An Experimental Study of Selected Methods towards Achieving 100% Recall of Synonyms in Software Requirements Documents

by

Xiaoye Lan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2015

© Xiaoye Lan 2015

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Software requirements documents written in natural language need to avoid the use of synonyms to reduce unnecessary confusion and ambiguity. In practice, synonyms are still common and are widely used in requirements documents. Lots of tools to identify synonyms have been developed. To evaluate these tools, two metrics are often used: recall and precision. Recall is the ratio of the number of relevant records retrieved to the total number of relevant records in the document. Precision is the fraction of retrieved records that are relevant. Industry practice leads us to believe that 100% recall is preferred over 100% precision for such tools. Available tools never actually achieve 100% recall. The goal of this thesis is to explore computational methods that could reach 100% recall in extracting synonyms from software requirements documents.

This thesis compares six WordNet-based methods and two context-based algorithmic approaches to extract synonyms from two different types of requirement documents. The eight methods were compared by their recall. The experiments results showed that the word co-occurrence-based method achieved the best recall in identifying synonyms of the software requirements documents. Further experiments showed that setting the parameters of the word co-occurrence-based method impacts the results of the experiments as well. The thesis also discusses potential issues of the word co-occurrence-based method in the design of the experiments. The document author's personal factors could influence the experiment results, but this influence can be avoided with careful design.

Acknowledgements

Firstly, I would like to thank my supervisor, Professor Daniel Berry, for his countinous support of my graduate study, without which, this thesis would not have heen possible.

Besides my advisor, I would like to thank my readers, Professor Lin Tan and Professor Richard Trefler, for their insightful comments and valuable time.

I also would like to thank my friend, Michael Wexler, for his help and encouragement.

Last but not the least, I would like to thank my family for the support they gave me througout writing this thesis and my life in general.

Dedication

This thesis is dedicated to the ones I love.

Table of Contents

List of Tables viii

List of Figures ix

1 Introduction 1

1.1 Motivation . . . 1

1.2 Previous Research Survey . . . 2

1.2.1 Generic NLP methods . . . 2

1.2.2 NLP methods and non-NLP methods in software engineering . . . 4

2 Research Methods 5

2.1 WordNet Based Methods . . . 5

2.1.1 Path-Length-Based Similarity Methods . . . 5

2.1.2 Information-Content-Based Methods . . . 7

2.2 Algorithm Based Methods . . . 10

2.2.1 Method from SWordNet . . . 10

2.2.2 Word Co-occurrence Vector Based Method . . . 12

3 Experiment and Evaluation 15

3.1 Evalution Methods . . . 15

3.2 Experiment Preparation . . . 16

3.2.1 Experiment Synonym Preparation 16
3.2.2 Preparation of the software requirements document for the experiment: an RFP 17
3.2.3 Experiment Result and the Analysis 19
3.3 Evaluation of the Experiment Results with a User Manual 21
3.4 Experiment with Different Parameters in the Word Co-occurence Vector Method 21
3.5 Threats and Further Considerations 23
3.5.1 Experimental Materials 24

4 Conclusion **25**

APPENDICES **26**

A Request for Proposal Documents Used in the Experiemnt **27**
A.1 RFP for Desktop Publishing Software with Replaced Synonyms 27

References **45**

List of Tables

2.1 Semantic Matrix for Sentence "You can also listen to the music and adjust the volume" . . . 13

3.1 Word replacement . . . 18

3.2 Number of returned results for each level recall . . . 19

3.3 Synonyms in iPhone user manual . . . 22

3.4 Number of returned results for 100% recall . . . 22

3.5 Number of returned results for 100% recall for different parameters . . . 23

List of Figures

2.1 An example of the hypernym hierarchy in WordNet [19] 6
2.2 An example of windows size 5 12

Chapter 1

Introduction

1.1 Motivation

In software requirement engineering, identifying synonyms in requirements documents is a common research topic and a substantial practical challenge [1]. The study of this thesis is to identify a suitable document processing method from a carefully selected list, using computational experiments.

The purpose of a requirements document is to gather clients' needs for a software system. Accurate understanding of the document directly impacts the quality of the system. Although there are some formal methods, natural language is still the main vehicle for such documents [14]. Natural language is an effective and useful tool to communicate requirements to stakeholders about software. However, it also presents some unique challenges for specifying software requirements. One of the challenges comes from the fact that to express the same meaning, natural language provides a choice of various words. In software requirements documents, terminology consistency is a critical success factor. A requirements document is often the product of many people, and different people often express the same meaning with different words. That phenomenon inevitably introduces synonyms into the requirements documents. Two words are synonyms when they share at least one word sense. A word sense is one meaning for a word. Synonyms do not have to have all their word senses in common; they just need to have at least one sense in common. For example, "book" and "reserve" are synonymous when both are verbs and are used in the context of "arrange things in advance". They are not synonyms when "book" is used as a noun. "Book" cannot be replaced with "reserve" in the sentence: "I read lots of books". Furthermore, there are other words that are semantically related, but are not

synonyms. They may be antonyms, hypernyms, etc. In a specific domain, hypernyms could be synonyms as well.

Synonyms in requirements documents can cause problems to human readers due to the fact that synonyms can bring unnecessary uncertainty about meanings of words into documents. For example, if the writer takes "program" and "software" as synonyms and uses them interchangeably to describe the software to be developed, the reader may wonder if "program" and "software" are different, especially if there are multiple pieces of software mentioned in the document.

The presence of synonyms in requirements documents can cause trouble also for requirements enigeering tools. For example, to generate a UML diagram from a natural language requirements description, synonyms have to be identified in order to prevent duplicated concepts in the diagram [21]. However, synonym identification is not an easy task. One of the challenges was illustrated by an abstraction identifier process, described by Goldin and Berry [7]. In their research, the algorithm of an abstraction finder tool relies on mulitiple occurrences of important words that name abstractions. They found that without a synonyms file, some abstractions in the document could not be identified because each of synonyms appears too few times to be counted as an abstraction. To help deal with the issue, researchers have been using WordNet to identify the synonyms in a requirements document first, and then taking that synonyms list as a reference file in processing the requirements document. That brings some results, but it still could fail to resolve all the issues. Words often have ambiguous meanings. A word could mean different things in different contexts. The word "assurance", for example, means a particular set of software engineering practices in requirements documents, but has a much broader and looser meaning in other contexts.

This thesis is to explore, by reviewing previous research and analyzing the particular needs of requirements engineering, a suitable and effective way for synonym identification in software requirements documents

1.2 Previous Research Survey

1.2.1 Generic NLP methods

Identifying semantic similarity words has been studied for a long time in natural language processing, and it is still an ongoing research topic [23]. A lot of NLP methods have been designed and tested to extract synonyms. These methods can be classified into four categories:

1. methods that are based on the distributional hypothesis [8]: Words occurring in the same context may have the same meaning.

2. methods that are based on a thesaurus, or an online lexical dictionary, such as WordNet [11] [27]: These methods leverage the structure of the taxonomy in the dictionary. To measure the similiarity of two words, these methods use the distance between two senses in the taxonomy, where the distance is defined as the number of edges between two words in the taxonomy.

3. methods that work on the assumption that two words mean the same if they have the same translation [24].

4. methods that use patterns in a corpus to extract synonyms. The patterns represent the relations between two words [22].

All of the above methods have achieved good results in processing general documents in different ways, but none has been able to meet one fundamental need of software requirements engineering: 100% recall. Software requirements engineering involves understanding and analysis of the contents of requirements documents. Current NLP techniques use lexical and syntactic information to learn semantic information. Lexical and syntactic tools cannot achieve 100% recall [2]. In software development practice, if a synonym identification method cannot provide 100% recall of the synonyms, the system analyst would have to search through the full document manually to find what has missed by the tool. So the tool is of no real help.

People may think that NLP tools provide high precision in synonym identification. Even if this is true, it is not the preference of an analyst in the software development domain. It is usually much much easier for an analysts to cross off the false positives than to find the synonyms in the document that could not be identified by the tool. So in a practical sense, for a synonym identification tool for requirements documents, people favor high recall over high precision.

One reason why NLP techniques cannot get 100% recall of the synonyms from the software requirements documents is the fact that NLP models or methods are usually generated from the general English corpus. Some words are synonyms only within a specific domain. These words may not be synonyms in general English context. For example, “page” and “notify” may be synonyms in the software development domain, but they are not synonymous in general English sense.

1.2.2 NLP methods and non-NLP methods in software engineering

There has been synonym research that focuses on software development area. Sridhara, Hill, et al [23] performed a comparative study of six methods that used English-based semantic similarity as their identification technique. They evaluated the effectiveness of the methods to identify synonyms in software code comments and to find identifiers in software's code. Their study shows that each method needs to include a large number of synonym candidates in order to find the actual synonyms that are used in the software [23]. Based on the experiences, they gave two suggestions to customize the techniques for extracting synonyms from software. One suggestion is to enhance WordNet with software-domain-specific-synonyms. The other suggestion is to improve the word probability used in the information content-based technique. That probability can be derived from word usage in software comments. Yang and Tan [28] realized that there are problems with the probability method. So they proposed a simple context-based similarity measure, SWordNet, to identify semantically related words in comments and code. Their approach achieved very nice performance in recall and precision. More details about this method are discussed in Chapter 3. Kawai, Yoshikawa, et al [10] proposed an approach to extract compound words from requirements documents. Their method employs the similarity of the contextual information shared by the words to extract the synonyms. Compound words are split into morphemes. A compound word's weight is determined by the aggregation of its morphemes. This method was applied to a real Japanese requirements document and achieved good performance in extracting synonyms. However, in all this research, recall was not the key objective. Based on their evaluations, none of these methods achieved 100% recall on their tasks either.

