# Exploration and Mining of Source Code Level Traceability Links on Stack Overflow

András Kicsi[1], Márk Rákóczi[1] and László Vidács[2]
[1]*Department of Software Engineering, University of Szeged, Szeged, Hungary*
[2]*MTA-SZTE Research Group, University of Szeged, Szeged, Hungary*

Keywords: Traceability, Testing, Test-to-Code.

Abstract: Test-to-Code traceability is a valid problem of software engineering that arises naturally in the development of larger software systems. Traceability links can be uncovered through various techniques including information retrieval. The immense amount of data shared daily on Stack Overflow behaves similarly in many aspects. In the current work, we endeavor to discover test-to-code connections in the code shared and propose some applications of the findings. Semantic connections can also be explored between different software systems, information retrieval can be used both in cross-post and in cross-system scenarios. The information can also be used to discover new testing possibilities and ideas and has the potential to contribute to the development and testing of new systems as well.

## 1 INTRODUCTION

Software testing is an integral part of software engineering and is considered invaluable in most cases in both the development and maintenance phase. Proper testing practices are well defined and can greatly contribute to the quality of a software product. As tests are written routinely, larger software systems can accumulate tens of thousands of test cases. Lacking proper documentation, the vast amount of tests can results in difficulties the maintenance as for example during bug localization we have to possess adequate information on what the test is meant to assess. This problem is called test-to-code traceability which already has a well-established literature. Coding practices like maintaining good naming conventions can make this problem nearly a trivial task, but in reality, most systems lack this kind of foresight in advance.

Stack Overflow is currently the most popular question and answer website throughout software development communities and is commonly considered an invaluable asset for developers. It represents an immense amount of data that is freely accessible via the internet. Users frequently share not just textual questions and answers but also code snippets that contain whole methods, stack trace messages, and other valuable information. It is also not uncommon to find the exact same code in a faulty form as part of a question and in its correct form given by an answer. Different parts of the same software can also appear in different questions or answers, which means that we could even find several valid traceability links between posts, or even at different questions.

SOTorrent (Baltes et al., 2019) is a publicly available dataset assembled for a mining challenge that investigated the exploration possibilities inside this vast amount of data. The dataset contains extensive data on ten years of Stack Overflow activity between 2008 and 2018 that encompasses 40.606.950 questions and answers in total with version information that include hundreds of millions of versions. The goal of the dataset is to provide structured access to Stack Overflow post data and versions. Although the origin software system is not always identifiable and the valid traceability links are not noted explicitly in the data, it could still provide ample opportunities for exploration and experimentation with different approaches even from a traceability aspect.

The paper is structured as follows. Section 2 introduces our approach and Section 3 describes the techniques used in computing semantic similarity between methods. Section 4 displays the results and provides brief discussion while Section 5 overviews related literature. The paper concludes with Section 6.



Kicsi, A., Rákóczi, M. and Vidács, L.
Exploration and Mining of Source Code Level Traceability Links on Stack Overflow.
DOI: 10.5220/0007981003390346
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 339-346
ISBN: 978-989-758-379-7

## 2 GOALS AND METHOD

We decided to approach this data from a test-to-code traceability point of view with an exploratory mindset and strive to uncover some of the traceability links that lie in the code published on Stack Overflow. Our previous experiments (Csuvik et al., 2019; Kicsi et al., 2018) deal with software code handled as text as an information source for traceability research. In the current work, we explore the depths of Stack Overflow using similar assets, working with Doc2Vec and Latent Semantic Indexing (LSI).

