

Identifying Word Relations in Software: A Comparative Study of Semantic Similarity Tools *

Giriprasad Sridhara, Emily Hill, Lori Pollock and K. Vijay-Shanker
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716
{gsridhar, hill, pollock, vijay}@cis.udel.edu

Abstract

Modern software systems are typically large and complex, making comprehension of these systems extremely difficult. Experienced programmers comprehend code by seamlessly processing synonyms and other word relations. Thus, we believe that automated comprehension and software tools can be significantly improved by leveraging word relations in software. In this paper, we perform a comparative study of six state of the art, English-based semantic similarity techniques and evaluate their effectiveness on words from the comments and identifiers in software. Our results suggest that applying English-based semantic similarity techniques to software without any customization could be detrimental to the performance of the client software tools. We propose strategies to customize the existing semantic similarity techniques to software, and describe how various program comprehension tools can benefit from word relation information.

1. Introduction

In general, humans seamlessly process synonyms and other word relations. For example, a developer skimming for code related to “removing an item from a shopping cart” understands that the method `deleteCartItem(Item)` is relevant, even though it uses the synonym ‘delete’ rather than ‘remove’. However, the large size and complexity of today’s software systems necessitates the development of automatic tools to help the software engineer in program comprehension and thus complete software maintenance tasks effectively and efficiently.

We believe that many of the tools to aid program comprehension can reap significant benefits in effectiveness, either directly or indirectly, by capturing and exploiting word

relations in the code. We are interested particularly in six general types of word relations. *Synonyms* are words that have similar meaning and can often be used interchangeably [23], such as ‘remove’ and ‘delete’. *Antonyms* are words that have opposite meanings [23], such as ‘valid’ and ‘invalid’. *Hypernyms and hyponyms* represent words that have a more general or specific meaning, respectively [23]. For example, ‘vehicle’ is a hypernym of ‘car’, and ‘car’ is a hyponym of ‘vehicle’. *Meronyms and holonyms* represent part-of and whole relations, respectively [23]. For example, ‘leaf’ is a meronym of ‘tree’ and ‘tree’ is a holonym of ‘leaf’. Although the notion of *semantic similarity* has been defined to subsume the above relations [29], in the remainder of this paper, we assume that the idea of semantic similarity does not include antonym relations. Finally, we are also interested in *topically similar* words which are word pairs that belong to the same topic. For example, online auction management software would contain words related to online auctions, such as sniping and bidding.

We believe that these kinds of word relations can aid automated software analysis and program comprehension tools. Broadly defined, search tools use queries and similarity measures on software artifacts (source code, documentation, maintenance requests, version control logs, etc.) to facilitate a particular software engineering or program comprehension task. Tools which use natural language or keyword queries and matching can benefit by expanding queries and adding related words to textual artifact representations. For example, synonyms are especially useful in overcoming vocabulary mismatches between the query and software artifacts, especially with regard to the concept assignment problem [8].

Several software maintenance tools have been developed that use some notion of synonyms. Shepherd et al. [31] developed FindConcept, a tool that expands search queries with synonyms to locate concerns more accurately in code. FindConcept obtains synonyms from WordNet [11], a lex-

*This work is supported by NSF Grant No. CCF-0702401.

ical database of word relations that was manually constructed for English text. Although FindConcept demonstrates that synonyms can improve performance in software tools, there has been no formal evaluation demonstrating that synonyms derived for general English text, such as the synonyms found in WordNet, are effective in finding the synonyms used in software.

A second maintenance tool that uses synonyms is iComment [33]. iComment is a tool that automatically expands queries with similar topic words to resolve inconsistencies between comments and code and therefore helps to automatically locate bugs. Their lexical database of word relations was automatically mined from the comments of two large programs. Their approach of automatically mining topic relations from code tacitly assumes that existing tools for semantic similarity on English text are insufficient when applied to software.

Thus, in this paper, we investigate whether semantically similar words used in software are also deemed to be similar by semantic similarity techniques developed for English. The answer will give us important insight into whether existing similarity techniques will be effective when used directly in software tools or whether they should be customized for software to maximize effectiveness. In this context, we note that Etzkorn et al. [10] concluded that comments form a sub-language of English, for example, in the restricted usage of verbs.

To answer the above open question, we performed a comparative study of the effectiveness of six well known semantic similarity techniques developed for the English language [16], when applied to the software domain. We analyzed how each technique ranked a set of human-identified related word pairs from a suite of 12 open source Java programs from different application domains. Our results demonstrate that off-the-shelf English-based semantic similarity tools can report many spurious word relations for words that are not in fact related. These spurious results could reduce the effectiveness of the client software tool.

The major contributions of this paper are:

- Results from quantitative and qualitative studies of the effectiveness of six well known English-based semantic similarity techniques, applied to the problem of determining word relations in software
- Analysis and discussion of the limitations and opportunities for customizing semantic similarity tools specific to the software domain
- Descriptions of how various client software tools could benefit from accurate word relation information

2. Tools to Benefit from Word Relation Clues

Many tools could potentially benefit from word relation information provided by semantic similarity techniques. In

this section, we briefly describe how tools typically used directly or indirectly for improving program comprehension could benefit from this information.