Chapter 2

Research Methods

This study chooses six popular WordNet-based methods and two algorithm-based methods to analyze. The two algorithm-based methods were expected to have the largest possibility to achieve 100% recall.

2.1 WordNet Based Methods

The six WordNet-based methods can be classified into two categories: (1) path length based similarity and (2) information content based similarity. Each category of methods uses the structure of WordNet to measure two words' similarity.

In WordNet, words are organized by their senses into the hypernym (is-a) hierarchy. Figure 2.1 shows a fragment of the WordNet hypernym hierarchy. Verbs and nouns are in separate hypernym hierarchies. The relation between an adjective and an adverb cannot be described by "hypernym". So one of the limitations of WordNet based methods is that they can compute only the similarity of noun-noun and verb-verb pairs. They cannot measure the similarity between noun-verb pairs and other kinds of pairs.

2.1.1 Path-Length-Based Similarity Methods

The simplest path-length-based-method works under the intuition that the shorter the path between two words in the WordNet hierarchy, the more similar the two words are. From Figure 2.1, dime is more similar to coin than to money. This method could be

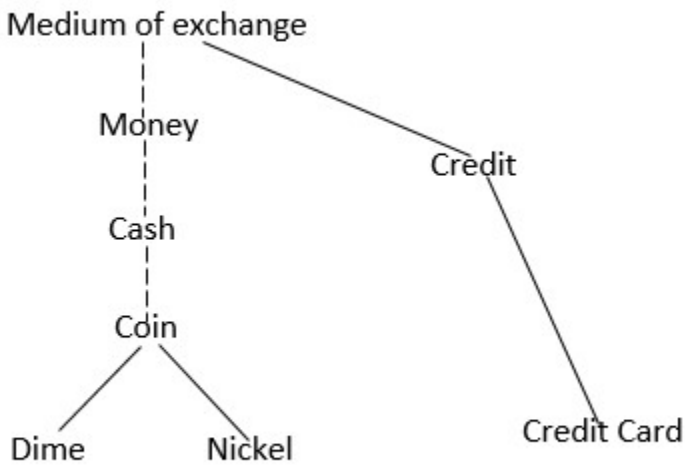


Figure 2.1: An example of the hypernym hierarchy in WordNet [19]

implemented simply by counting the number of edges in the shortest path between two words or senses in the graph defined by the WordNet hypernym hierarchy. This simplest method replies on the assumption that the edges in the hypernym hierarchy are uniform. But in practice, the edges near leaves represent narrower distances than the edges near the root of the hypernym hierarchy. For example, in Figure 2.1, the edge between "nickel" and "coin" represents more similarity than the edge between "money" and "medium of exchange". Both the Leacock & Chodorow (LCH) [11] and the Wu & Palmer (WUP) [27] methods consider this issue. They use the depth of the hierarchy to normalize the length of the path. In LCH method, the similarity of two words is defined as:

$$Consim_{LCH}(c_1, c_2) = -\log \frac{len(c_1, c_2)}{2D}$$

Here $len(c_1, c_2)$ represents the path length, i.e., the number of edges between the two senses and D is the depth of the hierarchy, the max path length from the root to any leaf. The maximum of similarity value of all the sense pairs of two words is taken as the two words' similarity value:

$$sim_{LCH}(w_1, w_2) = \max_{\substack{c_1 \in sense(w_1) \\ c_2 \in sense(w_2)}} Consim_{LCH}(c_1, c_2))$$

Also WUP uses the depth of the hierarchy to normalize the similarity. But unlike LCH similarity, which directly uses the depth of the whole hierarchy, WUP similarity uses the depth of each sense along with the depth of the least common subsumer (LCS) to define similarity. The LCS of two senses is the lowest node in the hierarchy which is a hypernym of both senses. The WUP similarity is defined as:

$$Consim_{WUP}(c_1, c_2) = \frac{2 \times depth(LCS(c_1, c_2))}{depth(c_1) + depth(c_2)}$$

2.1.2 Information-Content-Based Methods

Also methods based on the information content rely on the structure of the WordNet. However, in order to avoid unreliable edge distances, these methods leverage the probability that a concept occurs in a corpus. "Concept" here is a word sense in WordNet. The probability $P(c)$ of the concept c is the probability that the word sense c to be randomly selected in the corpus. In the WordNet hierarchy, the higher the concept is, the higher its probability is. An occurrence of a word in the corpus should be counted for all of the instances of the concept. For example, in Figure 2.1 , when "nickel" occurs in the document, the probability of "coin", "cash", "money" should all increase. The information content is defined as negative the log likelihood:

$$IC(c) = -logP(c)$$

When the probability of concept increases, the information content gets lower. Under the intuition that the more information two concepts share, the more similar the two concepts are, Resnik[19] first proposed a method using this structure. He used the information content of the LCS to represent the common information content. The similarity measure is

$$sim_{RES}(c_1, c_2) = -logP(LCS(c_1, c_2))$$

Lin [12] extended Resnik's method and pointed out that the similarity measure between two concepts should be more than the measure of the commonality between two concepts.

The more differences are there between A and B, the less similar they are. Lin also proved that the similarity between A and B is measured by the ratio between the amount of information needed to state the commonality between A and B, as well as the information needed to fully describe what A and B are. Based on this idea, Lin's similarity measure is defined as:

$$sim_{LIN}(c_1, c_2) = \frac{2 \times logP(LCS(c_1, c_2))}{logP(c_1) + logP(c_2)}$$

Also, Jiang and Conarh (JCH)'s [9] approach employs the notion of information content. Their approach combines a lexical taxonomy structure with corpus statistical information. The method based on the edge in the taxonomy could be improved by the computational evidence derived from the distributional analysis of data. Their method expresses the similarity of words in a distance format:

$$Dist_{JCH}(c_1, c_2) = 2 \times logP(LCS(c_1, c_2)) - (logP(c_1) + logP(c_2))$$

Using the distance of two concept, the similarity of two words is

$$sim_{JCH}(w_1, w_2) = \frac{1}{Dist_{JCH}(c_1, c_2)}$$

To illustrate the difference between the six methods, the similarity between "dime" and "nickel" in Figure 2.1 is computed using the six different methods. The path length between "dime" and "nickel", len("dime", "nickel") is 2, and the depth of the hierarchy is 4. By the definition of the LCH method, the similarity between "dime" and "nickel" is

$$\begin{aligned} Consim_{LCH}(\text{“dime”, “nickel”}) &= -\log \frac{len(\text{“dime”, “nickel”})}{2D} \\ &= -\log \frac{2}{8} \\ &= 0.6 \end{aligned}$$

The LCS of "dime" and "nickel" in Figure 2.1 is "coin" and the depth of "coin" in the hierarchy is 3. The depth of each of "dime" and "nickel" is 4, so the similarity of "dime" and "nickel" computed by WUP is

$$
\begin{aligned}
Consim_{WUP}(\text{“dime”, “nickel”}) &= \frac{2 \times depth(LCS(\text{“dime”, “nickel”}))}{depth(\text{“dime”}) + depth(\text{“nickel”})} \\
&= \frac{2 \times 3}{4 + 4} \\
&= 0.75
\end{aligned}
$$

Suppose there is a corpus containing 100 words. “dime” occurs 10 times, “nickel” occuers 12 times, and “coin” occurs 4 times. Using the RES method to compute the similarity of “dime” and “nickel” gives

$$
\begin{aligned}
sim_{RES}(\text{“dime”, “nickel”}) &= -\ logP(LCS(\text{“dime”, “nickel”})) \\
&= -logP(\text{“coin”}) \\
&= -log\frac{10 + 12 + 4}{100} \\
&= 0.59
\end{aligned}
$$

Using the LIN method, the similarity of “dime” and “nickel” is

$$
\begin{aligned}
sim_{LIN}(\text{“dime”, “nickel”}) &= \frac{2 \times logP(LCS(\text{“dime”, “nickel”}))}{logP(\text{“dime”}) + logP(\text{“nickel”})} \\
&= \frac{2 \times logP(\text{“coin”})}{logP(\text{“dime”}) + logP(\text{“nickel”})} \\
&= \frac{2 \times (-0.59)}{-1 - 0.92} \\
&= 0.61
\end{aligned}
$$

Similarly, the similarity of “dime” and “nickel” computed by the JCH method is

$$
\begin{aligned}
Dist_{JCH}(\text{“dime”, “nickel”}) &= 2 \times logP(LCS(\text{“dime”, “nickel”})) - (logP(\text{“dime”}) + logP(\text{“nickel”})) \\
&= 2 \times logP(\text{“coin”}) - (logP(\text{“dime”}) + logP(\text{“nickel”})) \\
&= 2 \times (-0.59) - (-1 - 0.92) \\
&= 0.74 \\
sim_{JCH}(\text{“dime”, “nickel”}) &= \frac{1}{Dist_{JCH}(\text{“dime”, “nickel”})} \\
&= \frac{1}{0.74} \\
&= 1.35
\end{aligned}
$$

2.2 Algorithm Based Methods

This section introduces two simple algorithm-based methods. These two methods are considered as dumb methods, since they use only the computational algorithm, without the assistance of any NLP technique, such as POS [2]. Each of the two algorithm-based similarity methods is based on the distributional hypothesis that words that occur in similar contexts may have similar meaning [8]. The variations among this kind of methods are in how to define the context and in how to measure similarity. Yang and Tan [28] propose a simple method based on the context shared by two sentences. The other method is very similar to the hyperspace analogue language method (HAL) [13]. A word is represented by its word co-occurrence vector. Similarity of two words is measured by the distance between two words' vectors.

