Quantifying Semantic Similarity of Software Projects Using Deep Learning

Akond Ashfaque 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 significant man hours. 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 re-usable 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.* 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 proposal uses Microsoft's Deep Semantic Similarity Model(DSSM) to quantify semantic similarity of nine open source software projects. The paper also includes relevant empirical findings that illsutrates how the proposed emtholodlgy works with nine real-world software projects obtained from Github. In addition to the empirical fidnings, the author discusses hwo the proposed methodlokgy can be extended to advance industry and academic research in the area of software reuse.**

***Index Terms*—software repositories; semantic similarity;**

I. Introduction

Lim [1] in his study observed that reuse of software components overall has a positive impact on software quality. Lim observed that when software is reused in a software team, defects decreases, and productivity increases. Software teams can use different aspects of software artifacats to achieve software resuability for example, architecture of the software, source code and of the software, and deisgns and docuemntations of the software [2]. Aongst these numerous aspects of software artifacts, source code of software is the most frequently referred artfacts for software reuse [3]. Researcher has observed that software engineers tend to adopt different startegies to idnetify and manipulate re-suable 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 comsuming leading to reduced productivity [4]. Automated tools that quanifies similarity amongst software projects can help software developers to identify and work on re-usable protions of existing software codebase. This paper proposes and illsutrates a methoslogy that quantifies the semantic similarities between software projects in order to help software developers in fidning similar software projects and software compoents.

The goal of this paper is to help software teams in facilitating software re-usability using semantic similarity and deep learning techniques.

The author proposes a methodlogy that parses codebases of software repsoitoreis to create a list of natural language tokens, and use that collection of natural lagugae tokens to investigate the semantic similarity between software projects. The author has used open source utilities such as SrcML.NET[1], and Swum.NET[2] to parse and filter the software projects into natural languag e tokens. The author has also used Microsoft's *Deep Semantic Similarity Model (DSSM)* and *Sent2Vec*[3] utilities to inevstigate and quantify the semantic similarity between different software projects. The author chose open source software projects written in C, C#, and Java from Github[4] to illsutrate how the porposed methodlogy works. In this work the author has also presented how different combinations of DSSM can impact the semantic simialrity between software projects.

The author presents the contributions of this paper as follwoing:

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

The rest of the paper is organized as follows: we provide background information and related work in Section II. We present our methodology in Section III. We present empirical findings in Section IV. We discuss the findings our study in Section V. The limitations of our paper are presented in Section VII. Finally we conclude this paper in Section VIII.

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.

[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/

A. Background

The author provides brief backgorund on the utillities 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 absed 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 alnguage tokens from software projects and stored them in XML formats. The granualrity of extracting the anturallanguage tokens is at a file level i.e. in the XML file the corresponding tokens of one file was indexed accordng 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 langage tokens extracted from software projects are used as source and atrget docuemnt 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 parametrs that can be tuned to setup different experiemnts 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 comapred. The antural alnguage 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 it is DSSM
- *inTgtModelType* : the type of the source model, which can be DSSM or CDSSM. In this paper for all the experiments it is DSSM
- *outFilenamePrefix* : the filename prefix to be used to output the similarity scores and the semantic vectors of the natural language tokens tht are served as input

B. Semantic Similarity in Software Engineering

The paper is closely related to prior academic work that have investigated on how semnatic similarity can be used to aid software developers and software teams. Srinivas et al. [6] used dsitribution of features through semantic similairty to cluster and classify software projects. Kuhn et al. [7] used Latent Semantic Indexing(LSI) to group components of the software that are semantically similar. In another work Kun et al. [8] used semantic simialrity to enrich the revese engneering steps of software projects. Asuncion et al. [9] used semantic simialrity to facilitate software traceability in large scale software projects. Cubranic et al. [10] created and propsed a tool called *Hipikat* that used semantic features of software tasks to recommend next prooablabe software modification tasks within an IDE. Maletic et al. [11] investigated and provided empirical findings on how semantic and structural information of source code elements can facilitate softwaare developers in understanding software programs. Kiefer et al. [12] investigated how semantic features of software repositories extracted using Web Ontology Languages can be used to chraecterize a software's evolution. Jalbert et al. [13] leaveraged on textual semantics of bug reports toidentfy duplicate bugs in order to reduce development cost. Antoniol et al. [14] used information retrieveal to investigate how connections between the source code and dcumantation can be discovered. Yao et al. [15] used a semantic-based approach to classify and extarct necessary information from reusable software compoennts.

