

Quantifying Semantic Similarity of Software Projects Using Deep Semantic Similarity Model(DSSM)

Akond Ashfaq Ur Rahman
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
aarahman@ncsu.edu

Abstract—Software re-usability can help software organizations to achieve rapid construction of software saving software teams in effort and time. Finding similar projects within a software organization as well as within the open source domain, can help software teams to gather source code that can be reusable helping them to construct software at a faster rate. *The goal of this paper is to help software teams in facilitating software re-usability using semantic similarity and deep learning techniques on source code of software applications.* The author uses available open source techniques to gather natural language tokens from software projects, and use them with deep learning techniques to quantify semantic similarity. The author uses Microsoft's Deep Semantic Similarity Model (DSSM) to quantify semantic similarity on nine open source software projects. The paper also includes relevant empirical findings that illustrates how the proposed methodology works with nine real-world software projects obtained from Github. In addition to the empirical findings, the author discusses how the proposed methodology can be extended to advance industry and academic research in the area of software reuse and increased productivity.

Index Terms—software repositories; semantic similarity; Deep Semantic Similarity Model (DSSM)

I. INTRODUCTION

Lim [1] observed that re-use of software components overall has a positive impact on software quality. Lim observed that when software is re-used in a software team, defects decreases, and productivity increases. Software teams can use different aspects of software artifacts to achieve software re-usability e.g., architecture of the software, source code and of the software, and designs and documentations of the software [2]. Amongst these numerous aspects of software artifacts, source code of software is the most frequently referred artifacts for software reuse [3]. Researcher has observed that software engineers tend to adopt different strategies to identify and manipulate reusable portions of software code such as syntax-based search using *grep-like* tools, communicating with colleagues, and using the web [3]. These strategies can be time and effort consuming leading to reduced productivity [4]. Automated tools that quantifies similarity amongst software projects can help software developers to identify and work on re-usable portions of existing software codebase. This paper proposes and illustrates a methodology that quantifies the semantic

similarities between software projects in order to help software developers in finding similar software projects and software components. *The goal of this paper is to help software teams in facilitating software re-usability using semantic similarity and deep learning techniques on source code of software applications.* The author proposes a methodology that parses codebases of software repositories to create a list of natural language tokens, and use that collection of natural language tokens to investigate the semantic similarity between software projects. The author has used open source utilities such as SrcML.NET¹, and Swum.NET² to parse and filter the software projects into natural language tokens. The author has also used Microsoft's *Deep Semantic Similarity Model (DSSM)* and *Sent2Vec*³ utilities to investigate and quantify the semantic similarity between different software projects. The author chose open source software projects written in C, C#, and Java from Github⁴ to illustrate how the proposed methodology works. In this work the author has also presented if different combinations of DSSM parameters can impact the semantic similarity between software projects. The author presents the contributions of this paper as following:

- A proposed methodology that illustrates how Microsoft's DSSM and Sent2Vec utilities can be used to quantify the semantic similarity between software projects.
- A discussion on how the proposed methodology can be used to advance industry and academic research in the domain of software re-usability.

The rest of the paper is organized as follows: the author provides background information and related work in Section II. The author presents the methodology in Section III. The author presents empirical findings in Section IV. The author discuss the findings of the study in Section V. The limitations of the study are presented in Section VI. Challenges in conducting the study is presented in Section VII. Finally the author concludes the paper in Section VIII.

¹<https://github.com/abb-iss/SrcML.NET>

²<https://github.com/abb-iss/Swum.NET>

³<http://research.microsoft.com/en-us/projects/dssm/>

⁴<http://github.com/>

II. BACKGROUND AND RELATED WORK

The author uses this section to describe the necessary concepts used in the study as well as prior academic studies that are related to this paper.

A. Background

The author provides brief background on the utilities that have been used in the paper.

1) *SrcML.NET*: SrcML.NET is a framework developed in C# that is used to perform program transformation and code analysis. SrcML.NET is based on the ‘srcML’ project from Kent State University Software Development Laboratory. SrcML.NET provides an API to extract source code elements from a software project. In this study the author extracted raw natural language tokens from software projects and stored them in XML formats. The granularity of extracting the natural language tokens is at a file level i.e. in the XML file the corresponding tokens of one file was indexed according to its file name.

2) *Swum.NET*: Swum.NET is a tool developed in C# that removes alpha-numeric symbols and converts identifiers constructed in a camel case or pascal case format to get the natural language tokens.