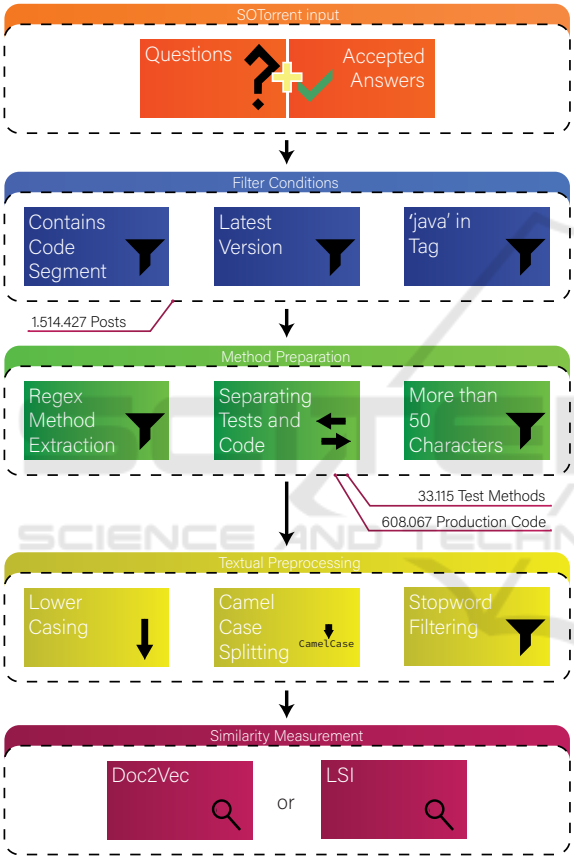


Figure 1: Steps taken in processing the Stack Overflow posts.

Figure 1 shows how the data went through several preparation and filtering steps. The input itself was the question and accepted answer of each Stack Overflow question in SOTorrent. Firstly, since in the current scope we only work with the information in software code we filter out the posts without any code segments. This is an easy task since code segments are labeled within SOTorrent itself. We decided to only consider the latest version of each post since various versions do not have a high likelihood of greatly aiding traceability. In the current experiment we decided to work with Java code only, thus we limited the posts to the ones containing *java* in their tags. These rather generous filtering methods still produced more than 1.5 million posts to consider, and each post could still contain several code segments wich can even have multiple methods. For the extraction of these methods, we used regular expressions that rely on the typical structure of a Java method, thus also filtering out stack trace segments, system messages, and various code segments written in other languages. The extracted methods were split into two categories, we considered a method either a test case or production code. A method was recognized as a test case if it contains either an assert statement, a JUnit annotation, or has *test* in its name. Overly short methods seemed to cause a high amount of noise in the results because of their high resemblance to each other, thus we also filtered out the methods that contained no more than 50 characters. These steps still resulted in tens of thousands of test methods and hundreds of thousands of production methods. We also applied textual preprocessing with lower casing, camel case splitting using regular expressions and stopword filtering to filter out several words that are highly common in Java. The resulting text was the input of similarity measurement conducted either by LSI or Doc2Vec.

Most of the traceability links discovered in Stack Overflow are nearly impossible to validate with total certainty, in most cases we can only assess whether a test case *seems* to aim to test a production code method. As we rely on semantic similarity, it is highly likely that we would get at least semantically similar pairs. Opposed to the traditional scenario of test-to-code traceability, easing the maintenance of systems is not a realistic goal here. Rather we adopted a more exploratory point of view and also did experiments with data mining in mind.

Stack Overflow contains huge amounts of software code potentially useful for developers. The topic of testing is not usually distinguished in the questions. Although some questions also contain *test* or *testing* in their tags, this is not mandatory for testing related questions. Furthermore, not every post that contains tests considers testing as its main topic. With semantic similarity, we could mine the testing posts related to our current interests more reliably than a simple search. The similarity measure can be applied on any text defined by a user which can be formatted as simple natural language text (for example "admin user login database") or whole methods for which we could either ask the most similar test cases or a test that should be very similar to what could test the method. This can be useful for gathering ideas on what to test our methods for as well as provide insights on how

```
@Test
public void testHandleRequestView() throws Exception {
  HelloController controller = new HelloController();
  ModelAndView modelAndView = controller.handleRequest(null, null);
  assertEquals("hello.jsp", modelAndView.getViewName());
}
```

Listing 1: An example test case.

```
public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
   logger.info("Returning hello view");
   return new ModelAndView("hello.jsp");
}
```

Listing 2: The production method found most similar to the code of Listing 3.

other developers do it. Since Stack Overflow represents the greatest current community database of such information, we consider this a worthwhile goal.