2.2.1 Method from SWordNet

The SWordNet method leverages the context of words in the comments and code to identify semantically related words. The original purpose of SwordNet method was to improve searching in source code. Since the method has good performance in finding semantically related words among comments, it's a good candidate for extracting the semantic related words in requirements documents.

The approach first splits comments and identifiers into word sequences, and then clusters the word sequences based on shared words. In each cluster, all the sentences share at

least one word. After this preparation, the main step of this method is to calculate the semantic similarity between two word sequences and extract the corresponding semantically similar words. This method applies the longest common sequences to find the overlap between two word sequences. The longest common word sequence $LCWS(S_1, S_2)$ is defined to be the longest sequence of words shared between S_1 and S_2. The similarity measure of two word sequences S_1, S_2 is defined to be

$$similarityMeasure(S_1, S_2) = \frac{wc(LCWS(S_1, S_2))}{min(wc(S_1), wc(S_2))}$$

where the function $wc(S)$ represents the number of words in a word sequence S.

If the similarity value is greater than or equal to a chosen threshold, then semantically related words can be extracted from the differences between the two sequences. For example, let's set the threshold to be 0.7 and two sentences are "The software must provide the functionality to specify page margins for the publishing document." and "The software must provide the feature to specify page margins for the publishing document." The longest common sequences for the two sentences are "the software must provide the to specify page margins for the publishing document". The similarity of two sentences is 0.9, which is greater than the threshold. The word pair "functionality" and "feature" could be extracted from the word sequences as candidates for the semantically related words.

One problem of this simple way is that it doesn't consider the possible weights of words. Some words are less meaningful than others. To overcome this problem, the inverse document frequency (idf) is introduced into the measure. The function $idf(t, d, D)$ reflects how important a word t is in a document d, that is an element of a document, D:

$$idf(t, d, D) = log_{base} \frac{|D|}{|d \in D : t \in d|}$$

In the SWordNet method, the normalized idf is used to make the function $idf(i, d, D)$'s range within $[0, 1)$. The normalized idf is

$$normalized_{idf}(t, d, D) = \frac{e^{idf(t,d,D)}}{1 + e^{idf(t,d,D)}}$$

After leveraging the normalized idf, the new similarity measure is

$$NewSimilairtyMeasure(S_1, S_2, S_c) = \frac{2 \times \sum_{normalized_idf}(S_c)}{\sum_{normalized_idf}(S_1) + \sum_{normalized_idf}(S_2)}$$

$\sum_{normalized_idf}(s)$ is the sum of the normalized idfs of all the words in the word sequence s.

To improve the extraction of semantically related words, this method preprocesses the input by removing all the stop words and normalizing the words to their base forms.

2.2.2 Word Co-occurrence Vector Based Method

Another method to evaluate is the Hyperspace Analogue to Language (HAL) model. The HAL model is a high dimensional semantic space. Word co-occurrence relations are recorded by a word co-occurrence matrix. Each word in the corpus is presented in the rows and in the columns of the matrix. One important parameter in this method is window size. For a target word t, the window size W defines how many words before and after t should be considered as co-occurrence words. The $CoValue$ (co-occurrence value) of each word w with respect to the target word t in a window of size W is equal to W minus the distance from w to t.

$$CoValue(w, t, W) = W - D(w, t)$$

where $D(w, t)$ is the distance between w and t, which is defined to be the number of words between w and t in the corpus.

The closer a word w is to the target word in the document, the higher is its co-occurrence value. In Figure 2.2, let the window size be 5, and let the target word be "functionality". The co-occurence value for the word "provide" to the target word is 4, since there is only one word "the" between "provide" and "functionality". When a word w appears more than once around the target word, the co-values for all the occurences of w are summed up in order to determine its total $CoValue$. In the same example, word "the" appears twice within the window size. The co-occurrence value of the first "the" is 5 and the co-occurence value of the next "the" is 1. So the total $CoValue$ of "the" is 6, the sum of 5 and 1.

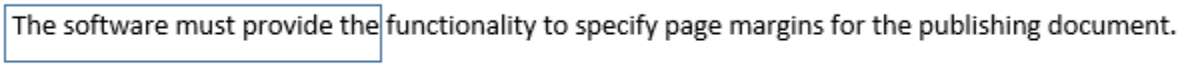


Figure 2.2: An example of windows size 5

In order to build the word co-occurence matrix for a document with a window size of W, we slide a window of size W from the beginning of the document to the end of the document,

one word at a time. As the window moves, we define the target word t to be the word immediately following the window. We define $CoVal(w, t, W)$ to be 0 for each word w in the document which is not in the window. To create the co-occurrence matrix for a window of size W, we list each word in the document in order as the label for each row and as the label for each column. We let each row label represent a fixed target word t, and define each entry in the row to be the co-occurrence value of its column label w with respect to t. In other words, the entry located at row t and column w is computed by CoVal(w, t, W). Table 2.1 shows an example word co-occurrence matrix for the sentence "You can also listen to the music and adjust the volume" with a window size of 5. Let's consider the window starting at the second word, which is the phrase "can also listen to the". In this case, t would be the word right after this phrase, which would be "music". If we let w be "you", then w is not within the window. Therefore, $CoVal$(w,t,W) = 0. However, for if w is "listen", then according to the above formula, $CoVal$(w,t,W) = 5 − 2 = 3. While building the word co-occurrence matrix for the document, we define the co-occurrence vector V of t to be the with column label t concatenated to the row with label t in the co-occurrence matrix. For a target word t in an $N \times N$ matrix, t's co-occurrence vector size is $N+N = 2N$. For the above example, with t equal to "music", the row [0, 1, 2, 3, 4, 5, 0, 0, 0, 0] has label t, and column [0, 0, 0, 0 ,0, 3, 0, 5, 4, 2] has label t. Therefore, its co-occurrence vector is [0, 1, 2, 3, 4, 5, 0, 0, 0, 0, 0,0, 0, 0 ,0, 3, 0, 5, 4, 2].

Table 2.1: Semantic Matrix for Sentence "You can also listen to the music and adjust the volume"

	you	can	also	listen	to	the	music	and	adjust	volume
you	0	0	0	0	0	0	0	0	0	0
can	5	0	0	0	0	0	0	0	0	0
also	4	5	0	0	0	0	0	0	0	0
listen	3	4	5	0	0	0	0	0	0	0
to	2	3	4	5	0	0	0	0	0	0
the	1	2	3	5	7	0	3	4	5	0
music	0	1	2	3	4	5	0	0	0	0
and	0	0	1	2	3	4	5	0	0	0
adjust	0	0	0	1	2	3	4	5	0	0
volume	0	0	0	0	1	5	2	3	4	0

The semantic similarity between two words can be measured by the distance between

the words' two vectors. The original HAL method used the Euclidean distance to compute the similarity between two words. In this study, the cosine distance is used to compute the similarity. The range of cosine distance is from 0 to 1. The cosine similarity is close to 1, when the two words are very similar to each other.

$$CosineSimilarity(w_1, w_2) = \frac{V_{w_1} \cdot V_{w_2}}{\|V_{w_1}\|\|V_{w_2}\|}$$

where V_w is the vector for word w.

In the Table 2.1, the cosine similarity between words "music" and "volume" is 0.38. The word co-occurrence vector for the word "volume" is $[0, 0, 0, 0, 1, 5, 2, 3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ and the word co-occurrence vector for the word "music" is $[0, 1, 2, 3, 4, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 5, 4, 2]$. So the cosine similarity between two words is

$$CosineSimilarity(\text{"volumn"}, \text{" music"}) = \frac{1 \times 4 + 5 \times 5}{\sqrt{55} \times \sqrt{109}} = 0.38$$

There are lots of words only appearing once. This would make the matrix very sparse. When processing a corpus, the input parameter "word frequency" F, is given to decide at least how many times a word w in the corpus should occurr in order to be included in the matrix. An appropriate F would reduce the size of the matrix. When w has a high frequency but is less meaningful, it will be put into a stop list. For example, the words "to", "the", "a","an" are examples of such words. The stop word list will be eliminated during the building of the matrix. Different forms of the same word might appear in one corpus. For example, buy, bought, buys. These words could be considered as one in the matrix. Before building the matrix, lemmatization is done on the corpus: that is, each word is changed to its base form. This could also reduce the size of matrix. A note again about window size: As mentioned, window size defines a range before and after the target word. The larger the window size is, the more words would be included in the window. But if the size is too large, the relation of two co-occurrence words might not be that tight. If the window size is too small, some useful word relations will be omitted.