Concern Location. Semantic similarity can be used to automatically identify concerns by expanding a user-specified query, and this can be added retroactively to improve existing concern location techniques [31]. Concern location, or the concept assignment problem, is made more difficult as a result of vocabulary mismatches when translating between a high level idea, or concern, and that concern’s implementation in the code [8]. Previous work has shown that expanding a query with WordNet synonyms, in conjunction with a sophisticated location algorithm, can locate concerns more effectively than traditional searching mechanisms [31].

Asymmetric word relationships—such as hypernyms, hyponyms, meronyms, and holonyms—can also be leveraged to take advantage of the fact that the query may be at a different conceptual level than the code. Thus, *selectively* expanding the query with more specific hyponyms and meronyms could improve results. We note that blind expansion of the query may lead to poor results. For example, adding ‘increment’ for the query term ‘add’ is not appropriate in all contexts (e.g., for the query “add item to list”) but may be useful in other contexts (e.g., “add one to score”). In addition, when comments are used as part of the concern location process, the query can be expanded with more general hypernyms and holonyms, because comments are written at a more abstract or higher level than the code.

Program Exploration. Based on previous work [13], we hypothesize that synonyms and other semantic relationships can be used to improve automatically generated recommendations for program exploration from a starting point. For example, when exploring code related to “movie listings,” it is important to recognize meronyms of ‘movie listing’, such as ‘date’, ‘time’, ‘location’, ‘theater’, and ‘show’. Methods that handle movie listing data may not contain either ‘movie’ or ‘listing’, but many of its meronyms. Existing approaches [28, 30] can take advantage of this information by incorporating a textual component in addition to structural information [13].

Refactoring. Synonyms and hypernyms can be used to locate and rank code clones [5, 17], a common refactoring task. Clones that have similar structure as well as similar identifiers indicate more tightly coupled clones. For example, synonyms could be incorporated into a code fragment distance analysis [5]. Rather than just comparing types and values for distance, the content of the identifiers could also be compared by using semantic similarity.

Refactoring can also benefit from antonyms [32]. Methods performing opposite functionality within the same class can be good candidates for refactoring into aspects. For example, the methods `lock()` and `unlock()` may be called

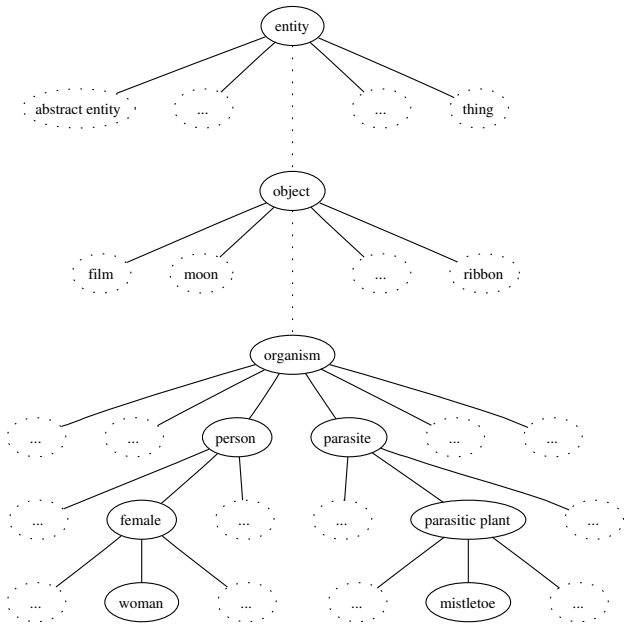


Figure 1. Sample WordNet *is-a* hierarchy for the nouns ‘woman’ and ‘mistletoe’.

before and after an operation occurs—textbook examples of good refactoring candidates in aspect-oriented programming [18].

Repository-based Program Comprehension Aids. Like concern location, traceability between software artifacts [1] can leverage semantic similarity. In addition, existing work in recommending relevant code examples or other artifacts from open source repositories [4, 14] could benefit from semantic similarity. Rather than just using the structure and words of code fragments, the artifacts could be augmented with topic words. For example, when searching for artifacts related to “CVS repository browsing” in a GUI, semantically relevant words such as ‘history’, ‘branch’, ‘hierarchy’, and ‘navigate’ could be added to improve recommendation results. Semantic similarity is also useful when analyzing maintenance requests, such as maintenance request assignment [2]. In addition, tools for finding differences between versions of software [3] can also benefit by using synonymous relations between words—for example, when the change between two versions involves renaming a method to a synonymous word, e.g., renaming `add()` to `append()`.

3. Text-based Semantic Similarity Tools

This paper examines the key open question of whether the successful technology developed for semantic similarity

measurement in English text can be directly applied to the software domain. We begin with an overview of the state of the art in the English text domain.

Many semantic similarity detection tools are based on WordNet [11], a lexical database of English word and sense relations. A *sense* is a particular meaning of a word. For each sense of a particular word, WordNet provides the *synset*, a list of synonyms for that sense. For example, for one sense of the word ‘bank’, WordNet specifies the following synset: ‘depository financial institution’, ‘bank’, ‘banking concern’, ‘banking company’. The definition, or *gloss*, of this synset states ‘a financial institution that accepts deposits and channels the money into lending activities’. Thus, each synset is a representation of a particular concept. WordNet includes glosses and senses for words that are nouns, verbs, adjectives and adverbs.