To summarize our current research goals, we propose the following research questions:

***RQ1:*** *While the current method is not the conventional use of traceability, can it still produce valid traceability links?*

***RQ2:*** *To what extent are the semantic connections recoverable from source code with different approaches?*

# 3 BACKGROUND

As our previous work (Csuvik et al., 2019) established, both LSI and Doc2Vec can have a valid place in uncovering traceability links from source code handled as a source of semantic information. Here we provide a brief overview of these methods. We used the Gensim (Rehurek and Sojka, 2010) representation of both techniques in this task.

## 3.1 Doc2Vec

Doc2Vec was introduced by Google's developers (Mikolov et al., 2013) and can be considered an extension of Word2Vec commonly used in various machine learning approaches in recent years. It operates with vector representations of words that are transformed to a lower number of dimensions via neural networks. The hidden layer has fewer neurons than the input and output layers and the weights of the hidden layer provide the word embedding output we need. Doc2Vec differs from Word2Vec only in also adding a unique identifier for each document to the input layer, thus distinguishing different documents (like sentences, articles or software code methods) which permits a word to have a different meaning in different contexts. For our results shared in this paper, we set the vector size to 300 and the model was trained for 5000 epochs.

## 3.2 Latent Semantic Indexing

LSI (Deerwester et al., 1990) is an older technique which has been used as mainstream in many tasks of semantic analysis. It is often considered one of the base techniques of many software engineering research approaches that rely on information retrieval and it is not new in traceability either (Rompaey and Demeyer, 2009) (Kicsi et al., 2017) and even in test-to-code traceability it is used as part of the recent state-of-the-art techniques (Qusef et al., 2011). Similarly to Doc2Vec, it relies on semantic information of text handled as vectors and produces a more compact form of vectors that results in the semantically more similar documents obtaining more similar vectors. LSI uses singular value decomposition to achieve this task. For our results shared in this paper, we set the *num_topics* parameter to 400 by which Gensim sets the size of the intermediate matrix.

# 4 EXPERIMENTS AND DISCUSSION

In our previous research (Kicsi et al., 2018), we proposed dealing with the test-to-code traceability problem in a recommendation system manner for which we provided several arguments. In our current case, this seems even more established as there are a lot of differences from the regular problem. We have to consider several difficulties in talking about results in this unconventional context. Since of course SOTorrent, like real life software code, does not note traceability

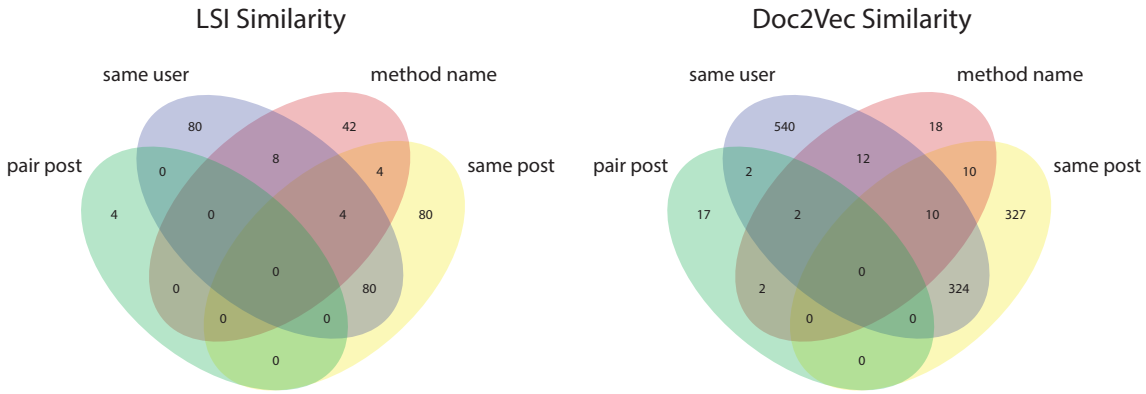


Figure 2: Results relying and similarity by LSI and Doc2Vec (top 1000 most similar methods, top 5 matches considered).