Chapter 3

Experiment and Evaluation

3.1 Evalution Methods

There is no standard way to evaluate computational measures of semantic similarity. Part of this research was to determine one such way. One simple way is to compare the measurement of semantic similarity with human ratings and see how well its ratings correlate with human ratings of the same word pair. Usually a gold standard word set is built and the words are from one or several thesauri, like WordNet, Roget's, etc. Miller and Charles [17] created a sample of 30 pairs of nouns rated by 38 undergraduate students. Resnik replicated the experiment with the same set of words. A baseline for evaluation was built when a correlation was done between Resnik's replication and Miller and Charles' original set [19].

Another evaluation method is to use the methods to solve the standardized TOEFL (Test of English as a Foreign English) synonyms questions. Each of TOEFL synonym questions consists of a question word and four candidates. The test taker is supposed to pick one of the four candidate words which is most similar to the question word. Landauer and Dumais [5] first compiled and used 80 of these questions, which have been frequently used as an evaluation benchmark.

All the methods listed here use existing thesauri of different types. For this type of methods, Muller states that "Comparing to already existing thesauri is a debatable means when automatic construction is supposed to complement an existing one, or when a specific domain is targeted." [18] This thesis focuses on software requirements documents, and each requirements document might be in a different domain. Directly referring to an existing,

generic thesaurus is not likely to meet this study's goal. So I took a different approach. Inspired by the experiment design of Kawai, Yoshikawa, et al, I planned to manually create a set of targeted synonyms for each document and to manually add these synonyms into the documents that were tested. Another synonym selection method is to manually review the documents and select synonym word pairs as targeted pairs. The next section shows which one serves the purpose better.

3.2 Experiment Preparation

3.2.1 Experiment Synonym Preparation

To start the experiments, I first tried to manually identify the synonyms in a small-sized requirements document, which contains 60 sentences and 1021 words. Before identifying the synonymous word pairs, three steps were needed to process the document. First, a program was run to extract all the distinct words from the document. Second, a stop list was provided to remove noise words from the list of distinct words. Finally, lemmatization was done on the word list. At the end, the words in the list were in their base forms, and duplicated words were removed. Based on the clean distinct word list, all the possible word pair combinations were generated. I recorded the time for reviewing the word pairs, as well as the time to cross off the false positives.

At the end, there were 200 words left in the list. The words could provide 19,900 possible word pair combinations. The 19,900 word pairs were to be manually analyzed to cross off those false positives. The author manually reviewed the 560 word pairs, which took about 10 minutes. Assuming the review speed is constant, the estimated time for reviewing all the word pairs would be about 6 hours! The assumption may not be true since usually the reviewer gets tired and her speed would be slower as the work progresses. The real total time of reviewing would probably be more than the estimated time. Another thing worth noting in the small experiment is that the number of unique words in a document may not grow significantly with increasing the document size since the same words may be used more frequently.

This small exercise gives a reference point for computation: Usually a software requirements document is at least thousands of sentences long and a lot of distinct words. When there are 500 hundred distinct words, there will be 124,750 word pairs. The total time of reviewing that would be more than one day, with all the possible human errors due to fatigue. Considering that impact, I decided to provide a list of synonyms, and add

them to the requirements document. The perecentage of known synonyms not found in the document gives an estimate of how many of the orignal symonyms remain unfound in the document.

3.2.2 Preparation of the software requirements document for the experiment: an RFP

I first chose a standard software requirements document, an RFP, for the experiment. The RFP used in this experiment was the Request for Proposal Document for Desktop Publishing Software that is shown in Appendix A.1. It is a medium sized RFP, and contains about 400 sentences and 761 distinct words. From the calculation from the Section 3.2.1, it's hard to manually identify all the synonyms in the document. So I added a set of selected synonyms to the documents, by replacing some of words in the document with their synonyms that is shown in Appendix A.1.

Since the frequency of words may influence the difficulty of synonym identification, I chose to replace words with different frequencies. Generally, the more frequently a word occurs in the document, the easier the word will be identified. In the RFP, 10 words were selected to represent high, medium, and low frequency words. In Table 3.1, the left column shows the original words in the RFP. The middle column shows the replacing synonyms, and the right column shows the number of occurrences of the original words in the RFP.

The position of synonym replacement may also influence the result. So I chose the positions carefully: Each word to be replaced was randomly chosen from all the positions that word occurs. For example, "worksheet" occurs 10 times in the document. All the line numbers are collected first. Then a program was run to select 5 positions randomly to replace "worksheet" by "spreadsheet". This would eliminate placement bias.

After getting the RFP documents prepared, the eight methods were run on the document. The six WordNet methods were applied using the Python Natural Language Toolkit (NLTK). NLTK is a leading platform providing interfaces over 50 corpus and lexical resources. A WordNet corpus reader in the NLTK provides six semantic similarity measurements. With the help of the WordNet corpus reader, the similarity of two words can be easily measured. Before calculating the semantic similarity, all of the distinct words were extracted from the RFP document. Stop words were also removed from the distinct word list. Then all of the words were set to their basic forms. After these preparation steps, the semantic similarity of all pairs of senses of every pairs of words in the word list was computed and for each pair of words, the maximum value computed for the sense pairs was taken as the pair's semantic similarity value.

Table 3.1: Word replacement

Pair	Orignal Words	Replacement Word	Frequency
1	funtionality	feature	179
2	retain	keep	42
3	specify	set	62
4	create	add	53
5	single	individual	37
6	display	show	17
7	edit	change	28
8	worksheet	spreadsheet	10
9	text	content	54
10	modify	change	14

The SWordNet method was applied using the source code of the program. I was able to get the SWordNet method source code from the authors. The SWordNet method is implemented with Perl. The source code has four functions: to get the idf of each word, to group the sentences into clusters, to compute the semantic similarity of synonym pair candidates, and to remove the duplicated word pairs and rank the word pairs. There are four important parameters in the program. The parameter "threshold" sets the similarity value that determines which word pairs are included in the returned results. The parameter "longest" defines the maximum number of words that can be in a sentence. The parameter "shortest" sets the minimum length of a sentence that can be compared with other sentences. The paramter "gap" defines two sentences' difference, which should be less than or equal to the gap. Each of these four paramters is reflected in the returned results, and each's settings can be varied among different documents.

The word co-occurrence method is implemented in Python. Two classes are designed to store the information of the RFP documents. The Matrix class is used to store the position of each word. Any word's row and column could be looked up. The class Word was designed to store a word's information, like its string. When running the program, the window size and word frequency threshold were passed to the program as parameters. The RFP document was passed on as input, and the output was the word-pair list sorted from the highest semantic similarity value to the lowest semantic similarity value.

3.2.3 Experiment Result and the Analysis

Different methods are typically evaluated by comparing their precision and recall of the processed synonyms. In this experiment, attention was given to recall instead of to precision, because recall is harder to achieve and more critical. To compare the performance of different methods in achieving desired recall rates, I followed the research of Sridhara et al. on this topic [23]. According to this study, the key value to compute to determine the effectiveness of a method M is the number of word pairs from a document D that M applied to D must return in order that 100% of the targeted synonym pairs planted in D to be among the returned word pairs. Among a collection of mehtods, the method with the smallest key value is considered the best. For example, if method A can get all targeted synonyms within 3,000 returned word pairs, but method B needs to have 30,000 returned words pairs to get the same targeted synonyms, then method A is better than method B. Theoretically, if a method returns all the possible word pair combinations from a document, the method could achive 100% recall, but this would not ease the analyst's work. This study not only evaluates if a method can achieve the desired recall, but also compares the number of returned pairs needed for each method to achiveve the recall. For ease of the analysis, the word pairs are sorted by the similarity value. I then measure how many word pairs returned contain 25%, 50%, 75% and 100% of the targeted synonyms. For example, to get 25% recall, the PATH method needs to return 276 word pairs.

Table 3.2: Number of returned results for each level recall

Method	25%	50%	75%	100%
PATH	276	902	53,234	110,024
LCH	1,099	1,522	39,710	114,033
WUP	71	13,403	55,731	84,378
JCN	900	53,232	106,411	146,070
RES	5,401	43,389	50,429	146,070
LIN	674	7,236	103,894	146,070
HAL	5	73	231	2,456
SWordNet	840	1,898	-	-

The first discovery is that the word co-occurrence vector based method (HAL) has the best performance. With all of the WordNet based methods, WUP did the best. However, it needed to return 71 word pairs to get 25% recall, while the word co-occurrence method needed to return only 5 word pairs to get the same recall. To get 100% recall, the word

co-occurrence based method needed to return only 2,456 word pairs. That is dramatically smaller than for all of the rest.

The experiment indicates that it could take about 50 minutes for an analyst to review two thousand word pairs. Three WordNet based methods (WUP, LCH, and PATH) had returned around 110,000 word pairs to achieve 100% recall. This was almost 50 times more than that of the word co-occurrence method. The other three methods in the WordNet category couldn't identify all the semantically related word pairs from the targeted pairs. The SwordNet method has a better performance than the methods from the WordNet category for 25% and 50% recall, but it could not identify all the targeted word pairs in the RFP document.

The second discovery is that WordNet-based methods scored some of the synonyms in the target pairs with a "0". For example, both the LIN and the JCN methods considered the word pair ("functionality", "feature") as non-related words. Another word pair ("worksheet", "spreadsheet") is a pair of synonyms in the requirements document, but they got very a low or 0 score by some of the WordNet based methods. This shows that some words in a specific domain may not be considered as synonyms in generic English.