3) *Deep Semantic Similarity Model (DSSM)*: DSSM is a variant of deep neural networks designed for text analysis. DSSM can be trained on large collection of text documents, and maps source-target document pairs to feature vectors in a latent space in such a way that the distance between source documents and their corresponding target documents in that space is minimized. In this paper, natural language tokens extracted from software projects are used as source and target document pairs. Gao et al. [5] demonstrated the effectiveness of DSSM using two tasks that reveal interestingness namely, automatic highlighting and contextual entity search. DSSM has different parameters that can be tuned to setup different experiments for the purpose of study. The author has used different values for the *MAX_ITER* parameter in this paper. *MAX_ITER* corresponds to the maximum number of iterations to train DSSM.

4) *Sent2Vec*: Sent2Vec is a utility that maps a pair of short natural language tokens to a pair of feature vectors in a continuous, low-dimensional space where the semantic similarity between the natural language tokens is computed as the cosine similarity between their vectors in that space. Sent2Vec performs the mapping using the DSSM. Sent2Vec has different parameters that can be tuned to setup different experiments. The parameters that have been used in the study are:

- *inSrcModel* : the neural network to embed the source string.
- *inTgtModel* : the neural network to embed the target string.
- *inFilename* : the input sentence pair file where each line is a pair of natural language tokens, separated by tab. For this paper the set of natural language tokens presented before the tab comes from one of the two software projects

that are being compared. The natural language tokens of the other software project are added after the tab.

- *inSrcModelType* : the type of the source model, which can be DSSM or CDSSM. In this paper for all the experiments the author uses DSSM.
- *inTgtModelType* : the type of the source model, which can be DSSM or CDSSM. In this paper for all the experiments the author uses DSSM.
- *outFilenamePrefix* : the filename prefix to be used to output the similarity scores and the semantic vectors of the natural language tokens that are served as input.

B. Semantic Similarity in Software Engineering

The paper is closely related to prior academic works that have investigated on how semantic similarity can be used to aid software developers and software teams. Cubranic et al. [6] created and proposed a tool called *Hipikat* that used semantic features of software tasks to recommend next probable software modification tasks within an IDE. Maletic et al. [7] investigated and provided empirical findings on how semantic and structural information of source code elements can facilitate software developers in understanding software programs. Kiefer et al. [8] investigated how semantic features of software repositories extracted using Web Ontology Languages can be used to characterize a software’s evolution. Jalbert et al. [9] leveraged on textual semantics of bug reports to identify duplicate bugs in order to reduce development cost. Antoniol et al. [10] used information retrieval to investigate how connections between the source code and documentation can be discovered. Yao et al. [11] used a semantic-based approach to classify and extract necessary information from re-usable software components.

This paper takes a different stance on how semantic features of software projects can be beneficial for software teams and software developers. This paper proposes a methodology that takes natural language tokens as input to DSSM, and uses the trained DSSM to quantify the semantic similarity amongst multiple software projects.

III. METHODOLOGY

The author uses this section to describe the steps to perform relevant implementation and analysis. As shown in Figure 1, the study involved four major steps namely, collection of software repositories, extracting tokens, training models, and obtaining similarity scores. Finally the author ends this section by describing the experiments used in this study.

A. Collecting Software Repositories

For analysis the author used popular Github projects written in three languages: C, C#, and Java. The author hypothesizes that semantic similarity might be different for different projects in different programming languages, and analysis of such aspect can bring better insight with respect to semantic similarity. Keeping this assertion in mind, the author has considered three different programming languages: C, C#, and Java.

The author selected projects that were ‘trending’ on Github between March 01, 2016 – March 30, 2016, and had a size

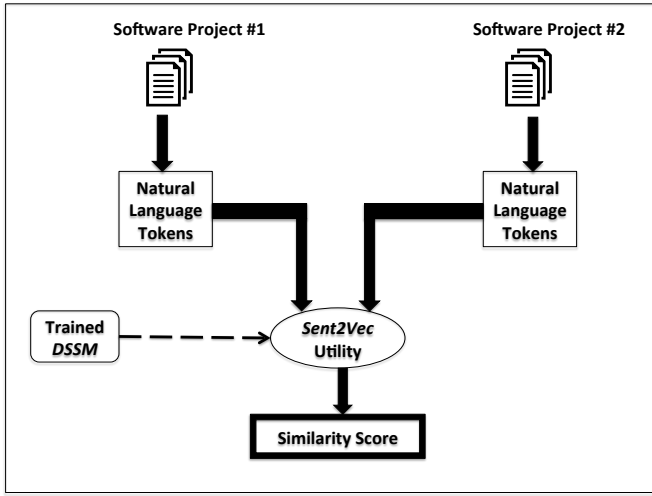


Fig. 1. Major Steps Used in the Paper

in between 1MB – 100MB. The process was repeated for three programming languages namely C, C#, and Java. By implementing this process two projects from the C language, one project from the C# language, and three projects from the Java language. Please note that ‘trending’ is a feature of Github that ranks Github repositories that gained popularity amongst Github users for a certain time period such as a day, a week, or a month.

