Learning Semantically Related Words in Software through Word Embedding

Wangsheng Hu[§], Hongyu Zhang[†], Xiaodong Gu[♯], Dongmei Zhang[†], and Jianjun Zhao[§]
[§]Shanghai Jiaotong University, Shanghai, China
{dao1993, zhao-jj}@sjtu.edu.cn
[†]Microsoft Research, Beijing, China
{honzhang,dongmeiz}@microsoft.com
[♯]The Hong Kong University of Science and Technology, Hong Kong
xguaa@cse.ust.hk

ABSTRACT

Searching for previously written code is important for software maintenance and reuse. A common problem in code search is that the keywords specified in a user query may not match the exact words used in source code - a keyword could be an abbreviation, a synonym, or a homonym of a word in code. Such a vocabulary mismatch problem adversely affects the accuracy of code search results. As pointed by previous work, one major difficulty in solving this problem is that semantically related words used in software practice are often not semantically related in ordinary English. In this paper, we propose SWordMap, a Word Embedding based method for automatically identifying semantically related words used in software. We show that code search accuracy is improved after expanding queries with the semantically related words identified by our approach. Furthermore, our approach outperforms the related approach that leverages the context of words in comments and code.

Keywords

Code search; query expansion; semantic similarity

1. INTRODUCTION

One of the common tasks in software development practice is code search [11, 13, 16, 7, 12]. Developers frequently search for code (including classes, functions, code snippets) for software maintenance (such as refactoring, adding new features, and bug-fixing) and reuse (reusing previously written code that implements similar functionality). Today's code search tools, e.g., Sando [4], Ohloh [3], Kugle [1], and Sourcerer [11], perform code search primarily based on keywords/text matching. In practice these tools have limitations in returning code that a developer wants. One challenge is the vocabulary mismatch problem [6], which occurs when people name the same thing or concept differently.

In code search, vocabulary mismatch means that the keywords specified in a query are different from what are used in source code. For example, a programmer would like to find the method that executes a thread, she may issue a query *execute a thread*, which may miss some correct results such as *Thread.run*. Synonyms/homonym are useful in overcoming the vocabulary mismatches between the query and software artifacts [19]. However, using synonyms/homonym in English is not sufficient for code search as semantically related words in English could be different from semantically related words in software. For the above example, *run* and *execute* have different meanings in English but are synonyms in software. The differences in word usage between software and English also exist in abbreviations. For example, developers may use *IRQ* in source code and *interrupt* in query [25]. It has been observed that expanding a code search query with inappropriate synonyms can return even worse results than the original query [19].

A number of techniques have been proposed to improve code search accuracy by identifying semantically similar words in software [17, 25, 26, 10]. For example, Shepherd et al. [17] extract similar verb-DO (Direct Object) pairs from software projects to identify semantically related words. Yang et al. [26] present SWordNet, which extends Shepherd's technique to identify rPairs (semantic related pairs) in software through text similarity comparison. As an example, if *disable all interrupt sources* and *disable all irq sources* both exists in linux kernel code, *interrupt* and *irq* would be identified as an rPair due to their shared context. Both of the above methods identify semantically related words from source code and comments. The identified semantically related words could be project specific if they are learned from local projects, and could be limited in number if they are learned across projects. Furthermore, SWordNet may generate false positives when the shared context contains many common English words [26]. For example, SWordNet mistakenly considers *match* and *literal* as an rPair from comments *we have a match* and *we have a literal*.

In this paper, we propose SWordMap, a *Word Embedding* based method for learning semantically related words from a large-scale software corpus. Word Embedding [15] is a recently developed technique, which utilizes a neural network model to learn vector presentation of words with the desideratum that words with similar meanings will map to similar vectors. We train a Word Embedding model on 5,913,765 sentences, which are extracted from 708,473 *Stackoverflow* questions. We obtain semantically related words for 19,332 words in software. Our evaluation results show that by expanding queries with the identified words, we can achieve better code search results than SWordNet. The results confirm the usefulness of our method in programming practice.

We have publicly released the identified semantically similar words at http://SWordMap.github.io.

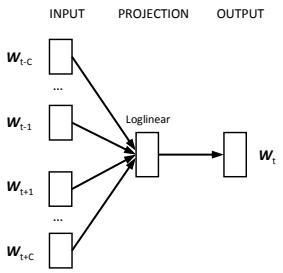

Figure 1: A Neural Network Language Model

2. THE PROPOSED METHOD

This section describes SWordMap, our proposed method for learning semantically similar words in software through Word Embedding.