The third discovery is about the characteristics of the SWordNet method. The method was originally applied to the software source code to find synonyms in inline comments. When applying the method in the experiments to an RFP document, the following situations were discovered:

When I set the program parameters "longest" = 10 and "shortest" = 5, as they were used for processing inline comments [28], most of synonyms added to the RFP could not be identified. For example, the word pair ("specify"," set") appears in two sentences:

The software should provide the feature to specify spell check attributes.

The software must provide the functionality to set fill attributes for the frame.

In these two sentences, "specify" and "set" are synonyms. But the SwordNet method discarded the word pair ("specify spell check attributes", "set fill attributes for the frame"), since the phrases' lengths are more than 2 words. The SWordNet method removes all phrases with more than 3 words. The SwordNet method performs better when the two sentences are in the format: aXbYc, where a, b and c represent the same part in the two sentences, and X, Y represent different parts, each of whose length is less than 3. For example:

The software should provide the feature to specify spell check attributes.

The software should provide the functionality to set spell check attributes.

Here "specify" and "set" can be easily extracted as synonyms.

3.3 Evaluation of the Experiment Results with a User Manual

A user manual can be used as a software requirements specification [3]. The iPhone user manual is taken as another requirements document to analyze. This user manual contains around 3,200 sentences and 3,000 distinct words. In preparing the user manual for the experiments, setting up the experiment, and conducting the experiment, I basically followed the same procedures and steps as for the RFP.

However, unlike the experiment above, artificial synonyms were not added into the document. Instead, 16 pairs of synonyms were identified manually from the document as the targeted pairs. Table 3.3 shows the targeted word pairs identified in the document.

As expected, the experiement showed that the co-occurrence vector based method gave a better results. To achieve the 50% recall, the co-occurrence vector method needed to return around 2,000 word pairs. Among all other methods, even the best would need to return 10,000 word pairs for 50% recall, which is 5 times more than that for the co-occurrence method. To get 100% recall, the word co-occurrence vector method returns 30,000 word pairs, which is only 1% of the entire word pair set. For WordNet-based methods to get 100% recall, they need to return the whole set, which contains around 200,000 word pairs. SwordNet method still can't achieve 100% recall. Only 50% of the targeted word pairs were identified by the SwordNet method. However, it should be pointed out that the SWordNet method makes the word pairs with its program. That makes it more sensitive to format of the document.

3.4 Experiment with Different Parameters in the Word Co-occurence Vector Method

This experiment is to study how the two parameters, window size and word frequency, influence the results of the word co-occurrence vector method. Several combinations of

Table 3.3: Synonyms in iPhone user manual

Pair	Word1	Word2
1	select	choose
2	information	info
3	touch	tap
4	photo	image
5	switch	change
6	view	browse
7	delete	remove
8	passcode	password
9	press	tap
10	restore	download
11	slide	swipe
12	paired	connected
13	organize	manage
14	headphones	headsets
15	ringer	ringtone
16	unpaired	disconnected

Table 3.4: Number of returned results for 100% recall

Method	100%
PATH	215,421
LCH	220,451
WUP	198,714
JCN	220,451
RES	220,451
LIN	220,451
HAL	30,134
SWordNet	Can achieve only 50% of the recall

different window sizes and word frequencies were evaluated on the desktop publishing RFP. For each combination, the experiment examined how many word pairs the methods

have to return in order to achieve 100% recall.

Table 3.5: Number of returned results for 100% recall for different parameters

Windows Size	Word Frequency	Number of Returned Results for 100% recall
1	1	7,802
	3	3,200
	10	Can achieve only 70% recall
5	1	5,997
	3	2,456
	10	Can achieve only 80% recall
10	1	5,655
	3	3,528
	10	Can achieve only 80% recall

From the experiment results in Table 3.5, the data shows that when the window size is 5 and the word frequency is 3, the method needs the fewest number of word pairs to achieve 100% recall. As the word frequency parameter increases, the number of word pairs needed decreases. However, there are some interesting patterns: 1) If each synonym pair from the targeted pairs occurs fewer than 10 times in the document, and the word frequency parameter is set to be 10, some word pairs will not be identified, and the method will not achieve 100% recall. This means if the word frequency parameter is set too high, the size of word co-occurrence matrix will decrease. Then some words with lower frequency will not be identified. But if the word frequency parameter is set too low, the matrix will be too sparse. There will be a lot of 0 values in the matrix, which will influence the calculation of the similarity values. The window size parameter controls how many words will be considered as the context of the target word. If the window size parameter is too small, there will not be enough context information collected. On the other hand, if the window size parameter is too large, some words will not have very tight relations with the target word, and the similarity value won't be accurate.

3.5 Threats and Further Considerations

In the second experiment, I manually added the synonyms in the RFP. The word pairs were selected without consideration of any characteristics of the eight methods. The selection of

the targeted synonym pairs focused on the semantic relations of the works in the software domain even though the word pairs may not be considered as synonyms in general English. However, the selection of the domain-specific words was still based on the author's personal interpretation of the meaning of the words. Secondly, even though the experiment was designed to make the word replacement random, the replacement itself may have been influenced by the author's personal factors. Third, there were hundreds of thousands of returned results, but there may be just a few synonymous word pairs in the returned results. It is not feasible to manually review all of the returned results to identify all synonymous word pairs. There may be some synonym pairs other than the word pairs in the targeted synonym pairs among the returned results.

There are three other facts that may have influenced the experiment's results. First, in the second experiment, there are 10 word pairs selected as targeted word pairs. Choosing a larger number of word pairs, e.g., 100, would make the results and conclusion much stronger, and reduce the risk of experimental failure due to a small sample size. Second, the word pairs were chosen manually in different frequency ranges. The results may be different if a different set of words are chosen from each word frequency range. If the word pairs are selected randomly in each frequency range instead of by my choice, the results may be different. Third, as mentioned before, the domain-specific words selected are based on the author's personal interpretation of the meaning of the words and the understanding of the domain. Some words are not highly technical, such as "change". If more domain specific words are included in the targeted word pairs, the methods targeting the software domain, such as SWordNet, may show better performance.

3.5.1 Experimental Materials

To allow the reader to conduct replications or extensions of the experiment described in this Chapter, all the artifacts used to conduct the experiment can be found at: https://cs.uwaterloo.ca/~dberry/FTP_SITE/students.theses/XiaoYeLan.thesis/Experiments/.

Chapter 4

Conclusion

As the previous chapters have demonstrated, due to the nature of the domain specific content, and the preferred 100% recall of the synonyms as the performance target, Most NLP methods probably are not suitable for extracting synonyms from a software requirements document. The experiments with eight methods, including six WordNet-based methods and two algorithmic methods, showed that the word co-occurrence vector method gave the consistently best performance. The word co-occurrence vector method achieved 100% recall with the smallest number of word pairs. WordNet-based methods, either needed a lot more word pairs to achieve the 100% recall, or missed some of the word pairs in a software specific domain. An experiment was also set up to examine how the parameters could influence the results with the word co-occurrence vector method.

However, this thesis evaluated only a basic part of the word co-occurrence vector method. In the future, the variants of co-occurrence based method could be evaluated for software requirements documents. Some of the potential topics include, for example, considering different word weights when computing the similarity value, changing the way to calculate semantic similarity in the word co-occurrence vector based method, etc.

APPENDICES

Appendix A

Request for Proposal Documents Used in the Experiemnt

This request for proposal (RFP) Document gives the requirements for a desktop publishing program. The first section contains all the mandatory requirements and the second section contains point-rated requirements in which each requirement can be graded from 0 to the max points. The highlighted words are the words replacing the orignal words in the documents.

A.1 RFP for Desktop Publishing Software with Replaced Synonyms

1. Each instance of the software proposed must be provided in both English and French or bilingual (English and French).

2. The software must provide the functionality to assemble graphics, images, and text to produce presseady files.

3. The software must provide the functionality to collect multiple files (chapters) into a single volume.

4. The software must provide the functionality to create a new publishing document from a template.

5. The software must provide the functionality to: input files into the publishing document, including .rtf, .pdf, and .txt file formats; input image files, including .eps, .jpg, .gif, .png, .bmp, .tiff, and .pdf file formats, into the publishing document; and output from the publishing document to multiple file formats, including .html .pdf, .ps, and .xml file formats.

6. The software must provide the functionality to output the publishing document as a PDF/X-1a:2001 (ISO 15930-1:2001) document, a webeady PDF document, and as an interactive PDF document that includes bookmarks, hyperlinks, and cross references.

7. The software must provide the functionality to: print the publishing document, including single page, entire document, thumbnails, signatures and spreads; print multiple pages of the publishing document, including print sequential pages and print non-sequential pages; specify settings for crop marks, including weight and position; print publishing document printer's marks, including crop marks, page border, and page size; save document output settings, including printer's marks, embedded fonts, page size, position, orientation, and bleed; and reuse and share document output settings.

8. The software must provide the feature to apply crop marks and registration marks to the publishing document.

9. The software must provide the functionality to specify bleed in the publishing document.

10. The software must provide the functionality to: show overlapping colours, including overprint and knockout, in the publishing document; hide overlapping colours; show and hide bleed, crop area, margins and frames in the publishing document; and display invisible formatting characters, including soft returns, hard returns, page breaks, tabs, and document section breaks.