WordNet also connects the concepts in a hierarchy that models hypernym, hyponym, meronym and holonym relations for nouns and verbs. No such relation hierarchy exists in WordNet for adverbs or adjectives. We can visualize this hierarchy of word relations as a graph with the concepts (nodes) being connected via the different relations (edges), as in Figure 1. As we navigate from the root to the leaves, we move from abstract to concrete concepts.

The hypernymy relation in WordNet is very useful for relating words, especially for nouns. Hypernymy can also be viewed as an *is-a* relation. For example, a ‘woman’ is a ‘female’. The *is-a* relation beginning with ‘woman’ is shown in Figure 1. Notice that by following the *is-a* hierarchy, more general *is-a* relations can be inferred. For example, a ‘woman’ is also a ‘person’. Likewise, both a ‘woman’ and a ‘moon’ are objects. Thus, it is possible to derive both abstract and concrete hypernymy relations from this WordNet hierarchy.

3.1 Current Approaches

Due to space constraints, we present only a synopsis of the existing approaches to detecting semantic similarity of words in English text. The interested reader should refer to the detailed descriptions in [16, 26].

In this paper, we investigated six publicly available, well known semantic similarity techniques that perform well on English text [9, 16, 26]. The techniques and the category to which they belong are shown in Table 1. In general, these strategies are based on WordNet and use the hierarchical structure of WordNet to produce a score that indicates the similarity relation between words, with higher scores indicating higher similarity. The description of each technique category follows.

Path Based. The simplest semantic similarity tools for English text measure the length of the shortest path between two words in the WordNet *is-a* hierarchy and work under

Approach	Author(s)	Category
LCH	Leacock and Chadrow [20]	Path based
WUP	Wu and Palmer [34]	Path based
JCN	Jiang and Conrath [15]	Information content based
RES	Resnik [27]	Information content based
LIN	Lin [22]	Information content based
LESK	Banerjee, et al. [7]	Gloss based

Table 1. Semantic similarity techniques.

the intuition that a word w is more similar to words with shorter paths from w . For example, in Figure 1, ‘woman’ is more semantically similar to ‘person’ (shortest path length is 2), than to ‘mistletoe’ (shortest path length is 6).

This path based approach assumes that each edge in the word hierarchy represents a uniform distance in terms of semantic similarity. However, intuitively, edges at the leaves (e.g., edge between ‘woman’ and ‘female’) are closer in semantic similarity than edges at the top of the hierarchy, such as between ‘thing’ and ‘entity’. Both Leacock and Chadrow [20] and Wu and Palmer [34] add a correcting factor to take into account the depth in the hierarchy.

Information Content Based. The similarity measures of Resnik [27], Jiang and Conrath [15] and Lin [22] use the structure of WordNet along with probabilistic information derived from an English language corpus (i.e., the frequency of occurrence of a sense). From these probability distributions, the information content of the words and WordNet synset classes are determined. In the *is-a* hierarchy, the information content associated with a word increases as you navigate down the hierarchy (i.e., as the concepts become more concrete). The Resnik similarity measure defines the similarity between two words as the amount of information shared by the two words, which is measured by determining the information at the Lowest Common Subsumer (LCS) of the two words in the *is-a* hierarchy. The LCS of two words is the lowest node in the *is-a* hierarchy which is a hypernym of both words. In the Figure 1, the LCS of ‘person’ and ‘parasite’ (i.e., ‘organism’) is lower in the *is-a* hierarchy (thus has more information) than the LCS of ‘person’ and ‘moon’ (i.e., ‘object’). Thus, ‘person’ and ‘parasite’ are said to be more similar than ‘person’ and ‘moon’. The Lin similarity measure augments the Resnik measure to take into account the differences in information due to the distance between the LCS of different pairs of words in the hierarchy. More differences in information between two words implies that the words are less similar. The Jiang and Conrath similarity measure is very close to the Lin measure, although it was developed independently.

Gloss Based. Another approach to detecting semantic similarity [6, 7, 21] is based on the intuition that related senses of words are described using many of the same words. For

example, the similar words ‘car’ and ‘automobile’ both contain the word *vehicle* in their glosses. Because glosses can be short or incomplete, the glosses are first augmented with glosses of words that are related to the original words being compared via the various relations, particularly hypernyms and hyponyms. The extended gloss overlap, or LESK, technique [7] then uses this gloss information as follows: For every n consecutive words that overlap between the glosses of the pair of words under comparison, a score of n^2 is added to their similarity measure. The justification is based on the belief that a longer overlap is more indicative of similarity and hence should be favored by giving a higher score.

3.2 Comparing the tools on English text

Only the LESK measure is capable of comparing across parts of speech. However, when the glosses are short, there may not be enough overlapping words between the glosses, which can hinder the gloss based techniques [26]. Since adjectives and adverbs are not connected in any *is-a* hierarchy, the path based and information content based tools cannot be used for adjectives and adverbs. Several research groups have performed comparative studies of these tools on English text [9, 26]. The studies led the researchers to conclude that the information content based similarity technique proposed by Jiang and Conrath [15] and the extended gloss based measure [7] perform the best among the tools studied. Given the differences between word usage within the structure of programs and word usage in English text, it is not clear that these conclusions hold for the software domain.