2.1 Word Embedding

Words are not simply symbols. They have semantic meanings. Traditional techniques represent words as digital IDs or one-hot vectors [22] and are limited to represent the semantic meaning of a word, nor can they measure the semantic distance between two words. For example, for easy computation, one may assign an ID to the word *execute* and a different ID to the word *run*. From their IDs, it is impossible to identify that these words are similar words in some context.

Word Embedding (also known as distributed representation of words) is a technique to represent words as fixed-length vectors so that semantic similar words are close to each other in the vector space [14]. For example, the word *execute* may be represented as [0.12, -0.32, 0.01] and the word *run* may be represented as [0.12, -0.31, 0.02]. From their vectors, we can easily compute their distance and identify the semantic relations.

The word vectors can be inferred by their contextual features in sentences [15]. Intuitively, two words that always have the similar context in a sentence tend to be semantically related. For example, as *image* and *img* always occur in similar context such as *save image as png* and *save img as png*, we can infer that *image* and *img* are similar words. Therefore, we start the word embedding from language model, a model that captures the relations between a word and its context words [14]. In particular, we use neural language model, a model that captures the contextual relations using a neural network [14]. Figure 1 shows an example of neural language model. It captures the relations between word w_t with its context words, namely, $P(w_t|w_{t-C}, ..., w_{t-1}, w_{t+1}, ..., w_{t+C})$, where C is a constant. The network first embeds each word into a vector space (each word w_i is represented as a vector $\boldsymbol{w}_i$). Then, in the projection layer, vectors of context words are aggregated using non-linear functions such as log-linear(softmax) function [14]. Finally, it predicts w_t according to the aggregation of context. The vector representations $\boldsymbol{w}_i$'s are parameters of the network and are trained/updated to maximize the likelihoods in a text corpus.

After training the neural network, we get the vectorial representation $\boldsymbol{w}_i$ of each word w_i. We can then compute their distances (cosine or Euclidean) and identify the top N close words as semantically related words.

Table 1: Examples of Semantically Related Words

word	semantically related words
save	saves, saving, saved, store, backup...
delete	deletes, remove, deleting, deleted, insert...
directory	folder, directories, dir, subdirectory, folders...
image	images, picture, png, bitmap, img...
db	database, databse, sql, databases, jdbc...
conection	connection, connexion, connections...
excel	xls, xlsx, csv, spreadsheet, ods...

2.2 Learning Semantically Related Words in Software

We apply Word Embedding to learn semantically related words in software. We collect a corpus of 708,473 questions from *StackOverflow*[1], which are all Stackoverflow posts that have the tag "Java". We tokenize the raw data and use the results as the input corpus, which contains 5,913,765 sentences and 73,396,324 word occurrences. We use the CBOW (continuous bag-of-words) model [15] to train the neural language model and obtain word vectors. For implementation, we choose word2vec[2], which provides an efficient implementation of the neural language model for computing vector representations of words. The vector size is set to 200.

Figure 2 illustrates the embedding of the words in a 2D projection. After word embedding, we compute the pairwise distance among all words using the cosine similarity of their vectors. For each word, we select the top N most similar words. Finally, we obtain semantically related words of 19,332 unique words.

Table 1 shows some examples of the identified semantically related words in software. For example, for the word *save*, SWordMap successfully identifies its morphological variants such as *saves*, *saving*, and *saved*. It also identifies the semantically related words such as *store* and *backup*. SWordMap can identify abbreviations as well. For example, for the word *excel*, *xls*, *xlsx*, *csv* are identified. Interestingly, SWordMap can also recognize miss-spellings. For example, the word *conection* is misspelled, but SWordMap can still identify its semantically related words.

A number of previous methods also try to extract semantically related words based on the context of words in source code. For example, Shepherd et al. [17] extract verb-DO (direct object) pairs from software projects using natural language processing (NLP) techniques. However, it relies on heuristics regarding the naming convention and the structure of code identifiers and comments. SWordNet [26] infers semantic similarity of two words by computing the text similarity between the sentences that contain the words. SWordMap can identify more accurate semantically similar words, as Word Embedding can better capture the context of a word from a large-scale software corpus.

3. EXPERIMENTS

Our evaluation addresses the following research questions.