information, it is not an easy task to evaluate the results. We cannot really rely on proper naming conventions either for an automatized evaluation since Stack Overflow posts do not usually contain full qualified names either, only the names of the classes and methods the code which can have duplicates that can occur any number of times even within the same post. Thus, we do not think a full automated evaluation can be done in such a case. Worse yet, even manual evaluation seems less sure in these cases, since there is a myriad of examples of possibly valid test-to-code link pairs. While it would not happen in a software system, here it is still possible to have a test case that tests more than one production methods with the exact same properties but slightly different code. Some tested production methods could not even be accessible. A manual evaluation can look at the words and limited context of each method but lacking any other indicators it is still impossible for the majority of valid traceability links to be entirely sure. Furthermore, it is not necessary for every code segment to have valid software code, there are many examples where for the sake of keeping the code short and manageable, users substitute certain parts of the code with either a comment or simply natural language text.

Consider the example displayed in Listing 1, a test case from SOTorrent. We can be fairly certain that this test can test the proper functioning of the method displayed in Listing 2 taken from another post of another question, submitted by another user and not having a link to any GitHub repositories. There is no real guarantee, however, that the test case was meant to test this specific method. What we can assess is only that the method has the same name and number of parameters as the method under test.

There are some other insights, however, that can indicate that a test and production method are meant to belong together. Certain properties of a method, like the origin post, or the author can lead to more valid links. While these do not provide certainty as for example one user can submit any number of different test and production methods, they can still be used as a tool of assessment. To do this, we introduce the following sets:

- *Same User:* The pairs submitted by the same user.
- *Same Post:* The pairs that can be found in the same post.
- *Pair Post:* The pairs that can be found at the same question, but not the same post (we considered only the question and accepted answers as posts).
- *Method Name:* The pairs that seem to follow simple naming conventions, the name of the test case exactly matches the name of the production method with a "Test" added to the beginning or the end of the name.

The pairing of the test and code methods mentioned in the listings above are discovered both by LSI and Doc2Vec. Without being in any of the indicator sets, they were ranked as some of the most similar methods. Thus, we can provide an answer to our first research question.

**Answer to RQ1:** Both LSI and Word2Vec can produce pairs that seem highly accurate even for human observers, and even if they could not be easily recovered from the structural information of the dataset.

Since providing recommendations indeed seem a more viable case here than merely finding the single most similar element, we decided to consider the top 5 most similar methods for each query. Figure 2 displays how many of the pairs that scored in the top 1000 ranked by similarity of either LSI or Doc2Vec were found in our proposed indicator sets. We can see a huge difference in the performance of LSI and Doc2Vec but we have to note that a comparison here would not be valid. Firstly, the computation of Doc2Vec is much more demanding on this scale, so we decided to limit our exploration to the methods