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 methodlogy that takes natural language tokens as input to DSSM, and uses the trained DSSM to quantify the semantic similarity amongst multiple software porjects.

III. Methodology

The author uses this section to describe the steps to perform relavant 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 languages, and analysis of such aspect can bring better insight with respect tosemntic similarity. Keping this

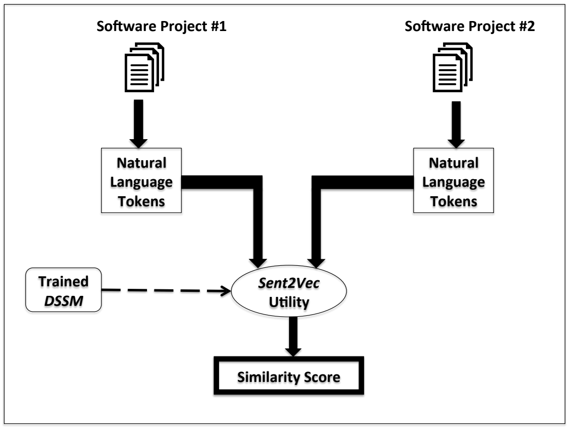

Fig. 1. Major Steps Used in the Paper

assertion in mind, the authors has considered three different programming languages: C, C#, and Java.

The author selected the the first three projects that were ‘trending’ between March 01,2016 March 30, 2016, and had a size in between 1MB 100MB. The process repeated for three programming languages namely C, C#,and Java. Please note that ‘trending’ is a feature of Github that ranks Github repsoitories that gained popularity amongst Github users for a certain time period such as a day, a week, or a month.

B. Token Extraction

The processof semantic similarity involves a collection of words or tokens using a which a semantic modelcan built on. To achieve this goal, the author used a two step process to gather necesary tokens from each of the software repsostories of interest. These steps are presented below:

- The author used an opne source tool called SrcML.NET to extract all the tokens from each repsoitory. 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 genrated from SrcML.NET. The motivation of excuting this step was to obtain natural language tokens from the tokens generated in the previous step by condcting the pre-processing steps:
  - convet 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 languae 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 Dep Structured Semantic Model(DSSM) to quantify the semnatic similarity between projects. According to Huang et al. [16] DSSM learns from a query or a document which can be used to compute semnatic siilarity. DSSM tarins itself by projecting semantically similar phrases taht 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 configuraion options the author has used two parameters namely *BATCHSIZE*, and *MAX_ITER*. *BATCHSIZE* refrs to how many trainign pairs can be used to train. *MAX_ITER* refers to the the total count of iterations DSSM will use to create the tarinign model.

Similar to Yih et al. [17] the author performed the follwoing actions to create the training model for each configuration:

- Shuffle the query pairs using the *WordHash* utility with the *shuffle* flag. Yieh et al. [17] performed a similar step to .
- Following Gao et al. [5] the authors generate the sequence of letter trigarm features using the *WordHash* utility with the *pair2seqfea* flag.
- The generated sequences of letter trigarm features are converted to binry files using the *WordHash* utility with the *seqfea2bin* flag.
- Next, the noise distribution of th training data is calculated using the *ComputelogPD* utility.
- Using the 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 these experiments the author has varied the tarining model with respect to DSSM configuration parameters as well as the documents that are using. Table I presents the experiemnts and the parametrs tht were changed. The experiemnts are referred by their anmes in Section IV. In each of these experiemnts, similar to Huang et al. [16]’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 [16]. In each of the experiemnt, the six projects are compared in a pair-wise manner i.e. each of the nine projects are comapred to the rest of the eight five of interest. The author followed the format of the provided example that comes with the Sent2Vec utility[5] to compare the similarity between two porjects. The author splitted the antural langages tokens of the two porjects 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

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

TABLE I
EXPERIEMNTS 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

to the tokens in bucket#1, for both projects. teh process was reapted for each combination of projects, and as a result, 30 input files were created.