4. Quantitative Evaluation

We investigate the effectiveness of state of the art semantic similarity tools when applied to software.

Independent Variable. The independent variable in our study is the English-based semantic similarity technique. As described in Section 3.1, we evaluated six similarity tools [16] shown in Table 1. We used the publicly available Perl module `WordNet::Similarity`¹ to implement the techniques. Each technique takes a pair of words such as `<remove, delete>` and returns a score representing the similarity between the words, where higher scores indicate more word similarity.

Subject Word Pairs. The subjects of our study are word pairs from programs. We collected 12 open source Java programs (shown in Table 2). We then developed a set of word pairs by randomly selecting 125 methods which had comments before the method declarations or within the method body.

¹<http://www.d.umn.edu/~tpederse/similarity.html>

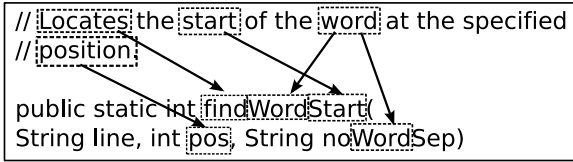


Figure 2. Example mappings for word pairs.

Name	SLOC	Methods	Methods with Comments
MegaMek	147,404	9,256	4,716
iReport	74,392	5,672	2,972
freemind	70,435	6,110	1,992
ganttproject	43,106	4,941	1,333
prefuse	40,427	2,494	1,939
JHotDraw	38,866	4,267	2,255
jajuk	30,847	2,137	1,812
javaHMO	23,797	1,787	445
jbidwatcher	23,179	1,918	681
JRobin	19,469	1,913	908
Dguitar	13,322	1,211	768
PlanetaMessenger	11,231	1,142	1,013

Table 2. Java programs used in the study.

We created a *gold set* of semantically similar word pairs used in software. Prior to investigating the similarity tools used in the study, one of the authors manually examined the same 125 methods, and manually mapped pairs containing a word from the comment and a word that he believed was related to it in the corresponding method body. We believe that comment to code word mappings are a reliable source of software vocabulary mismatches, with different words expressing the same concept. For example, Figure 2 contains the semantically similar pair <locates, find>, the abbreviation pair <position, pos>, and the remaining 3 pairs are exact matches.

By analyzing these 125 methods, we identified 1,621 word pairs of related words. We removed the pairs which were exact (or stemmed) matches, or had an abbreviation. The remaining 60 of the 1,621 were semantically similar pairs and form our gold set.

We also created our *total set* of word pairs by mapping all possible mappings of words occurring in a comment with words occurring in the associated method body, thus, *total set* consists of every comment–method word pair from the 125 methods. For the 125 commented methods that we considered, we identified 40,798 pairs to become *total set*. This set of word pairs is used as an approximation of false positives for the evaluation, as discussed below.

Dependent Variables and Measures. The dependent variable in our evaluation is the effectiveness of a similarity tool when applied to software. We measure the effectiveness in

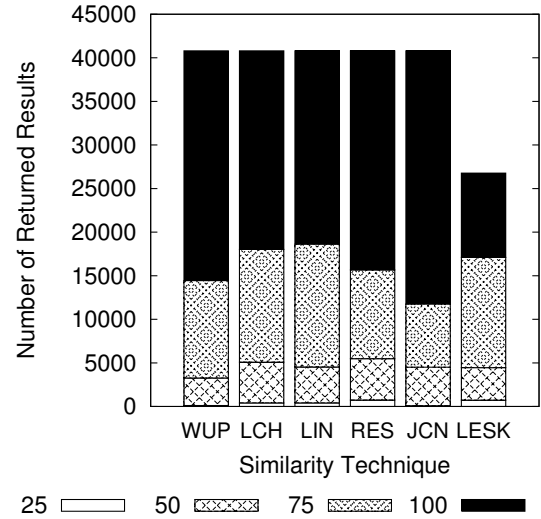


Figure 3. Number of returned results for 25%, 50%, 75%, and 100% recall.

terms of the number of returned results, to gain insight into the false positives, (i.e., unrelated pairs returned). Analyzing the false positives is important because there may be a trade off in the number of gold set pairs that are scored as similar and the total number of pairs scored as similar. For example, a very naive technique might determine that every pair is semantically similar, resulting in 60 true positives and as many as 40,738 false positives.

Unfortunately, determining the number of false positives would require manually verifying whether thousands of word pairs are semantically similar. To approximate the number of false positives, we instead use the number of returned results. Although there may be a few pairs that are true positives, we believe the majority are false positives.

Thus, we analyze the number of returned results at various levels of recall, where recall is interpreted as the percent of true positives. Specifically, the scored pairs from a given tool are rank ordered in decreasing score (i.e., decreasing similarity). Consider returned results for 25% recall which would be any 15 pairs from the gold set. The *returned results* at 25% recall is then the number of pairs returned with scores greater than or equal to the score of the 15th pair in the gold set to appear in the rank order.