3.1 RQ1: How accurate are the semantically related words produced by SWordMap?

Experimental Design This RQ is to evaluate the accuracy of the identified semantically related words. We randomly sample 100 words from the results produced by

[1]http://stackoverflow.com
[2]https://code.google.com/p/word2vec/

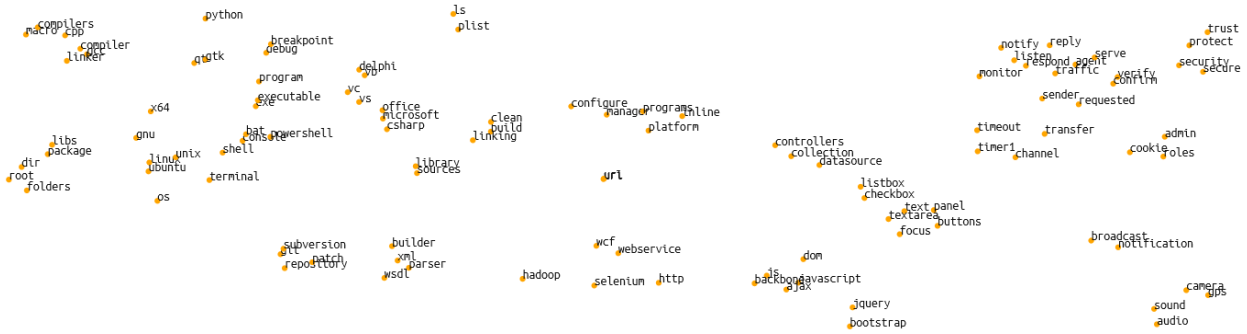

Figure 2: A 2D projection of embeddings of semantically related words (using t-SNE [23])

Table 2: Evaluated Java Projects

Project	Description	LOC	# Methods
iReport	Report Generator	74,506	7,587
javaHMO	Media Server	25,988	1,787
jBidWatcher	eBay Auction Monitor	23,502	1,918
jajuk	Music Player	20,679	2,132

SWordMap, and manually verify if the semantically related words of each sampled word are truly relevant. The accuracy is measured as the average number of correct semantic related words in the top 10 returned results (i.e., Precision@10). We also check whether the correct results can be found in English thesaurus (i.e., in either WordNet [5] or Merriam-webster [2]).

Results We examine the top 10 returned results for the 100 randomly sampled words, and check the number of related words that are truly semantically related. To reduce the subjectivity, the results are verified by two authors. Our results show that the average accuracy is 75.60% (with a standard deviation 0.22), which means that around 76% of the top 10 returned results are deemed relevant. Among the correctly returned results, 75.79% of them cannot be found in the thesaurus of WordNet and Merriam-webster.

3.2 RQ2: Can identified semantically related words improve concern location?

Experimental Design During software development, developers often need to search for code that implements a concern, which is a functionality that is scattered across multiple software modules. The results of concern location provide the *seeds* or starting points, and can be used in conjunction with program navigation tools to locate code [17]. This RQ evaluates if SWordMap can improve the accuracy of concern location.

To answer this RQ, we perform the same concern location queries as related work [17, 25, 26] used, on the same open source Java projects (as shown in Table 2). We use the same regular expressions as Yang et al. did [26] to search for functions that are related to a concern. For each query, we expand it with its semantically related words and construct the regular expressions (we use the top 40 most relevant words in our experiments). For example, for the user query *gather file*, if *collect* is identified as a related word of *gather*, the expanded queries will be *.*gather.*file.**, *.*collect.*file.**, *.*file.*gather.** and *.*file.*collect.**. We also compare our results with the results obtained by SWordNet (we directly use the results of SWordNet as given in [26]).

Results Table 3 shows the results of concern location for the four subjects. In terms of F-measure, SWordMap performs better than SWordNet in 5 out of 8 queries, and better than SWordNet-T in 6 queries. Note that SWordMap can answer queries that cannot be answered by the other two. For example, for the query *play song*, our approach outperforms SWordNet because it finds that *song* and *playlist* are semantically related words. Therefore, it can locate functions that contain *playlist*, such as *PlaylistFile.play*. Overall, the F-measure shows that our approach outperforms SWordNet and transitive SWordNet, the average improvement is 36.6% and 600%, respectively.

The results also show that SWordMap still has room for improvement. For example, for the query *load listing*, the pairs (*listing*, *container*) and (*load*, *reload*) should be identified in order to locate the concern. Our approach successfully identify (*load*, *reload*), but fail to identify (*listing*, *container*), which implies that our training corpus could be further expanded to gain more general results.

3.3 RQ3: Can identified semantically related words improve local code search?

Experimental Design Local code search helps developers find functions or code snippets in a local code base. The results of local code search tools provide the starting points for many programming tasks [16]. This RQ evaluates whether the results of SWordMap can improve the accuracy of local search of functions.