11. The software must provide the feature to: show and hide text flow (text thread) in the publishing document; auto-detect next page number and previous page number in text flow in the publishing document; and change the properties of a frame without having to re-create the frame and then adjust text flow if affected by a change to the frame.

12. The software must provide the feature to: create both single page and multi-page publishing documents; create pages of different sizes, including custom width and custom height, in a single publishing document; create pages of different orientations

in a single publishing document; and create and display facing pages, including facing pages that allow for different page formatting on the right and left sides.

13. The software must provide the functionality to: specify page margins for the publishing document; set custom margins, including by location coordinates and unit of measure; allow different margins on individual publishing document pages; save page settings with the publishing document, including page size, position, orientation, and margins; modify the publishing document page numbering format, including having main body page numbers formatted differently from appendix page numbers; and insert, delete, copy, and move pages in the publishing document.

14. The software must provide the functionality to divide the publishing document into sections such that document sections allow different formatting, page numbering, section header, section trailer, and assign pages to document sections.

15. The software must provide the functionality to: create a frame in the publishing document that can house different types of objects, including text and images; create multiple frames on a single page; allow frame content to span multiple pages; set frame height, width, position, including coordinates and unit of measure, and rotation, including degree of rotation; set frame shape, including rectangle, square, and custom; and edit the frame shape at the node level.

16. The software must provide the feature to auto-detect page margins, including create a full-page frame, when importing objects into the publishing document.

17. The software must provide the functionality to: set fill attributes for the frame; set stroke attributes for frame; and specify special effects for frame, including shadow, glow, feathering, bevel and emboss.

18. The software must provide the functionality to: set frame inset, including spacing between content and the frame border; specify the vertical alignment within a frame, including top, middle, and bottom; set the horizontal alignment within a frame, including left, centre, and right; specify the location of an object (image or graphi within a frame, including by coordinates, unit of measure to three decimal places; and provide a notification if text content is too large for the frame.

19. The software must provide the feature to: set text wrap for a frame, including shape and distance; specify the alignment of text wrap, including relative to spine, relative to column, and relative to page; and set path spacing (the distance between a path and wrapped text).

20. The software must provide the functionality to: crop objects to fit the shape and size of a frame, including cropping such that parts of object may not be visible; fit a frame to content and fit content to a frame; and keep the aspect ratio of an object while fitting the object to a frame and a frame to the object.

21. The software must provide the functionality to keep existing object paths and alpha channels such that the software will allow text wrap around clipping paths.

22. The software must provide the feature to: lock a frame; unlock a frame; drag a frame; anchor a frame; align and distribute frames, including align to top, align to left, align to bottom, and align to right; and set a reference point of the frame (for example, top left hand corner of frame or centre of fram and then set an action relative to the reference point, including rotate, align, position, flip, and distribute.

23. The software must provide the feature to: create editable layers in the publishing document; add multiple types of objects to a layer, including text, graphics, and images; create multiple frames on a single publishing document layer; name layers; order layers; lock layers; unlock layers; hide layers; and show layers.

24. The software must provide the feature to create non-printing layers (layers that display but do not print); for example die cuts and guidelines.

25. The software must provide the feature to: create new objects, including graphics, lines, and shapes; edit objects, including copy, move, delete, resize, group, ungroup, lock and unlock; modify object location and direction, including drag and drop, scale, rotate, tilt, flip, and distribute; and align objects, including relative to each other, relative to the frame, and relative to the page.

26. The software must provide the feature to: set object properties, including text wrap, and place on top of another object; add nodes to an object; delete nodes from an object; send to back (the object is placed behind other objects); and bring to front (the object is placed in front of other objects).
    ”

27. The software must provide the feature to: set the shape of an object's stroke; set the alignment of an object's stroke relative to the object segment, including inside the segment, outside the segment, and centreed on the segment; set stroke attributes for an object, including colour, gradient, pattern, line style, and thickness; set fill attributes for an object, including colour, gradient, and pattern; and specify the shape of object corners.”

28. The software must provide the feature to: show colours of different colour models, including CMYK and RGB in the publishing document; apply CMYK colour to an individual object; apply RGB colour to an individual object; and convert colour to greyscale.

29. The software must provide the feature to show and print colour separations, including CMYK colour separations and spot, colour separations, from the publishing document.

30. The software must provide the feature to apply spot colours to individual objects.

31. The software must provide the feature to select colours from software-supplied colour libraries, including from a Pantone Matching System colour library.

32. The software must provide the feature to: add additional colour libraries, including libraries provided by printing services; apply a specific colour from a colour library, including Pantone Matching System, to an individual object; and search by colour identifier within the colour library.

33. The software must provide the feature to: keep colours from source objects in the publishing document, including Pantone Matching System colours; create, save and re-use customized collection of colours (swatches) for use with the publishing document; grab a specific colour value from an object, for example, using an eye-dropper tool; and set swatch colour intensity, from 0

34. The software must provide the feature to add colours to swatches while importing objects if a colour does not already exist in the publishing document.

35. The software must provide the feature to display ink coverage (the percentage of ink for each plat for the publishing document.

36. The software must provide the feature to: apply transparency to an object; preview transparency; and set transparency (opacity and blending mod).

37. The software must provide the feature to: create and edit text frames in the publishing document; import text into a text frame in the publishing document; paste text from the clipboard into the publishing document; provide multiple text behaviours, including paragraph text and text that follows a path; and convert text objects to vector objects, including Bezier curves

38. The software must provide the feature to set transparency settings, including opacity and blending mode, for text.

39. The software must provide the feature to insert special characters, including French characters and symbols.

40. The software must provide the feature to: apply formatting to selected text; format characters, including size, font, colour, superscript, subscript, underline, bold, italic, small caps, all caps, drop caps, kerning, tracking and proper case; format text paragraphs, including horizontal and vertical direction, space above, space below, indenting, first line, hanging, offset, baseline shift, rule above, rule below, and leading; align text horizontally, including left, right, centre, and justified; and align text vertically, including top, middle, bottom, and justified.

41. The software must provide the feature to turn auto-hyphenation on and off.

42. The software must provide the feature to set pagination, including page break before, page break after, column break before, column break after, and "keep with".

43. The software must provide the functionality to use PostScript Type I, TrueType, and OpenType fonts.

44. The software must provide the feature to: spell check the publishing document; create a custom spelling dictionary; add terms from publishing document to the spelling dictionary; and edit the spelling dictionary.

45. The software must provide the feature to specify the language of the spell check, including English and French.

46. The software must provide the feature to: insert columns in a frame and specify column layout in a frame, including number of columns, width of columns, and distance between columns (gutter); insert multiple columns on a page in different frames; create multiple columns of different widths on a single page; and adjust column size after content has been added (such that text will reflow).

47. The software must provide the functionality to: create tables in the publishing document; specify table rows and table columns; auto-distribute table columns (columns will be evenly space in the publishing document; create table header rows, table footer rows, table header columns, and table footer columns (such that table headers and table footers allow for different formatting from rest of table; and create, edit, and format table captions.

48. The software must provide the feature to: merge selected cells and split selected cells; resize a table, including specify text reflow; insert rows, including row above and row below, in the table; insert columns, including column left and column right, in the table; and dynamically append new row during table input.

49. The software must provide the functionality to orient text within selected cell, including 90, 180, and 270 degrees, in the table.

50. The software must provide the feature to select table elements, including a single cell, multiple cells, a single row, multiple rows, a single column, and multiple columns.

51. The software must provide the feature to: format tables and table text, including font, colour, and size; align a table horizontally to a frame; specify table spacing, including inset and margins; specify table borders, including apply individually, apply to entire table, apply manually, and apply using a style; and specify table shading, including apply individually, apply to entire table, apply manually, and apply using a style.

52. The software must provide the feature to link an object in the publishing document to a source object (the document contains references to source objects and not an embedded copy of the object).

53. The software must provide the feature to: link to images, including .eps, .jpg, .gif, .png, .bmp, and .tiff file formats; show the properties of the linked source object; open the source of a linked image and graphic from within the publishing document; and relink an object to a different source object.

54. The software must provide the functionality to embed an image object, including .eps, .jpg, .gif, .png, .bmp, and .tiff file formats, into the publishing document (embedded objects display in the publishing document but contain no reference to a source object).

55. The software must provide the functionality to issue a warning if linked source objects have been changed or are missing.

56. The software must provide the feature to specify how linked objects are updated, including prompt before updating, update without prompting, and do not update.

57. The software must provide the functionality to: generate a table of contents based on styles in the publishing document; and provide choices for generating a table of contents, including the table of contents style.

58. The software must provide the feature to apply and modify the style of a table of contents, and to propagate changes made to the style to the table of contents.

59. The software must provide the functionality to: generate an index based on terms identified in the publishing document; select terms for inclusion in the index, including identify a term as a topic, and identify term as a subtopic; provide choices for generating an index, including using the index style; and modify the index style such that changes are propagated to the index.

60. The software must provide the feature to specify the relationship of a selected term to an index topic (for example, for "Canada Revenue Agency", see "Government Department").

61. The software must provide the feature to: create footers and specify footnote position, including at the bottom of the page, and at the end of a document section; provide choices for numbering and format of footnotes, including alphanumeric numbering; and modify a footer style such that changes are propagated to the footnotes.

