

Quantifying Semantic Similarity of Software Projects Using Deep Learning

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 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 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.

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 artifacts to achieve software reusability for example, architecture of the software, source code and of the software, and designs and documents 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 re-usable 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. Grechanik et al. [5] provided empirical evidence on how semantic-based search toolscan help software developers to find similar applications. 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.

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 how different combinations of DSSM 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: 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 of 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

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/>

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. [6] 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 provides the parameters that have been used in this paper:

- *MAX_ITER*: maximum number of iterations to train DSSM
- *TARGET_LAYER_DIM*: dimension of the hidden and output vectors of the target model

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 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 that are served as input

B. Semantic Similarity in Software Engineering

The paper is closely related to prior academic work that have investigated on how semantic similarity can be used to aid software developers and software teams. Merkl et al. [7] used artificial neural networks to find similar features between software projects. Srinivas et al. [8] used distribution of features through semantic similarity to cluster and classify software projects. Grechanik et al. [5] proposed and evaluated *Exemplar*, and provided empirical findings showing how semantic-based techniques can help to identify relevant applications for software developers. Kuhn et al. [9] used Latent Semantic Indexing (LSI) to group components of the software that are semantically similar. In another work Kun et al. [10] used semantic similarity to enrich the reverse engineering steps of software projects. Asuncion et al. [11] used semantic similarity to facilitate software traceability in large scale software projects. Kawaguchi et al. [12] proposed and introduced *MUDABlue* that uses semantic information of open source projects to categorize open source projects without prior information. Cubranic et al. [13] 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. [14] 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. [15] investigated how semantic features of software repositories extracted using Web Ontology Languages can be used to characterize a software's evolution. Jalbert et al. [16] leveraged on textual semantics of bug reports to identify duplicate bugs in order to reduce development cost. Marcus et al. [17] used latent semantic indexing to determine conceptually similar software components such as functions, files, and code segments that are duplicated across the software. Antoniol et al. [18] used information retrieval to investigate how connections between the source code and documentation can be discovered. Yao et al. [19] used a semantic-based approach to classify and extract necessary information from reusable 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 ??, 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 to semantic similarity. Keeping this assertion in mind, the authors have considered three different programming languages: C, C#, and Java.

The author selected 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 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 can be built on. 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 `_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 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. [20] 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 each other, and projecting semantically dissimilar phrases that are further from each other.

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. [21] 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. Yih et al. [21] performed a similar step to .
- Following Gao et al. [?] the authors 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 the 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 training model with respect to DSSM configuration parameters as well as the documents that are using. Table ?? 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 He et al. [?]’s approach the author has used Cosine similarity to quantify the vectors of tokens amongst two projects. In each of the experiments, the nine projects are compared in a pair-wise manner i.e. each of the nine projects are compared to itself, and the rest of the eight projects of interest.

IV. RESULTS

[?]

A. Software Repositories

B. Token Extraction

C. Similarity

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 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 [22] 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 Sourceforge⁵, can save 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, 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 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 proactive approach in containing 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 having vulnerabilities. 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 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 organization software teams can be geographically distributed, and can have large amounts 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.

⁵<http://sourceforge.net/>

VI. 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 website⁶ did not have that added dependency. As a result, the author was getting a "DllNotFoundException". 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.

VII. LIMITATIONS

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

Lack of Ground Truth: In Section IV the author has shown 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 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 to 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.

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

⁶<http://research.microsoft.com/en-us/projects/dssm/>

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] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshvanyk, and C. Cumby, "A search engine for finding highly relevant applications," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 475–484. [Online]. Available: <http://doi.acm.org/10.1145/1806799.1806868>
- [6] 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.
- [7] D. Merkl, A. M. Tjoa, and G. Kappel, "Learning the semantic similarity of reusable software components," in *Software Reuse: Advances in Software Reusability, 1994. Proceedings., Third International Conference on*, Nov 1994, pp. 33–41.
- [8] 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>
- [9] 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>
- [10] A. Kuhn, S. Ducasse, and T. Grba, "Enriching reverse engineering with semantic clustering," in *Reverse Engineering, 12th Working Conference on*, Nov 2005, pp. 10 pp.–.
- [11] 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>
- [12] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Mudablue: An automatic categorization system for open source repositories," *Journal of Systems and Software*, vol. 79, no. 7, pp. 939 – 953, 2006, selected papers from the 11th Asia Pacific Software Engineering Conference (APSEC2004). [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0164121205001822>
- [13] 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.
- [14] 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>
- [15] C. Kiefer, A. Bernstein, and J. Tappelet, "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>
- [16] 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.
- [17] A. Marcus and J. I. Maletic, "Identification of high-level concept clones in source code," in *Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on*, Nov 2001, pp. 107–114.
- [18] 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.
- [19] 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>
- [20] 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>
- [21] 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, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, 2015, pp. 1321–1331.
- [22] 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>