To answer the RQ, we design 25 function search queries for the four Java projects shown in Table 2. For each query, we manually create the "ground truth" by reading the project source code and selecting the directly related functions that match the query. To evaluate SWordMap, we expand the queries with the semantically related words produced by SWordMap using a simple Boolean model. That is, we build the Boolean clause of a word by automatically adding all its semantic related words using OR logic, and then concatenate all the sub-clauses by AND logic. For example, for the search query *delete auction*, assuming that the semantic related words of *delete* are *del* and *remove*, the semantically related word of *auction* is *bid*, then the Boolean query is *(delete OR del OR remove) AND (auction OR bid)*. For a user query, we first remove all its stop words. We then expand the query using the Boolean model and use it as the input to the code search tool.

To compare with the SWordNet approach, we also expand the same queries with the semantically related words produced by SWordNet (including its transitive version SWordNet-T). For each query, we manually check the correctness of the search results and compute the Recall/Precision values based on the ground truth.

Table 3: Results of Concern Location(SWN: SWordNet, SWN-T: Transitive SWordNet, SWM: SWordMap)

Search Task		Initial Search Query	Precision			Recall			F-measure		
			SWN	SWN-T	**SWM**	SWN	SWN-T	**SWM**	SWN	SWN-T	**SWM**
iReport	Add Textfield	.*add.*textfield.*	14.30%	5.30%	0	40%	60%	0	0.211	0.097	0
	Compile report	.*compile.*report.*	4%	0.17%	6.52%	25%	87.50%	37.50%	0.069	0.003	0.111
javaHMO	Gather music files	.*gather.*file.*	28.60%	0.50%	28.57%	50%	75%	50%	0.364	0.010	0.364
	Load movie listing	.*load.*listing.*	0	0	0	0	0	0	0	0	0
jBidWatcher	Add auction	.*add.*auction.*	3.70%	0.94%	9.41%	100%	100%	72.73%	0.071	0.019	0.167
	Save auction	.*save.*auction.*	1.61%	0.66%	7.41%	33.30%	77.80%	66.67%	0.031	0.013	0.133
	Set snipe price	.*set.*price.*	0	0	2.60%	0	0	16.67%	0	0	0.045
jajuk	Play song	.*play.*song.*	0	0	11.54%	0	0	75%	0	0	0.200
Average			6.53%	0.95%	**8.26%**	31.04%	50.04%	**39.82%**	0.093	0.018	**0.127**

Table 4: Results for Finding Directly Related Functions

Search Task		Precision				Recall			
		Initial	SWN	SWN-T	**SWM**	Initial	SWN	SWN-T	**SWM**
iReport	initialize a chart dialog	0	0	0	54.55%	0	0	0	100%
	get the choosen text	0	0	0	31.82%	0	0	0	100%
	get db connection credentials	0	0	0	40%	0	0	0	100%
	export to excel	0	0	0	100%	0	0	0	100%
	set attributes of a chart	0	0	0	14.81%	0	0	0	57.14%
	copy a dataset	16.67%	0.76%	0.13%	9.46%	14.29%	14.29%	100%	100%
javaHMO	load audio data	100%	25%	0.26%	66.67%	20%	60%	60%	40%
	collect all music files	0	0	0	0	0	0	0	0
	get the category of a music	0	0	0	9.09%	0	0	0	100%
	get the thumbnail of a movie	0	4.76%	0.20%	7.69%	0	100%	100%	100%
	set the singer of a song	0	0	0	8.70%	0	0	0	100%
	extract network domain information	0	0	0	100%	0	0	0	100%
	configure server ip address	0	0	0	16.67%	0	0	0	100%
	find image files	0	2.82%	0.17%	25%	0	100%	100%	100%
jBidWatcher	delete an auction	66.67%	0.61%	0.38%	10%	33.33%	83.33%	100%	100%
	get the best bid price	0	0	0	100%	0	0	0	100%
	update an auction	50%	1.31%	0.66%	17.74%	45.45%	100%	100%	100%
	buy an item	0	0.56%	0.26%	0	0	75%	100%	0
	login to ebay	0	0	0	0	0	0	0	0
	translate between currency	0	0	0	100%	0	0	0	100%
jajuk	get playable songs	0	0	0	0	0	0	0	0
	add a playlist	40.0%	10.17%	2.60%	4.17%	33.33%	100%	100%	33.33%
	remove a song from playlist	0	0	0	8.70%	0	0	0	100%
	get the author of a track	100%	0.51%	0.25%	100%	100%	100%	100%	100%
	silence the player	0	0	0	100%	0	0	0	100%
Average		14.93%	1.86%	0.20%	**37.0%**	9.86%	29.3%	34.4%	**77.22%**

Results Table 4 shows the results of local code search aiming at finding directly related functions in the four Java projects. SWordMap can help answer many queries that cannot be answered initially (without query expansion), with an average Recall and Precision of 77.22% and 37.0%. The results also show that in general, SWordMap outperforms SWordNet and transitive SWordNet significantly. For example, for the query *export to excel*, the matching functions include $CSVExporter.export$ and $CSVExporter.addCSV$. SWordMap successfully finds that CSV and *excel* are semantically related words, while SWordNet fails because *excel* does not show up in the iReport code.

