is also available known as **Requirements Assistant**. This tool helps to analyze and review requirements at sentence level, paragraph level, and the document as a whole.

Kiyavitskaya et al. (N Kiyavitskaya, 2008), presented a couple of tools to identify ambiguity in an Rstat of a requirement specification document. Their proposed method comprises of two phases. In the first phase various procedures are applied to discover lexical and syntactic ambiguities in the RS. In second step the tool is used to measure what is potentially ambiguous specific to each sentence.
Hui Yang et al. (Hui Yang, 2010), describes an automated approach for detecting nocuous ambiguities in RS, which carry a high risk of misunderstanding among different stake holders. They have implemented a prototype tool for Nocuous Ambiguity Identification (NAI). The tool first spots and extract sentences that have specific types of ambiguity in a given NL SRS document. LogitBoost algorithm is then used to classify whether an ambiguous sentence is nocuous or innocuous. The heuristics used to train the system are based on statistical information (e.g., word distribution and collocation) obtained from BNC. Some of the heuristics used to train the system are coordination matching, distribution similarity, and collocation frequency.

Nikora, A., Hayes, J., & Holbrook, E. (Nikora, Hayes, & Holbrook, 2011), provide an automated approach for identifying ambiguity in informal requirement specifications. They have used a dataset of sentences which were manually tagged as ambiguous or non ambiguous. They have transformed the text strings into vector representation using word count, term frequencies & TFxIDF (where TF is term frequency and IDF is inverse document frequency) and employed 29 classifiers using WEKA tool. Their work illustrates that using Part of Speech information and TFxIDF representations yields better performing classifiers than word counts and frequencies.

A Umber and I. S. Bajwa (A Umber, 2011), describes an automated approach to produce SBVR (Semantic of Business Vocabulary and Rules) standard based controlled representation of English SRS. There approach gives encouraging results with average recall of 86.76% and average precision of 89.39%.

Carlos Huertas et al. (Carlos Huertas, 2012), propose some guiding principles for a controlled sentence structure for writing the software requirements. They also provide a
prototype tool for automatic evaluation of the quality of a NL SRS based on the grammar structure of sentences.

Ayan Nigam et al. (Ayan Nigam, 2012), provides a tool for assessment of the requirement specification documents by identifying Lexical, Syntactic and Semantic ambiguities in the RS and notify about the possible sources of wrong interpretation.

2.3 Conclusion

This chapter provides the basic concepts of Requirement Engineering, Natural Language Processing, and Ambiguities in section 2.1.

The section 2.2 provides the overview about the past work and the literature survey done to know the state of the art in the field of disambiguation of documents represented in natural languages. This chapter helps the readers to be familiarized with the idea of methods and techniques proposed by various researchers in the past to resolve the ambiguities present in a document. The chapter also provides a comparison of different approaches used by the researchers based on the reported results.

The section 2.1.5 discusses the methods used to resolve the possible ambiguities in an SRS document which are traditionally used by RE Engineers. These methods are very time consuming and complex as there is a need of participation of every stakeholder. The method involves the meetings of stakeholders to discuss the most appropriate interpretation of the ambiguous statements in an SRS document.

The section 2.7 deals with the approaches proposed by different scholars for handling ambiguities in SRS documents. These approaches are only used for identifying the possible ambiguities in an SRS document. Ambiguities are then resolved manually.
So, there was a need of a system which can not only identify the possible ambiguities but also resolve it.