B. Token Extraction

The process of semantic similarity involves a collection of words or tokens using which a semantic model will be built to quantify the semantic similarity between two projects. To achieve this goal, the author used a two step process to gather necessary tokens from each of the software repositories of interest. These steps are presented below:

- The author used an open source tool called SrcML.NET to extract all the tokens from each repository. The SrcML.NET program takes the directory of each repository as input, and produces all tokens in one xml file.
- The author used another open source tool called Swum.NET that filters the tokens generated from SrcML.NET. The motivation of executing this step was to obtain natural language tokens from the tokens generated in the previous step by conducting the pre-processing steps:
 - convert camel-case and pascal-case tokens into natural language tokens
 - convert alpha-numeric tokens into natural language tokens such as converting to `_a_variable` to `a_variable`
 - remove tokens that are a length of one
 - remove tokens from the token collection that are stop words in the English language
 - remove tokens from the token collection that are programming language specific keywords such as void, int, and main.

After performing this step for each repository a .tsv file was created that was later used in training semantic models as well as evaluating the semantic similarity of software repositories.

C. Training Semantic Models

The author has used the Deep Structured Semantic Model (DSSM) to quantify the semantic similarity between projects. According to Huang et al. [12] DSSM learns from a query or a document which can be used to compute semantic similarity. DSSM trains itself by projecting semantically similar phrases that are close to other, and projecting semantically dissimilar phrases that are further to another. DSSM also provides configuration options to create different training models from the same document. Amongst these configuration options the author has used two parameters namely *BATCHSIZE*, and *MAX_ITER*. *BATCHSIZE* refers to how many training pairs can be used to train. *MAX_ITER* refers to the total count of iterations DSSM will use to create the training model. Similar to Yih et al. [13] the author performed the following actions to create the training model for each configuration:

- Shuffle the query pairs using the *WordHash* utility with the *shuffle* flag.
- Following Gao et al. [5] the author generate the sequence of letter trigram features using the *WordHash* utility with the *pair2seqfea* flag.
- The generated sequences of letter trigram features are converted to binary files using the *WordHash* utility with the *seqfea2bin* flag.
- Next, the noise distribution of the training data is calculated using the *ComputelogPD* utility.
- Using a configuration file, and the *DSSM_Train* utility, the author created training models that are used to perform the analysis of this paper. The author describes which model is used for what experiment in Section III-D.

D. Experiments

The author has designed four experiments to evaluate the similarity between two projects, in a pair-wise fashion. In these experiments the author has varied the training model with respect to DSSM configuration parameters as well as the documents that are using. Table I presents the experiments and the parameters that were changed. The experiments are referred by their names in Section IV.

In each of these experiments, similar to Huang et al. [12]’s approach the author has used the Sent2Vec utility for comparing the six projects of interest. Sent2Vec uses cosine similarity to quantify the vectors of tokens amongst two projects [12]. In each of the experiment, the six projects are compared in a pair-wise manner i.e. each of the six projects are compared to the rest of the five projects of interest. The author has followed the format of the provided example that comes with the Sent2Vec utility⁵ to compare the similarity between two projects. The input files to the Sent2Vec utility was created in

⁵<http://research.microsoft.com/en-us/downloads/731572aa-98e4-4c50-b99d-ae3f0c9562b9/>

TABLE I
EXPERIMENTS USED IN THE STUDY

Name	Training Source	MAX_ITER
Experiment-1	Largest repository	500
Experiment-2	Largest repository	100,000
Experiment-3	Smallest repository	500
Experiment-4	Smallest repository	100,000

TABLE II
REPOSITORIES USED IN THE STUDY

Name	Version or Branch	Language	Size (MB)	Count
Git ⁶	master	C	6.60	247,303
Redis ⁷	3.20	C	1.70	183,295
ShareX ⁸	master	C#	6.40	788,962
Douya ⁹	master	Java	9.60	781,618
RxJava ¹⁰	master	Java	1.00	560,690
SeeWeather ¹¹	master	Java	3.20	164,263

the following manner: the author splitted the natural languages tokens of the two projects of interest into 100000 buckets. Each line in the input file corresponded to the tokens for both projects in each bucket. For example, line #1 in the input file corresponded to the tokens in bucket#1, for both projects. The process was repeated for each combination of projects, and as a result, 30 input files were created. The ‘Training Source’ column in Table I corresponds to the repository of how the DSSM model will be trained. For example, in Experiment-1, and Experiment-3, the natural language tokens from the largest and the smallest repositories of the six software projects that are used in the study will train the DSSM model.