Threats to Validity. It is possible that other semantic similarity tools exist which could do better on software than the tools investigated in this study. To minimize this threat, we selected six publicly available, well known semantic similarity tools that are known to perform well on English text.

We randomly selected 125 commented methods across 12 open source programs for our gold set. There are only

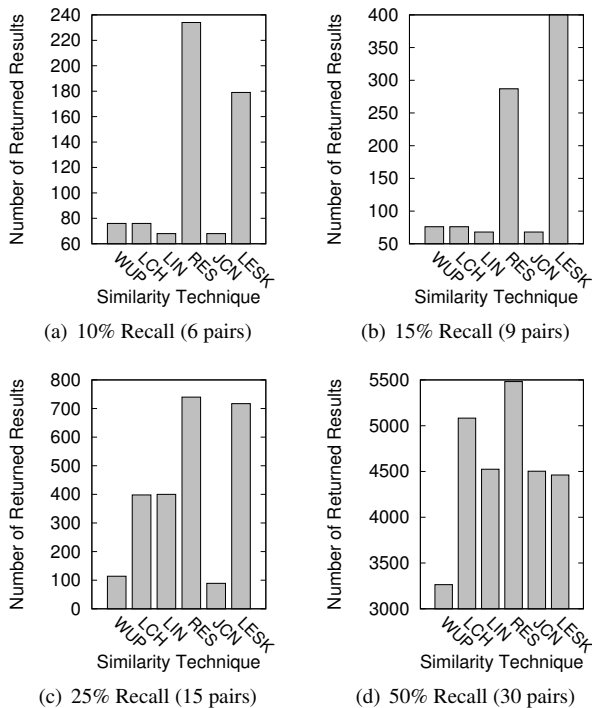


Figure 4. Number of returned results at 10%, 15%, 25% and 50% recall.

sixty pairs of semantically similar words in the gold set for this study, and therefore the results may not generalize to pairs outside of the study. We note that existing evaluation studies of semantic similarity tools for English have obtained meaningful results from only 30 word pairs [9]. Since all of our programs are Java, our results may not generalize to other programming languages.

Due to the number of returned results (over 40,000 for most tools), it was not feasible to manually inspect all the results to determine the exact number of false positives for each tool. Thus, we use the number of returned results as an approximation of the number of false positives. We believe that true positives outside the gold set are few, and that false positives dominate the returned results. To reduce this threat, we verified the returned result set for the tools WUP, JCN, LCH and LIN at 25% recall. For these tools, the number of returned results at 25% recall is not very large, ranging from 90 to 400. We manually verified each entry in these sets and concluded that our hypothesis was correct, i.e., that false positives do indeed dominate the returned results. Lastly, all tools score antonyms as semantically similar, although we include no antonyms in the gold set.

Results and Analysis. Although there were differences

among the tools at lower levels of recall, all tools returned orders of magnitude more pairs than the size of the gold set at most recall levels, and only distinguished themselves at 25% or lower recall. All six semantic similarity tools returned above 4000 word pairs in order to obtain 50% recall of the 60-pair gold set, and up to 40,000 word pairs at 100% recall! Figure 3 depicts a stacked bar chart showing the number of returned results for each technique at 25%, 50%, 75%, and 100% recall. Higher bars imply more unrelated words being returned along with the desired recall of the gold set. For example, LESK returns 27,000 total pairs at scores higher or equal to the scores of the gold set pairs. Thus, LESK determines that 27,000 pairs have similar or more semantic similarity than semantic similarity of the 60 gold set pairs.

By looking at the 100% recall levels, we see that all techniques except LESK determine the whole total set to be as semantically similar as the gold set. However five of our 60 gold set pairs are words that are different parts of speech, and these techniques are not designed to find words of different parts of speech as similar, and thus do not score these pairs. To achieve 100% recall, these techniques have to accept *any input pair* as being semantically similar. At 75% recall,² the returned results from the techniques range between 12,000 and 18,000, about 400 times the desired size of the return results. These results suggest that these tools are good for English text, including pairs that should be considered similar for software, but must also include many unrelated word pairs to obtain semantically similar ones.

It is only at the lower levels of recall, 25% and lower, that the techniques are distinguished from one another. Figure 4 focuses on the number of returned results at 10%, 15%, 25% and 50% recall. Even at 50%, the techniques are similar in effectiveness and still in the 3000-5500 range of returned results. WUP performs slightly better than the others, but is still an order of magnitude (100 times) the desired size of the returned results (30 of 60 pairs). At 25% recall, the techniques can be grouped into three categories. JCN and WUP returned about 7 times more than the desired number of results, LCH and LIN returned about 400 results, and RES and LESK returned about 700 results. The relative performance of the techniques is similar for 10% and 15% recall, with RES and LESK returning much higher number of results than the others. LESK tends to suffer because as described in Section 3, it is the only technique that compares words across parts of speech and this greedy approach causes it to include many more pairs than the other techniques. RES may be performing relatively worse because it treats all senses of a word equally, and this might not work as well in the software domain.

²Since there are 60 pairs in the gold set, at 75% recall, the pairs returned are those with scores greater than or equal to the score of the 45th pair in the gold set to appear in the rank order out of the 60 pairs.