4. RELATED WORK

In order to improve the accuracy of code search (and information retrieval in software engineering in general), a number of techniques for extracting semantically related words in software have been proposed [18, 17, 8, 9, 25, 24, 10, 26, 20, 21]. For example, Shepherd et al. [18, 17] leveraged NLP techniques to extract verb-DO pairs from method signatures and comments. Hill [8] proposed SWUM, which refines the verb-DO work. Yang et al. [25] identified semantically related word pairs (rPairs) if the two words are used in the same context in comments and code. They further proposed SWordNet [26], which improves their previous work by introducing a new similarity measure and ranking function. Their experimental results show that SWordNet outperforms SWUM in code search. Similarly, Howard et al. [10] identified semantically similar pairs from comment-code mappings, focusing on the main action verb extracted from comments and method signatures. Tian et al. [20, 21] presented SEWordSim, which also identifies semantically related words from Stackoverflow posts. However, their approach measures the similarity of words simply by word co-occurrence. SWordMap differs from the related techniques in that, it considers deep semantic about the context of a word and is therefore more accurate.

5. CONCLUSION

In this paper, we propose SWordMap, which applies Word Embedding technique to software documents and identifies semantically related words in software. These semantically related words can be used to expand code search queries. Our experiments on two code search tasks, i.e., concern location and local code search, have shown that SWordMap outperforms the related work SWordNet.

Currently we collect Stackoverflow posts as the training corpus, which may be still insufficient. In future, we will collect more software engineering documents from a variety of sources to expand the corpus. Our current experiments are small in scope and involve a degree of subjectivity. We will perform a large-scale evaluation in our future work.

6. ACKNOWLEDGMENT

We are grateful to Jinqiu Yang and Lin Tan, the authors of [26], for generously sharing with us the results of SWordNet.

7. REFERENCES

[1] Krugle code search. http://www.krugle.com/.

[2] Merriam-webster english dictionary and thesaurus. http://www.merriam-webster.com/.

[3] Ohloh code search. https://code.ohloh.net/.

[4] Sando code search tool. http://sandosearch.weebly.com/.

[5] WordNet. Princeton University, http://wordnet.princeton.edu/.

[6] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, 1987.

[7] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 842–851. IEEE Press, 2013.

[8] E. Hill. *Integrating natural language and program structure information to improve software search and exploration*. University of Delaware, 2010.

[9] E. Hill, L. Pollock, and K. Vijay-Shanker. Improving source code search with natural language phrasal representations of method signatures. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 524–527. IEEE Computer Society, 2011.

[10] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker. Automatically mining software-based, semantically-similar words from comment-code mappings. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 377–386. IEEE Press, 2013.

[11] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18(2):300–336, 2009.

[12] F. Lv, H. Zhang, J.-g. Lou, S. Wang, D. Zhang, and J. Zhao. Codehow: Effective code search based on API understanding and extended boolean model. In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 260–270. IEEE, 2015.

[13] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: finding relevant functions and their usage. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 111–120. IEEE, 2011.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[16] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz. Sando: an extensible local code search framework. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 15. ACM, 2012.

[17] 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 *Proceedings of the 6th international conference on Aspect-oriented software development*, pages 212–224. ACM, 2007.

[18] D. Shepherd, L. Pollock, and K. Vijay-Shanker. Towards supporting on-demand virtual remodularization using program graphs. In *Proceedings of the 5th international conference on Aspect-oriented software development*, pages 3–14. ACM, 2006.

[19] G. Sridhara, E. Hill, L. 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.

[20] Y. Tian, D. Lo, and J. Lawall. Automated construction of a software-specific word similarity database. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 44–53. IEEE, 2014.

[21] Y. Tian, D. Lo, and J. Lawall. Sewordsim: software-specific word similarity database. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 568–571. ACM, 2014.

[22] J. Turian, L. Ratinov, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[23] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[24] S. Wang, D. Lo, and L. Jiang. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 604–607. IEEE, 2012.

[25] J. Yang and L. Tan. Inferring semantically related words from software context. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 161–170. IEEE Press, 2012.

[26] J. Yang and L. Tan. Swordnet: Inferring semantically related words from software context. *Empirical Software Engineering*, 19(6):1856–1886, 2014.