Sent2Vec generates scores for each line in the input file. This score reflects the semantic similarity scores between the tokens that are separated by tabs. The author used a median approach to get the overall semantic score between two projects i.e. the median of all the scores for each line in the input file was used to determine the semantic similarity. For example, if the semantic similarity scores between project P and project Q is [0.1, 0.01, 0.54, 0.21, -0.43, 0.76, 0.31], then the semantic similarity is measured as 0.21, according to the author’s proposed approach.

IV. RESULTS

The author describes relevant empirical findings in this section.

A. Software Repositories

Table II presents the repositories names and the number of tokens for each project after applying filtering techniques used in the study.

Table II presents the count of tokens achieved from each repository after performing the filtering steps. According to Table II the largest and the smallest repositories used in the study are ShareX, and SeeWeather respectively. The filtered natural language tokens obtained from ShareX, and SeeWeather are used to create two different training models for DSSM. The goal of creating two different training models is to see if inclusion of tokens might have an impact of similarity

TABLE III
RESULTS FOR EXPERIMENT-1

	Git	Redis	ShareX	Douya	RxJava	SeeWeather
Git	—	0.001	0.000	0.003	0.001	0.001
Redis	0.000	—	-0.001	-0.005	0.003	-0.002
ShareX	0.005	0.008	—	0.011	0.033	0.003
Douya	0.002	0.004	-0.015	—	-0.005	0.008
RxJava	0.000	0.001	-0.143	-0.012	—	-0.003
SeeWeather	0.001	0.000	-0.011	0.000	0.000	—

measure. The scripts used to create the training DSSMs and used to execute Sent2Vec utility for Experiment-1, 2, 3, and 4 are available online ¹². The files that are used as input to the Sent2Vec utility are available online ¹³. The output of the Sent2Vec utility for each of the input files are also available online ¹⁴. A sample snapshot of one of the input files to Sent2Vec and the corresponding output is provided in Figure 2(a), and Figure 2(b) respectively. As shown in in Figure 2(a), the left column corresponds to the tokens of one project, whereas the tokens on the right column corresponds to the tokens of the other project, in a pair-wise comparison.

path die error list len	0.059659
retval cwd current	-0.071269
depth last elem next list	-0.052226
last slash node list	-0.112386
next len path node list	-0.061811
path path old node	-0.019025
pfx pfx len node list	0.019311
arg path node	-0.058535
stat isdir die list direction	0.045515
strbuf reset iter iter	0.083571
strbuf addstr directory list list	0.067599
find last iter current	-0.051256
dir sep xstrdup orig copy	-0.057937
strbuf setlen iter	0.054426
xmempdupz strbuf reset node list	-0.064294
strbuf getcwd key iter	

Fig. 2. Sample Input and Output Used for Sent2Vec

B. Similarity

As stated before in Section III the author used two different training models as well as used different values for the two parameters of DSSM namely *MAX_ITER* and *TARGET_LAYER_DIM*. The combination of these settings led to different experiments.

According to Table III the most similar projects are ShareX and RxJava as they have the highest semantic similarity scores. Interestingly, ShareX is written in C#, and RxJava is written in Java. Please recall that the semantic similarity score is the

¹²<http://tiny.cc/ymroay>

¹³<https://github.com/akondrahman/Miscellaneous/tree/master/OpenSourceChallenge/input-to-sent2vec>

¹⁴<https://github.com/akondrahman/Miscellaneous/tree/master/OpenSourceChallenge/output-of-sent2vec>

TABLE IV
RESULTS FOR EXPERIMENT-3

	Git	Redis	ShareX	Douya	RxJava	SeeWeather
Git	—	0.001	0.000	-0.003	0.001	-0.001
Redis	0.000	—	-0.001	-0.005	0.003	-0.002
ShareX	0.005	0.008	—	0.011	0.033	0.003
Douya	0.002	0.004	-0.015	—	-0.005	0.008
RxJava	0.000	0.001	-0.014	-0.012	—	-0.004
SeeWeather	0.000	0.000	-0.011	0.000	0.000	—

median of all the semantic scores between the natural language tokens of the two corresponding software projects that are splitted across 100,000 buckets. Please note that the entries in Table III and IV are not symmetric across the diagonal as the order of tokens in the input file for two projects can be different.