English Synonyms

Word Pair	LESK		JCN		LIN		RES		WUP		LCH	
	Score	RR	Score	RR	Score	RR	Score	RR	Score	RR	Score	RR
Make-Create	2410	8	9593499	68	1	68	7.31	407	1	76	3.33	76
Start-Begin	887	53	9593499	68	1	68	6.98	520	1	76	3.33	76
Determine-Check	474	179	9593499	68	1	68	8.14	234	1	76	3.33	76
Locate-Find	441	206	0.57	151	0.88	149	6.59	731	0.85	575	2.64	169
Display-Show	261	467	12876699	13	1	68	9.20	72	1	76	3.68	16

Software Synonyms

Word Pair	LESK		JCN		LIN		RES		WUP		LCH	
	Score	RR	Score	RR	Score	RR	Score	RR	Score	RR	Score	RR
Write-Save	328	311	9593499	68	1	68	10.09	39	1	76	3.33	76
Assign-Set	181	967	0.18	751	0.66	636	5.59	1516	0.75	2131	2.23	2370
Find-Search	61	7000	0.10	3171	0.51	1376	5.33	1804	0.8	1320	2.30	1399
Notify-Fire	12	23173	0.07	8085	0	18680	0	18627	0.25	18061	1.39	12945
Output-Print	2.23	26574	0.13	1555	0.62	804	2.23	10079	0.97	76	0.73	18604

Table 3. Similarity scores and returned results (RR) for select English and software pairs.

To summarize, even at a low recall (10%, 15%, and 25%), the techniques return 10 times the desired number of results or more. *These results suggest that to accept many word pairs that are used synonymously in software as being semantically similar in English, we need to include a large number of results which can be harmful to the performance of the client software tool.*

5. Qualitative Evaluation

To learn more about the kinds of word pairs that the techniques consider to be semantically similar versus human intuition about which word pairs from software would likely be found by an English-based semantic similarity technique, we conducted a closer examination of individual word pairs with two qualitative studies.

The goal of the first qualitative study was to investigate how the techniques handled software word pairs that humans believed were related in English and word pairs that humans believed were common in software, but not in English text. We hypothesized that the techniques would not perform well for pairs not commonly related in English text, such as the pair <notify, fire>. Two authors inspected the gold set and selected a representative set of 5 pairs that they believed to be used synonymously in English and 5 pairs that they believed to be used synonymously within software (but are not generally considered to be true synonyms in English). To avoid bias when selecting the pairs, the authors had no knowledge of the actual scores assigned to these pairs by any of the techniques. We examined the similarity scores and the number of returned results for the 10 word pairs for each technique.

In contrast, the second qualitative study focused on examining which word pairs the semantic similarity techniques scored as most similar and least similar. We hy-

pothesized that we would see many common software word relations, not similar in English, clustered at the bottom of the scores. We extracted the top 10 scored word pairs and lowest 15 scored word pairs from the scored pairs ranked by each semantic similarity technique. We chose the bottom 15 because all of the techniques except LESK include the 5 gold set pairs that involve different parts of speech as the bottom 5 results, which are never scored by those techniques.

Threats to Validity. In the first qualitative study, we selected 5 pairs to represent typical synonyms in English and in software. The pairs were selected by two authors who had no knowledge of the techniques' output. The pairs were chosen as representatives of their kind: the authors felt that the English pairs would be correctly ranked by semantic similarity techniques derived for English, and the software pairs were considered to be word relations unique to software (and thus would be scored incorrectly by the techniques). It is possible that conclusions drawn from these 5 examples may not generalize to all pairs in our gold set, or all word pairs in general. In the second qualitative study, we only analyzed the top 10 and the bottom 15 scored pairs for each technique. It is possible that different conclusions could be drawn when considering a more extensive set.

Results and Analysis. Table 4 shows the scores assigned by the 6 similarity techniques for 5 pairs of English and software synonyms. As expected, all the techniques scored the English synonyms higher than the software synonyms, except for the pair <write, save> and <output, print> for WUP. The pair <write, save> is given a high score by all the techniques. Unlike the other software synonyms, WordNet contains <write, save> because it is a very common synonym in computer science. However, note that the other synonyms common to software are not included

	JCN	WUP	LCH	LIN	RES	LESK
Top 10	first - start last - end nothing - null display - show write - save determine - check add - append make - create start - begin	nothing - null write - save add - append last - end start - begin first - start make - create display - show determine - check	nothing - null display - show last - end first - start add - append determine - check make - create start - begin write - save	last - end display - show write - save determine - check append - add nothing - null add - append start - begin make - create	write - save display - show remove - clear write - print last - end first - start determine - check sort - compare open - start	make - create start - begin last - end nothing - null first - start determine - check locate - find write - save remove - clear
Bottom 15	remove - delete same - equal remove - cleanup create - new add - plus dumps - export save - output create - clone full - max check - is exceeds - greater test - if check - if make - new create - new begins - init	save - output compare - equal flush - write remove - cleanup add - plus reset - remove store - add dumps - export full - max notify - fire exceeds - greater test - if check - if make - new create - new begins - init	locate - find save - output add - plus remove - cleanup dumps - export add - put remove - delete output - print full - max notify - fire exceeds - greater test - if check - if make - new create - new begins - init	first - start reset - remove create - clone add - create dumps - export store - add eliminate - strip same - equal full - max notify - fire exceeds - greater test - if check - if make - new create - new begins - init	make - create store - equal store - add no - null reset - remove add - create dumps - export compare - equal full - max notify - fire exceeds - greater test - if check - if make - new create - new begins - init	remove - delete flush - write sort - compare no - null delete - cleanup create - clone compare - equal reset - remove full - max notify - fire exceed - greater append - plus dumps - export add - put output - print save - output