62. The software must provide the feature to: create interactive crosseferences in the publishing document; format crosseference source content; and auto-update a crosseference if the source modifies.

63. The software must provide the feature to create and customize crosseference styles, and apply crosseference styles to crosseference sources.

64. The software must provide the feature to provide and edit variables, including page markers, document section markers, date information, and path information, for insertion into the publishing document, and propagate changes throughout the publishing document if a variable is changed.

65. The software must provide the feature to: store re-usable objects, including images, graphics, animation, and text in a publishing document library; name the publishing library; name each publishing library object; sort the publishing library by object name; drag and drop to and from publishing libraries, including dragging and dropping graphics objects and text objects either from the publishing library to the working file, or from the working file to the publishing library; and search the publishing library.

66. The software must provide the feature to propagate changes to the publishing document when a publishing library source object is changed.

67. The software must provide the feature to: select all objects displayed; add to and subtract from the existing selection without having to re-select; select characters, single words, multiple words, single paragraphs, multiple paragraphs, and the entire frame content; and copy selected objects to the clipboard and paste content from the clipboard into the publishing document, including graphics, images, and text, while keeping aspect ratios and pixel dimensions of pasted objects.

68. The software must provide the feature to: create new re-usable styles that can be named and shared; apply a style to the selected frame, including single frame and multiple frames; allow manual override of style formatting; and modify styles, including formatting attributes and style name.

69. The software must provide the feature to propagate changes made to a style within the publishing document while keeping manual attributes (for example, in a heading with an applied style of bold, a single word is manually changed to italic; when the style is modified the single word with the manual attribute will remain itali.

70. The software must provide the feature to: create re-usable character styles, including character formatting, character colour, underline, and strikethrough; apply a character style to selected text, including a single character, multiple characters, a single word, multiple words, and all content in a frame; create re-usable paragraph styles, including text formatting, tabs, text size, font, colour, alignment, spacing, position, hyphenation, justification, and pagination; apply a paragraph style to selected text, including the current paragraph and multiple paragraphs; and use styles to sequentially number paragraphs.

71. The software must provide the feature to: create re-usable list styles, including list numbering, bullet symbol, list format, and list numbering format; apply list styles to the selected text, including single list, and multiple level list; create re-usable cell styles, including fill, stroke, padding, spacing, horizontal alignment, vertical alignment, text style, and indenting; and apply a cell style, including apply to a single cell, multiple cells, selected cells, single row, multiple rows, single columns, multiple columns, table header row, and table footer row.

72. The software must provide the feature to create re-usable table styles, including fill, stroke, table header rows, table footer rows, table header columns, table footer columns, padding, spacing, horizontal alignment, alternating fill, and alternating stroke, and apply a table style to an entire table.

73. The software must provide the feature to create a child style (the child style inherits its attributes from another parent style, but the child style can still be edited and propagate attribute changes to child styles if the parent style is edited.

74. The software must provide the functionality to: provide master pages to standardize and lock layout and common content for child pages in the publishing document; create new master pages, including margins, footers, headers, page numbers, common graphics, single-sided, left and right pages, and spreads, for use with the publishing document; apply different master pages to individual pages in the publishing document; and insert variables on master pages, including page number, total pages, filename, and date, that will appear on every child page.

75. The software must provide the functionality to propagate changes made to the master page to the child pages.

76. The software must provide the feature to create, edit, save and re-use templates (templates must include styles, master pages, colour management, swatches, page size, margins, orientation, layers, and common objects) that can be used as the source for a new publishing document..

77. The software must provide the feature to: assemble individual desktop publishing files as chapters in a book such that chapters remain as independent files; provide choices for chapters, including start a chapter on a new page, start a new chapter on the same page, and start a chapter on an odd page; number the book, including chapters, paragraphs, tables, figures, and pages; specify a numbering format for the book, including having multiple numbering formats within the book; restart numbering within the book; auto-update book numbering as content changes; and update book-linked reference numbering as content changes, including tables of contents, footnotes, crosseferences, and index numbering.

78. The software must provide the functionality to: generate multiple tables of contents, indexes, and crosseferences spanning chapters for a single book; spell check across the book (across multiple files); print multiple chapters; and print an entire book.

79. The software must provide the functionality to create a mail merge.

80. The software must provide the feature to: search text; search for special characters, including space, tab, hard return, soft return, and non-breaking space; choose a search action, including find next, find all, replace current, and replace all; and

navigate search results, including next result and previous result, such that focus is moved to the location of the search results and the search result is highlighted.

81. The software must provide the feature to produce PDF documents based on PDF/x standards, including PDF/x-1a, and specify the output PDF version, including 1.3 and higher.

82. The software must provide the feature to: finalize the publishing document; identify changes made to linked objects, including linked graphics and linked images; resolve issues; and collect all related files, including colours, fonts, linked graphics, linked images) into folder hierarchy with native

83. The software must provide the feature to keep metadata from a source file in the publishing document.

84. The software must provide the feature to: output a publishing document as an accessible PDF document, including description tags, reading order, and alternate text; keep existing alternate text for tables, lists, footnotes, and hyperlinks when converting publishing document to a PDF document; and output a publishing document to accessible HTML format, including alternate text and reading order.

85. The software must provide the feature to keep existing alternate text when a publishing document is converted to HTML.

86. The software must provide the feature to add alternate text to images and graphics in the publishing document.

87. The software must provide the feature to keep the existing colour settings, including RGB, CMYK, greyscale, and Duotone, of a PDF document when that PDF is embedded in the publishing document.

88. The software must provide the functionality to keep the existing content and layout when a native publishing document is converted to a PDF document.

89. The software must provide the feature to open, show and edit a publishing document created using a Mac OS version of the software.

90. The software must provide the feature to import from native MS Word file formats, including .doc and .docx file formats, into the publishing document.

91. The software must provide the feature to keep the content, including text, graphics, charts, text flows, formatting characters, and images, and format when a native MS Word document, including .doc, .docx, .dot, and .dotx file formats, is imported into the publishing document.

92. The software must provide the feature to add new incoming MS Word document text styles to the publishing document when a native MS Word document is imported into the publishing document.

93. The software must provide the feature to map incoming text styles from the MS Word document to existing styles in the publishing document when a native MS Word document is imported into the publishing document.

94. The software must provide the feature to keep MS Word table content and formatting when a native MS Word table is copied and pasted into the publishing document.

95. The software must provide the feature to keep existing MS Word colour values and linked references, including tables of contents, indexes, footnotes, and crosseferences when a native MS Word document is imported into the publishing document.

96. The software must provide the feature to copy and paste a selection from a native MS Word document into the publishing document.

97. The software must provide the feature to keep content and format, including data, charts and colour values, when a native MS Excel worksheet, including .xls, and .xlsx file formats, is imported into the publishing document.

98. The software must provide the feature to copy and paste a selection from a native MS Excel worksheet into the publishing document.

99. The software must run on SSC/CRA's computing infrastructure as defined in Appendix 2.

100. All desktop software must conform to the CRA desktop standard technical requirements as defined in Appendix 3.

POINT RATED REQUIREMENTS

1. The software should provide the feature to include multimedia content in the publishing document.

2. The software should provide the feature to import content containing linked references, including tables of contents, footnotes, and cross references, into the publishing document and keep linked references.

3. The software should provide the feature to specify the page imposition of the publishing document.

4. The software should provide the feature to show individual publishing document pages at different rotations.

5. The software should provide the feature to copy and paste a frame and frame content into the publishing document.

6. The software should provide the feature to modify content and the content type housed in frame without having to re-create the frame.

7. The software should provide the feature to apply and manipulate frame corner effects both individually and as a group.

8. The software should provide the feature to adjust the angle of the frame corner.

9. The software should provide the feature to generate image captions from image metadata in the publishing document.

10. The software should provide the functionality to specify special effects for objects.

11. The software should provide the functionality to specify colour settings, including RGB colour and CMYK colour for the publishing document.

12. The software should provide the feature to slice objects with transparency applied, including opacity and blending mode, into individual colours (to allow easier interpretation for a Raster Image Processor (RIP)).

13. The software should provide the functionality to create and edit text directly and view the name of an applied style in a dedicated editor.

14. The software should provide the feature to import text while keeping existing manual attributes (formatting that has been manually applied to text).

15. The software should provide the feature to import text while removing existing manual attributes (strip manual attributes applied to text).

16. The software should provide the feature to specify methods for pasting text.

17. The software should provide the feature to substitute fonts if a font is unavailable.

18. The software should provide the feature to specify font substitution, including notification of the substituted font and navigate to font substitution.

19. The software should provide the functionality to keep the original font reference if a font is unavailable (the font reference is not overidden).

20. The software should provide the feature to specify spell check attributes, including auto-correct.

21. The software should provide the functionality to specify application preferences for spell check.

22. The software should provide the feature to convert a text selection to a table.

23. The software should provide the functionality to auto-number tables.

24. The software should provide the feature to select table numbering formats from a software-provided list.

25. The software should provide the feature to input content containing tables into the publishing document and keep table content, table structure, and table styles.

26. The software should provide the functionality to apply table formatting manually and using styles.

27. The software should provide the functionality to initiate collaboration and review of publishing documents via a web conferencing session (real-time review including a shared desktop and chat functions).

28. The software should provide the functionality to display all references to a single source object in the publishing document.