The 'Training Source' column in Table I correspomds to the repository of how the DSSM model will be trined. For example, in Experiemnt-1, and Experiemnt-3, the DSSM model will be trained by the natural language tokens from the largest and the smallest repositories fo the six software projects that are used in the study.

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 i.e. the median of all the scores for each line in the input file was used to determine the semantic simialrity between the two projects of interest. For example, if the semantic simialrity 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 simialirty is measured as 0.21, according to the proposed approach.

IV. RESULTS

The author describes relevant empirical fidnngs in this section.

A. Software Repositories

Table II presents the respoitories 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 respository after performing the filtering steps. According to Table II the largest and the smallest repsotiroies used in the study are ShareX, and SeeWeather respectively. The filtered natural language tokens obtained from ShareX, and SeeWether are used to create two different training models fro DSSM. The goal of creating two different traaining models is to see if inclusion of tokens might have an impact of similarity measure. The scripts used to create the training DSSMs and used to execute Sent2Vec utility for Experiemnt-1, 2, 3, and 4 are avaialble online [12]. The file that are used as input to the Sent2Vec utility is availabile online [13]. The output of the Sent2Vec utility for each of the input files are also avialble online [14]. A sample snapshot of one of the input files to Sent2Vec and the correspondng output is prrovided in Figure 2(a), and Figure 2(b) respectively.

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	—

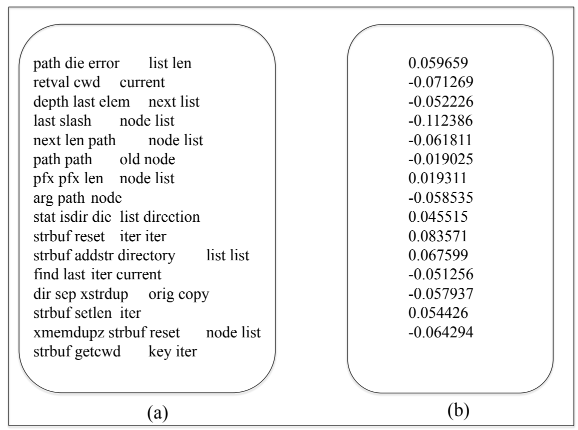

Fig. 2. Sample Input and Output Used for Sent2Vec

B. Similarity

As stated before in Section III the author used two different trainign models as well as used different values for the two aprameters of DSSM namely *MAX_ITER* and *TARGET_LAYER_DIM*. The combination fo these settings led to different experiemnts and the fidnings from each of these expriemnts are presented in Table III, **??**, IV, and **??** respectively for Expriment-1, 2, 3, and 4.

According to Table III the most similar projects are ShareX and RxJava as they have the highest semantic simialrty scores. Interestingly, ShareX si written in C#, and RxJava is written in Java. Please recall that the semantic simialirty score is the median of all the semantic scores between the natural language tokens of the two corresponding software projects that are slitted 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.

[12]https://github.com/akondrahman/Miscellaneous/tree/master/OpenSourceChallenge/scripts
[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	—

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

- H_0: The semantic simialirty scores for Experiemnt-1 and Experiment-3 are not different.
- H_1: The semantic simialirty scores for Experiemnt-1 and Experiment-3 are different.

With 99% statistical confidence the author has observed that the semantic similairty scores between Experiemnt-1 and Experiemnt-3 are not statistically different. Furthermore, the author has applied more hypothesis tests to observe if the semantic simialrity scores obtained in each experiemnt is different to that of the others. The author has observed that there are no signficant differences between the seamntic simialirty scores of Experiemnt-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 include the findings in this paper.

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

- DSSM and Sent2Vec has the potential tocapture ssemantic similarity between two software projects that are written in different programming languages. For example, the highest simialirty score in all experiemnts were between SchareX and RxJava.
- Number of iterations to train DSSM did not have an impact on the semantic similairty between software projects as the semantic score obtained from all experiemnts are not significantly different.
- The natural language tokens obtained from two different repositories did not have an impact on the semantic similairty score as the semantic score obtained from all experiemnts are not significantly different.