Table 4. Top 10 and Bottom 15 gold set word pairs for each semantic similarity technique.

in WordNet, and thus are scored very low by the rest of the techniques. The other exception is <output, print> for WUP. This pair is only higher than WUP's score for the English synonyms <find, locate>, which are scored extremely low by the other techniques in comparison to the remaining English synonyms. Thus, it appears that rather than WUP giving <output, print> a higher score than English synonyms, perhaps it should be questioned whether or not <find, locate> is as common an English synonym as we thought. In fact, as we see in the second qualitative study, LCH scores <find, locate> so low that it occurs in the bottom 15 of 40,000 ranked pairs in the total set.

Table 5 shows the top 10 and the lowest 15 scored pairs for each technique. The most obvious pairs missed by all the tools except LESK, are the 5 lowest pairs of two different parts of speech: <exceed, greater>, <test, if>, <check, if>, <make, new>, and <create, new>. Two other bottom pairs have similar senses but not in all contexts. For example, <same, equal> are not synonyms when speaking of possessions, "I have the *same* computer as you." Likewise, the glosses of <exceeds, greater> are very similar, but do not include the same sense of being larger. There were also two pairs for which one of the words was not even present in WordNet: in <full, max>, max is not linked as an abbreviation for 'maximum', but rather as a street name for an illegal drug; and in <begin, init>, init has no gloss at all.

The remaining pairs of interest in the bottom are common software synonyms. We did not expect the tools to miss pairs like <remove, delete> and <locate, find>,

which appear synonymous to programmers and perhaps lay people alike. However, we were not surprised that the tools missed pairs like <flush, write>, <dump, export>, <notify, fire>, and <output, print>. These pairs are clearly synonyms to programmers, and are not likely to be recognized as such outside the domain of software. Notice the last two were included in our select software synonyms in Table 4.

Lastly, there were some pairs in Table 5 that appear in both the top 10 and bottom 15 for different tools. These include <locate, find> (LCH), <first, start> (LIN), and <make, create> (RES). We believe these pairs confirm our observation of the previous section; namely, that these three path based and information content based tools perform worse than JCN and WUP.

6. Discussion

Applying existing semantic similarity tools to software.

Based on the performance of the techniques on the gold set, JCN and WUP appear to perform better than the others, although all techniques perform poorly at 50% recall and above (just 30 gold set word pairs). The conclusion that JCN and WUP perform well can be drawn by considering the number of results returned by the techniques at a given level of recall as well as by considering the performance on the manually selected software pairs. Although LCH is also a path based technique like WUP, it performs worse than

WUP at all recall levels. Similarly, the other information content based measures, LIN and RES, do not perform as well as JCN.

Previous studies have suggested that LESK and JCN are among the best techniques for English text [26]. However, the superior performance of LESK is not replicated by our study, especially at the lower recall levels. On the other hand, at the highest recall levels, LESK outperforms both WUP and JCN. We hypothesize that LESK's poor performance at lower recall levels is due to the fact that LESK can determine similarity between words of different parts of speech, and thus considers similarity between many more word pairs than the other techniques. This property is also noticeable at higher recall, since LESK is the only tool to reach 100% recall before returning the entire total set.

JCN performs the best of the three information content based techniques. This is consistent with findings on English text [9, 26]. Since these techniques compute their score based on a probability distribution from English text, we expect their performance to deteriorate on a specialized domain such as software. Thus the poor performance of information content based techniques could be due to the fact that the information content score of words for English does not apply for semantically similar words in software.

Customizing semantic similarity techniques for software. Our study suggests that two promising ways to customize existing semantic similarity techniques for the software domain. One way is to augment WordNet with relations specific to software, possibly by mining word relations from software. Recall from the qualitative study that the pair <write, save> was correctly identified as semantically similar by all the techniques. This is because WordNet has been augmented to include some very common software word relations. The techniques' success in scoring <write, save> leads us to believe that augmenting WordNet with word relations specific to software is a promising approach warranting further research.

The second way to improve existing semantic similarity techniques is to improve the word probabilities used by the information content based techniques. Current information content based approaches utilize a probability distribution of words based on English text. Deriving these probabilities from word usage in the comments of software could lead to significant improvements. Even without such domain specialization, JCN performed better than most other techniques. This leads us to believe that JCN with domain-specific information could be the most viable approach to determining semantic similarity for software.

7. Related Work

Our current research is somewhat related to strategies that automatically map comments to code by calculating

a similarity measure between the set of words in both the method and the comment [12, 19]. Fluri et al. use a set-based similarity metric to explore how comments and code evolve over time [12]. Lawrie et al. compute the similarity between comments and the associated code to assess the quality of code [19]. Code containing methods and comments with high similarity is assumed to be of high quality. Their approach uses cosine similarity to calculate the overlap in word usage between comments and methods. Because cosine similarity does not map comment words to method words to calculate similarity, this approach is inadequate to create word pairs for our current experiments.