29. The software should provide the functionality to create custom variables, including fields and symbols.

30. The software should provide the functionality to create re-usable named graphics styles, including fill, stroke, content, and special effects.

31. The software should provide the functionality to apply graphic styles, including applying to the selected graphic and applying to multiple graphics.

32. The software should provide the functionality to create re-usable conditional styles (the style is applied based on a condition).

33. The software should provide the functionality to create keyboard shortcut keys mapped to a style (invoking the keyboard shortcut key applies the styl.

34. The software should provide the functionality to disassociate individual objects on an individual child page from the master page such that the disassociated object can be managed manually and will not be affected by changes to the master.

35. The software should provide the functionality to keep font references in the publishing document template.

36. The software should provide the functionality to synchronize styles and colours from multiple chapters into a single book, including identifying the primary publishing style sheet to be applied to entire book.

37. The software should provide the functionality to search accessible alternate text in the publishing document.

38. The software should provide the functionality to track and indicate content and formatting changes, including who made the change, when the change was made, and what was modified.

39. The software should provide the functionality to perform a pre-flight check of the publishing

40. The software should provide the functionality to define and customize pre-flight settings.

41. The software should provide the functionality to display pre-flight results with crossreferences such that a user can navigate to highlighted sections in the pre-flight results.

42. The software should provide the functionality to save and re-use pre-flight settings.

43. The software should provide the functionality to import 3rd party pre-flight settings.

44. The software should provide the functionality to access classes and methods for publishing document objects.

45. The software should provide the functionality to keep the metadata from the publishing document in the output PDF document.

46. The software should provide the functionality to allow CRA to use custom players to play audio and video content in the publishing document (the software does not enforce using any specific player).

47. The software should provide the functionality to keep existing metadata when a PDF document is embedded into the publishing document.

48. The software should provide the functionality to retain existing interactivity, including hyperlinks, when a native publishing document is converted to a PDF document.

49. The software should provide the functionality to retain existing variable values when a native publishing document is converted to a PDF document.

50. The software should provide the functionality to create PDF navigation from a native publishing document table of contents when converting a native publishing document to a PDF document.

51. The software should provide the functionality to retain existing linked references, including footers, tables of contents, tables of tables, tables of figures, indexes, and crosseferences, when a native publishing document is converted to a PDF document.

52. The software should provide the functionality to retain existing colour settings, including RGB, CMYK, greyscale, and Duotone, when a native publishing document object is converted to a PDF document.

53. The software should provide the functionality to retain existing metadata when a native publishing document is converted to a PDF document.

54. The software should provide the functionality to retain existing font references when a native publishing document is converted to a PDF document.

55. The software should provide the functionality to retain multimedia content when a native publishing

56. The software should provide the functionality to retain existing accessibility elements when a native publishing document is converted to a PDF document.

57. The software should provide the functionality to retain existing publishing document text flow when a native publishing document is converted to a PDF document.

58. The software should provide the functionality to retain existing comments when a native publishing document is converted to a PDF document.

59. The software should provide the functionality to retain existing editable text when a native publishing document is converted to a PDF document.

60. The software should provide the functionality to assemble graphics, images, and text suitable for viewing with mobile devices.

61. The software should provide the functionality to save for mobile devices, including optimizing size, download time, quality, and colours.

62. The software should provide the functionality to assemble graphics, images, and text to produce a publishing document for distribution to e-book readers.

63. The software should provide the functionality to provide choices for importing a native MS Word document into the publishing document, including import table of contents, import footers, import endnotes, import graphics, and import page breaks.

64. The software should provide the functionality to retain existing MS Word text styles when a native MS Word document is imported into the publishing document.

65. The software should provide the functionality to override existing publishing document text styles with MS Word text styles when a native MS Word document is imported into the publishing document.

66. The software should provide the functionality to override existing MS Word text with publishing document text styles when a native MS Word document is imported into the publishing document.

67. The software should provide the functionality to save the style mapping from a native MS Word document to the publishing document for re-use.

68. The software should provide the functionality to retain MS Word table content and formatting when a native MS Word document is imported into the publishing document.

69. The software should provide the functionality to retain existing MS Word variable values when a native MS Word document is imported into the publishing document.

70. The software should provide the functionality to retain existing MS Word alternate text when a native MS Word document is imported into the publishing document.

71. The software should provide the functionality to retain existing MS Word metadata when a native MS Word document is imported into the publishing document.

72. The software should provide the functionality to retain existing MS Word hyperlinks when a native MS Word document is imported into the publishing document.

73. The software should provide the functionality to retain existing MS Word markups when a native MS Word document is imported into the publishing document.

74. The software should provide the functionality to import a native MS Excel spreadsheet into the publishing document as a table.

75. The software should provide the functionality to import a native MS Excel multi-sheet workbook into the publishing document as multiple tables.

76. The software should provide the functionality to import native MS Excel template file formats, including .xlt and .xltx file formats, into the publishing document.

77. The software should provide the functionality to import a range of cells from a native MS Excel worksheet into the publishing document as a table.

78. The software should provide the functionality to choose how a native MS Excel spreadsheet is imported into the publishing document, including import as a formatted table, import as unformatted text, and import as tabbed text.

79. The software should provide the functionality to retain existing MS Excel variable values when a native MS Excel spreadsheet is imported into the publishing document.

80. The software should provide the functionality to retain existing MS Excel metadata when a native MS Excel spreadsheet is imported into the publishing document.

81. The software should provide the functionality to retain existing MS Excel hyperlinks when a native MS Excel worksheet is imported into the publishing document.

82. The software should provide the functionality to log an audit trail of events and activities that is accessible by the user and includes the following: the date and time; identification of the user, machine or process initiating the event/activity; and a description of the event/activity.

References

[1] Vincenzo Ambriola and Vincenzo Gervasi. Processing natural language requirements. In *Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference*, pages 36–45. IEEE, 1997.

[2] Daniel Berry, Ricardo Gacitua, Pete Sawyer, and Sri Fatimah Tjong. The case for dumb requirements engineering tools. In *Requirements Engineering: Foundation for Software Quality*, pages 211–217. Springer, 2012.

[3] Daniel M Berry, Khuzaima Daudjee, Jing Dong, Igor Fainchtein, Maria Augusta Nelson, Torsten Nelson, and Lihua Ou. Users manual as a requirements specification: case studies. *Requirements Engineering*, 9(1):67–82, 2004.

[4] Alexander Budanitsky and Graeme Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources*, volume 2, 2001.

[5] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[6] Olivier Ferret. Testing semantic similarity measures for extracting synonyms from a corpus. In *LREC*, 2010.

[7] Leah Goldin and Daniel M Berry. Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Engineering*, 4(4):375–412, 1997.

[8] Zellig S Harris. Distributional structure. *Word*, 1954.

[9] Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

[10] Yasushi Kawai, Tomohiro Yoshikawa, Takeshi Furuhashi, Eiji Hirao, Ayako Kuno, and Tomohisa Gotoh. A study on extraction method of synonyms in specification documents. In *Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, pages 321–324. IEEE, 2012.

[11] Claudia Leacock, George A Miller, and Martin Chodorow. Using corpus statistics and WordNet relations for sense identification. *Computational Linguistics*, 24(1):147–165, 1998.

[12] Dekang Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304, 1998.

[13] Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996.

[14] L. Mich, M. Franch, and P. Novi Inverardi. Market research for requirements analysis using linguistic tools. *Requirements Engineering Journal*, 9(1):40–56, 2004.

[15] George A Miller. WordNet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[16] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database*. *International journal of lexicography*, 3(4):235–244, 1990.

[17] George A Miller and Walter G Charles. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28, 1991.

[18] Philippe Muller, Nabil Hathout, and Bruno Gaume. Synonym extraction using a semantic distance on a dictionary. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, pages 65–72. Association for Computational Linguistics, 2006.

[19] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995.

[20] Pierre Senellart and Vincent D Blondel. Automatic discovery of similar words. In *Survey of Text Mining*. Citeseer, 2003.

[21] Subhash K Shinde, Varunakshi Bhojane, and Pranita Mahajan. Nlp based object oriented analysis and design from requirement specification. *International Journal of Computer Applications*, 47(21):30–34, 2012.

[22] Andrey Simanovsky and Alexander Ulanov. Mining text patterns for synonyms extraction. In *DEXA Workshops*, pages 473–477, 2011.

[23] Giriprasad Sridhara, Emily Hill, Lori Pollock, and K Vijay-Shanker. Identifying word relations in software: A comparative study of semantic similarity tools. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 123–132. IEEE, 2008.

[24] Lonneke Van der Plas and Jörg Tiedemann. Finding synonyms using automatic word alignment and measures of distributional similarity. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 866–873. Association for Computational Linguistics, 2006.

[25] Tong Wang and Graeme Hirst. Exploring patterns in dictionary definitions for synonym extraction. *Natural Language Engineering*, 18(03):313–342, 2012.

[26] Wenbo Wang, Christopher Thomas, Amit Sheth, and Victor Chan. Pattern-based synonym and antonym extraction. In *Proceedings of the 48th Annual Southeast Regional Conference*, page 64. ACM, 2010.

[27] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[28] Jinqiu Yang and Lin Tan. Swordnet: Inferring semantically related words from software context. *Empirical Software Engineering*, pages 1–31, 2013.