V. DISCUSSION

In this section we discuss our findings by stating the implications of our 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 repsoitories. Naviagting through multiple software repositories to identify one or multiple components for re-use can be non-trivial as the process involves cognitive effort [18] and time [4]. An automated software tool that idnetifies similar software components within the software team, as well as in a open source software repositoory hub such as Github or Sourceforge[15], can save software devlopers' time and effort.

B. Software Development in a Distributed Environment

In many organizatiosn software teams are geo-graphically distributed. For the purpose of software r-usability, co-ordinating amongst such distributed teams can be non-trivial as these teams might be in different timezones. An automated tool that captures semantic similarity across projects can be helpfulin thsi regard. By-passign 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

Seamntic 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 compoennet B might be susceptible to vulnerabilties. Such estimation can help software teams to apply a pro-active approach in containing software vulnerabilities. For example, after developing a certain compoennt C, a software team can invesitgate how simalr it is to a known vulnerable software compoennt. If the team finds strogn similairty then that can be treated as an indication of compoent C having vulnerabilties. Such approach might reduce the probability of releasing vulnerable software components to end-users.

D. Intelligent IDE Design

Effective similarity measurement technques between softar projects can revolutionize code search features exsiting 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 organization softare teams can be geographically distributed, and can have large amoutn fo software repositories. In such conditions semantic-base similairity apporaches can help to narrow down the search space for IDEs to recommend the relevant software compoenents. For example, when the deevloper types the word crypto, the IDE might dsipaly the releavnt package that has the implementation of cryptographic methods located in a remote repsotiory. The IDE can also display multiple components based on a suitable rankign measure.

VI. CHALLENGES

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

- **Resovling CudaLib Dependency**: DSSM relies on the *CUDALib* dll to perform its computations. Unfortunately, the provided executable binary file, hosted on the website[16]

[15] http://sourceforge.net/
[16] http://research.microsoft.com/en-us/projects/dssm/

did not have that added that dependency. As a result, the author was getting a "DllNotFoundException". To resolbve this, the author first invesigated the source code of DSSM, and having observed DSSM's dll dependency on CUDALib, installed *CUDA Toolkit 7.5* and re-excuted the Visual Studi porject 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 contentts of the raw '.tsv' as well as the 'pairs.txt' files to understand the formattingfor input files.

VII. Limitations

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

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 i.e. the author has not demonstrated two projects that have higher similarity scores are actually similar.

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 granlarities such as namespaces, binaries, or classes might have a different impact on the simialrity scores between projects.

Selection of Software Repositories: The author has selected a small set of repsoitories for illstration of the methodology. The size of the these software repsoitories avried between 1 MB 100 MB. Further investigation is required to appropriately assess the capability of DSSm and Sent2Vec to quantiify semantic simialrity between software projects.

Use of Parameters: As shon in Section II and Section III the author has selected and used a specific set of parametres for training the DSSM models. Further exploration and experimantaion with allparameters of DSSM is needed to fully assess DSSM's capability of discovering semantic simialrity between software projects.

VIII. Conclusion

In this paper the author has described how Microsoft's Deep Semantic Simialrity Model (DSSM) and Sent2Vec can be used to identify and quanitfy semantic similarity between software projects. The auhtor has also discussed the future implications of this study and how future research in thsi area can help software teams to identify re-usable software compoennets with reduced effort. The author believes future academic and industry research in thsi area has the possibility of revolutionalizing standard organizational efforts in software enineering.

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] C. Srinivas, V. Radhakrishna, and C. G. Rao, "Software component clustering and classification using novel similarity measure," *Procedia Technology*, vol. 19, pp. 866 – 873, 2015, 8th International Conference Interdisciplinarity in Engineering, INTER-ENG 2014, 9-10 October 2014, Tirgu Mures, Romania. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212017315001255

[7] A. Kuhn, S. Ducasse, and T. Grba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230 – 243, 2007, 12th Working Conference on Reverse Engineering. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584906001820

[8] A. Kuhn, S. Ducasse, and T. Girba, "Enriching reverse engineering with semantic clustering," in *Reverse Engineering, 12th Working Conference on*, Nov 2005, pp. 10 pp.–.

[9] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806817

[10] 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.

[11] 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

[12] 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

[13] 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.

[14] 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.

[15] 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

[16] 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

[17] 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.

[18] 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