There is also research that avoids explicitly handling semantic similarity by using an information retrieval (IR) technique that embeds semantic similarity information [24, 25]. Latent Semantic Indexing is an IR technique that uses the co-occurrences of words in documents to discover hidden semantic relations between words. Since the technique is based on co-occurrences of words, the resulting word relations are not guaranteed to be semantically similar. LSI is also dependent on the set of documents (i.e., set of programs) from which the word co-occurrences are observed.

8. Conclusions and Future Work

We investigated the effectiveness of the state of the art similarity techniques when applied to software. Our analysis indicates that at low recall levels (up to 25%), JCN and WUP appear to perform better than other techniques. At higher levels, none of the techniques appear to perform well. In general, the number of returned results can be 10 times greater than the number of desired results, and much more for higher levels of recall. While the use of semantic similarity information is intended to improve the performance of software tools, the large number of returned results could instead be detrimental.

As discussed in Section 6, our study suggests two promising ways to customize existing semantic similarity techniques for the software domain: (1) to augment WordNet with relations specific to software, and (2) improve the word probabilities used by the information content based techniques. We plan to implement these improvements as future work.

References

- [1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10), 2002.
- [2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, 2006.

- [3] T. Apiwattanapong, A. Orso, and M. J. Harrold. A differencing algorithm for object-oriented programs. In *19th IEEE International Conference on Automated Software Engineering*, 2004.
- [4] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a search engine for open source code supporting structure-based search. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, 2006.
- [5] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis. Advanced clone-analysis to support object-oriented system refactoring. In *WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, 2000.
- [6] S. Banerjee and T. Pedersen. An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. In *International Conference on Computational Linguistics and Intelligent Text Processing*, 2002.
- [7] S. Banerjee and T. Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In *International Joint Conference on Artificial Intelligence*, 2003.
- [8] T. J. Biggerstaff, B. G. Mitbander, and D. Webster. The concept assignment problem in program understanding. In *ICSE '93: Proceedings of the 15th International Conference on Software Engineering*, 1993.
- [9] A. Budanitsky and G. Hirst. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics*, 32(1), 2006.
- [10] L. Etzkorn, C. Davis., and L. Bowen. The language of comments in computer software: A sublanguage of English. *Journal of Pragmatics*, 33, 2001.
- [11] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
- [12] B. Fluri, M. Würsch, and H. C. Gall. Do code and comments co-evolve? on the relation between source code and comment changes. In *14th Working Conference on Reverse Engineering (WCRE)*, 2007.
- [13] E. Hill, L. Pollock, and K. Vijay-Shanker. Exploring the neighborhood with Dora to expedite software maintenance. In *22nd IEEE International Conference on Automated Software Engineering (ASE)*, 2007.
- [14] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *27th International Conference on Software Engineering*, 2005.
- [15] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of International Conference on Research in Computational Linguistics (ROCLING X)*, 1997.
- [16] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Second Edition*. Prentice Hall, 2008.
- [17] J. Krinke. Identifying similar code with program dependence graphs. In *Eighth Working Conference on Reverse Engineering (WCRE'01)*, 2001.
- [18] R. Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co., 2003.
- [19] D. J. Lawrie, H. Feild, and D. Binkley. Leveraged quality assessment using information retrieval techniques. In *ICPC '06: Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06)*, 2006.
- [20] C. Leacock and M. S. Chodorow. *Combining local context and WordNet similarity for word sense identification.*, chapter 11. Wordnet: An Electronic Lexical Database. MIT Press, 1998.
- [21] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *SIGDOC '86: Proceedings of the 5th Annual International Conference on Systems Documentation*, 1986.
- [22] D. Lin. An information-theoretic definition of similarity. In *International Conference of Machine Learning*, 1998.
- [23] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
- [24] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, 2003.
- [25] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)*, 2004.
- [26] T. Pedersen, S. Banerjee, and S. Patwardhan. Maximizing semantic relatedness to perform word sense disambiguation. Technical report, University of Minnesota, Duluth, 2005.
- [27] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *International Joint Conference on Artificial Intelligence*, 1995.
- [28] M. P. Robillard. Automatic generation of suggestions for program investigation. In *13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2005.
- [29] G. Ruge. Automatic detection of thesaurus relations for information retrieval applications. In *Foundations of Computer Science: Potential - Theory - Cognition, to Wilfried Brauer on the occasion of his sixtieth birthday*, 1997.
- [30] Z. M. Saul, V. Filkov, P. Devanbu, and C. Bird. Recommending random walks. In *Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2007.
- [31] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *AOSD '07: Proceedings of the 6th International Conference on Aspect-oriented Software Development*, 2007.
- [32] D. Shepherd, L. Pollock, and K. Vijay-Shanker. Case study: supplementing program analysis with natural language analysis to improve a reverse engineering task. In *PASTE '07: Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 2007.
- [33] L. Tan, D. Yuan, G. Krishna, and Y. Zhou. **iComment: Bugs or bad comments?**. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, 2007.
- [34] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, 1994.