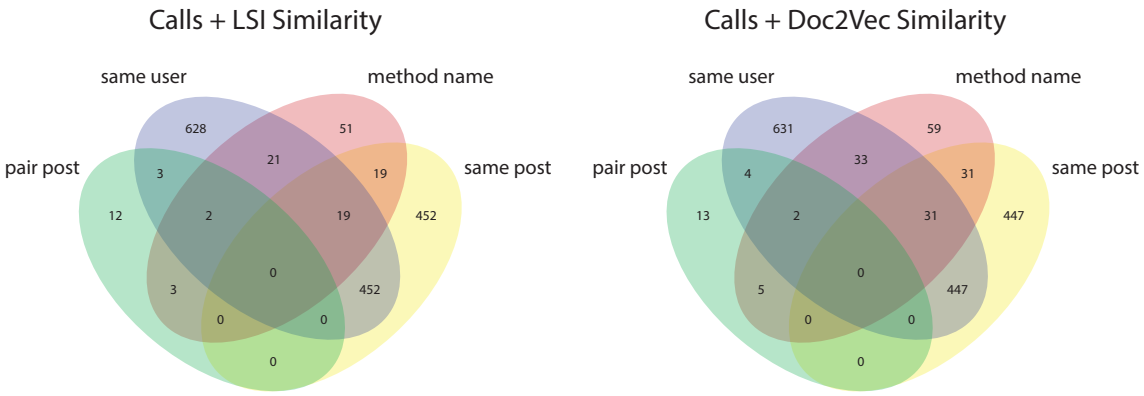


Figure 3: Results relying on called method names and similarity (top 1000 most similar methods, top 5 matches considered).

of only 100.000 of randomly chosen questions, which still took days to finish. LSI finished in about a day on the whole dataset (with the filtering described in Section 2), thus we included the methods of all questions. It is not really surprising that on a much larger dataset LSI scores less for example in finding the same user. Secondly, it should also be noted that these indicators do not necessarily resemble the quality of the results, for instance, detecting pairs authored by the same user can also stem from unique words or even coding style. Thirdly, as it is established, finding valid test-to-code traceability links is not our primary goal here, thus this does not evaluate our strategy as a mining technique.

Should we aim to mine real traceability links, we could work with a different approach. In test-to-code traceability relying on calls made by the test case is also one of the standard procedures. We could also do this, and consider only the production methods that are directly called from the test case. There is still room for several complications that do not permit this to be a foolproof approach. There are no qualified names and we do not have access to the call graphs either, thus this approach also seems very shallow. Considering a combination of this call matching technique with our most similar test-code pairs scores much higher on the indicator sets nevertheless, this can be seen in Figure 3. We can see that LSI here scores almost just as good in every set as Doc2Vec, even though at Doc2Vec still relies on 100.000 random questions while LSI considers every question. This probably means that call matching negates most of the advantage of the smaller set of Doc2Vec since the number of calls for each test case is not fewer in Doc2Vec's case either. Thus, we can see that with proper use, both LSI and Doc2Vec can contribute to finding test-to-code traceability links that seem genuine even on a larger scale.

**Answer to RQ2:** LSI and Doc2Vec both seem to produce a high number of test-to-code links that seem valid according to various indicators provided by the origin information stored in the dataset.

As our method aims to mainly serve as a mining technique, let us consider one more example. Aiming to implement a simple case of user login function we can quickly gather inspiration by using the query "user login" on our approach and getting a list of the semantically most similar methods, including the example given in Listing 3 which makes the necessary calls from a controller perspective for proper authentification. A user can survey any number of such methods and analyze them before creating his own. In case of wondering what LoginVO can be, it can also be submitted as a query and can be looked into. Additionally, we can also find the most similar tests to the "user login" query or either the code in the listing or one created by us and get a list of tests that were meant for a semantically similar problem. Thus, we believe our approach can be useful for software developers seeking to gather information from the vast amount of code of Stack Overflow.

## 5 RELATED WORK

Traceability in software engineering research typically refers to the discovery of traceability links from requirements or related natural text documentation towards the source code (Antoniol et al., 2002) (Marcus et al., 2005). Even as test-to-code traceability is not the most fashionable topic among link recovery tasks, there are several well-known methods that aim to cope with this problem (Rompaey and Demeyer, 2009). Test related traceability examples also can be found (Kaushik et al., 2011) (Rompaey and Demeyer, 2009) (Kicsi et al., 2018) (Qusef et al., 2011) (Csuvik et al., 2019), however no known perfect solution exists to the problem. In the research community serious

```
@Transactional
public LoginVO authenticateUser(LoginVO loginVO) {
    PmdUser pmdUser = loginMapper.getpmdUserFromLoginVO(loginVO);
    LoginVO loginVOFromDB = loginDAO.authenticateUser(pmdUser);
    if (loginVO.getUserName().equalsIgnoreCase(loginVOFromDB.getUserName())) {
        loginVO.setStatus(true);
    }
    return loginVO;
}
```