From Table IV the author observes that the similarity scores are overall the same between overall the projects. For further investigation the author has applied the following hypothesis test:

- H_0 : The semantic similarity scores for Experiment-1 and Experiment-3 are not different.
- H_A : The semantic similarity scores for Experiment-1 and Experiment-3 are different.

With 99% statistical confidence the author has observed that the semantic similarity scores between Experiment-1 and Experiment-3 are not statistically different. Furthermore, the author has applied more hypothesis tests to observe if the semantic similarity scores obtained in each experiment is different to that of the others. The author has observed that there are no significant differences between the semantic similarity scores of Experiment-1, 2, 3, and 4. As the empirical findings of Experiment-2 and Experiment-4 are similar to that of Experiment-1 and Experiment-3, respectively, the author does not present findings for Experiment-2 and Experiment-4 in this paper.

Observations: The author presents the following observations from the empirical findings of the study:

- DSSM and Sent2Vec has the potential to capture semantic similarity between two software projects that are written in different programming languages. For example, the author has showed in this study that the highest similarity score in all experiments were between ShareX and RxJava.
- Number of iterations to train DSSM did not have an impact on the semantic similarity between software projects as the semantic score obtained from all experiments are not significantly different.
- The natural language tokens obtained from two different repositories did not have an impact on the semantic similarity score as the semantic score obtained from all experiments are not significantly different.

V. DISCUSSION

In this section the author discusses the implications of the empirical findings for programmers and researchers.

A. Identifying Re-usable Software Components

Modern software teams have to maintain numerous software repositories. Software developers might have to use one or multiple components across one or different software repositories. Navigating through multiple software repositories to identify one or multiple components for re-use can be non-trivial as the process involves cognitive effort [14] and time [4]. An automated software tool that identifies similar software components within the software team, as well as in an open source software repository hub such as Github or Sourceforge¹⁵, eventually saving software developers' time and effort.

B. Software Development in a Distributed Environment

In many organizations software teams are geo-graphically distributed. For the purpose of software re-usability, coordinating amongst such distributed teams can be non-trivial as these teams might be in different time zones. An automated tool that captures semantic similarity across projects can be helpful in this regard. By-passing the communication and co-ordination methods such as setting up a Skype meeting, the tool can find and recommend software components that are similar to the developers' needs.

C. Application in Software Security

Semantic similarity between two projects has the potential to be used to estimate software vulnerability and defects. For example, if component A of project X has vulnerabilities, and if component B of project Y is similar to that of component A, then component B might be susceptible to vulnerabilities. Such estimation can help software teams to apply a pro-active approach in preventing software vulnerabilities. For example, after developing a certain component C, a software team can investigate how similar it is to a known vulnerable software component. If the team finds strong similarity then that can be treated as an indication of component C being vulnerable. Such approach might reduce the probability of releasing vulnerable software components to end-users.

D. Intelligent IDE Design

Effective similarity measurement techniques between software projects can revolutionize code search features that are already existing in current desktop and cloud-based IDEs. Overall, current IDEs provide auto-completion features, as well as recommendations for tasks, and recommendations for code artifacts within the software project. Within an IT organization software teams can be geographically distributed, and can have large amount of software repositories. In such conditions semantic-based similarity approaches can help to narrow down the search space for IDEs to recommend the relevant software components. For example, when the developer types the word *crypto*, the IDE might display the relevant package that has the implementation of cryptographic methods located in a remote repository. The IDE can also display multiple components based on a suitable ranking measure.

¹⁵<http://sourceforge.net/>

VI. LIMITATIONS

We discuss the limitations of our study in this section as following:

Lack of Ground Truth: In Section IV the author has showed how DSSM and Sent2Vec can be used to compute similarity scores between different software projects without describing the ground truth. The author has not demonstrated why or how RxJava and ShareX have the highest similarity scores.

Granularity: The tokens for each of the nine software projects are calculated at a file level. The author has not discussed how tokens collected at other granularities such as namespaces, binaries, or classes might have a different impact on the similarity scores between projects.

Selection of Software Repositories: The author has selected a small set of repositories for illustration of the methodology. The size of these software repositories varied between 1 MB 100 MB. Further investigation is required to appropriately assess the capability of DSSM and Sent2Vec to quantify semantic similarity between software projects.

Use of Parameters: As shown in Section II and Section III the author has selected and used a specific set of parameters for training the DSSM models. Further exploration and experimentation with all parameters of DSSM is needed to fully assess DSSM's capability of discovering semantic similarity between software projects.

VII. CHALLENGES

Conducting necessary experiments using DSSM was challenging in the following ways:

- **Resolving CudaLib Dependency:** DSSM relies on the *CudaLib* dll to perform its computations. Unfortunately, the provided executable binary file, hosted on the website¹⁶ did not have that added that dependency. As a result, the author was getting a "DllNotFoundException" when creating different DSSM training models for Experiment - 1, 2, 3, and 4. To resolve this, the author first investigated the source code of DSSM, and having observed DSSM's dll dependency on CudaLib, installed *CUDA Toolkit 7.5* and re-executed the Visual Studio project for CudaLib, and DSSM. With the newly created binary the author conducted the necessary experiments.
- **Format of Input for DSSM and Sent2Vec:** Understanding the format of what is passed into DSSM and Sent2Vec was non-trivial. The author explored the contents of the raw .tsv as well as the 'pairs.txt' files to understand the formatting for input files.

VIII. CONCLUSION

In this paper the author has described how Microsoft's Deep Semantic Similarity Model (DSSM) and Sent2Vec can be used to identify and quantify semantic similarity between software projects. The author has also discussed the future implications of this study and how future research in this area can help

software teams to identify re-usable software components with reduced effort. The author believes future academic and industry research in this area has the possibility of revolutionizing standard organizational efforts in software engineering.

REFERENCES

- [1] W. C. Lim, "Effects of reuse on quality, productivity, and economics," *IEEE Softw.*, vol. 11, no. 5, pp. 23–30, Sep. 1994. [Online]. Available: <http://dx.doi.org/10.1109/52.311048>
- [2] W. Frakes and C. Terry, "Software reuse: Metrics and models," *ACM Comput. Surv.*, vol. 28, no. 2, pp. 415–435, Jun. 1996. [Online]. Available: <http://doi.acm.org/10.1145/234528.234531>
- [3] V. Bauer, J. Eckhardt, B. Hauptmann, and M. Klimek, "An exploratory study on reuse at google," in *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices*, ser. SER&IPs 2014. New York, NY, USA: ACM, 2014, pp. 14–23. [Online]. Available: <http://doi.acm.org/10.1145/2593850.2593854>
- [4] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, July 2010.
- [5] J. Gao, P. Pantel, M. Gamon, X. He, and L. Deng, "Modeling interestingness with deep neural networks," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014, pp. 2–13.
- [6] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth, "Hipikat: a project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, June 2005.
- [7] J. I. Maletic and A. Marcus, "Supporting program comprehension using semantic and structural information," in *Proceedings of the 23rd International Conference on Software Engineering*, ser. ICSE '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 103–112. [Online]. Available: <http://dl.acm.org/citation.cfm?id=381473.381484>
- [8] C. Kiefer, A. Bernstein, and J. Tappolet, "Mining software repositories with isparol and a software evolution ontology," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 10–. [Online]. Available: <http://dx.doi.org/10.1109/MSR.2007.21>
- [9] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, June 2008, pp. 52–61.
- [10] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, Oct 2002.
- [11] H. Yao and L. Etzkorn, "Towards a semantic-based approach for software reusable component classification and retrieval," in *Proceedings of the 42Nd Annual Southeast Regional Conference*, ser. ACM-SE 42. New York, NY, USA: ACM, 2004, pp. 110–115. [Online]. Available: <http://doi.acm.org/10.1145/986537.986564>
- [12] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, ser. CIKM '13. New York, NY, USA: ACM, 2013, pp. 2333–2338. [Online]. Available: <http://doi.acm.org/10.1145/2505515.2505665>
- [13] W. Yih, M. Chang, X. He, and J. Gao, "Semantic parsing via staged query graph generation: Question answering with knowledge base," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, 2015, pp. 1321–1331.
- [14] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding understanding source code with functional magnetic resonance imaging," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 378–389. [Online]. Available: <http://doi.acm.org/10.1145/2568225.2568252>

¹⁶<http://research.microsoft.com/en-us/projects/dssm/>