Listing 3: An example of the methods returned for the "user login" query by LSI.

attempts have been made at combating the problem via plugins in the development environment (Philipp Bouillon, Jens Krinke, Nils Meyer, 2007) or via static or dynamic analysis (Sneed, 2004) and textual analysis (Kicsi et al., 2018) (Csuvik et al., 2019). The current state-of-the-art techniques (Qusef et al., 2014) rely on a combination of diverse methods. In this work, we also took advantage of various textual similarity techniques, and the combination of these resulted in a promising recovery precision.

Word2Vec (Mikolov et al., 2013) gained a lot of attention in recent years and became a very popular approach in natural language processing. With this method, calculating the similarity between text elements became a mainstream process (Le and Mikolov, 2014) (Mathieu and Hamou-Lhadj, 2018) (Tufano et al., 2018) (White et al., 2016) (Ye et al., 2016) (Guo et al., 2017) (Yang et al., 2016) (Nguyen et al., 2017). Textual similarity is useful for example in the problem of clone detection (White et al., 2016). Doc2Vec is an extension of the Word2Vec method dealing with whole documents rather than single words. Although not enjoying the immense popularity of Word2Vec, it is still prominent to the scientific community (Zhu and Hu, 2017) (Dai et al., 2015) (Wang et al., 2016) (DeFronzo et al., 2015). In requirement traceability, researchers also made use of word embeddings to recover appropriate links (Guo et al., 2017) (Zhao et al., 2018) (Ye et al., 2016).

The valuable data provided by Stack Overflow is subject to numerous research papers including ones using textual approaches. For example, the discussed topics and the developers' interest is investigated using Latent Dirichlet Allocation (LDA) (Barua et al., 2014). Bazeli et al. (Bazelli et al., 2013) have investigated personality traits based on the reputation of the users. Ginsca et al. (Ginsca and Popescu, 2013) applied user profiling to determine whether it can be exploited to spot high-quality contributions. Automated tagging is one of the most tackled research questions. This includes classifying the questions into researcher defined categories (Beyer et al., 2018) or predicting the existing tags based on the text (Saini and Tripathi, 2018). Stack Overflow data contains a considerable amount of code snippets as well. Looking for good code examples, Nasehi et al. (Nasehi et al., 2012) found that besides the code snippet, the explanation is also an important factor. Wang et al. (Wang et al., 2015) address the API usability problem by analyzing Stack Overflow data with social media analysis and topic modeling. GitHub is a highly used resource of the developer community containing a wealth of open source projects. Since several developers look for example code on Stack Overflow, it is natural to investigate whether code snippets from Stack Overflow appear in GitHub and whether they are adapted (Yang et al., 2017).

In our paper, we utilize NLP methods to find links between code snippets targeting test and production code parts hidden from users who apply usual ways of information search.

## 6 CONCLUSIONS

In this paper, we explored the code shared on Stack Overflow, the most important Q&A site for developers. Our approach retrieved test-to-code traceability links in Stack Overflow data, but not in a usual way. We applied mining techniques and natural language processing methods to obtain links between code snippets. Contrary to usual link recovery methods that work within the scope of a development project, our intention was to reveal links upon common concepts. We showed that our method can discover traceability links that seem valid for human observers and that the approach can be tailored to produce valid traceability links even on a larger scale. Our main objective, however, was to show that our method can be a valid way of mining and exploring Stack Overflow from a testing point of view.

## ACKNOWLEDGEMENT

This research was supported in part by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002). The Ministry of Human Capacities, Hungary grant 20391-3/2018/FEKUSTRAT is acknowledged.

## REFERENCES

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. (2002). Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983.

Baltes, S., Treude, C., and Diehl, S. (2019). SOTorrent: Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets. In *MSR '19 Proceedings of the 16th International Conference on Mining Software Repositories*.

Barua, A., Thomas, S. W., and Hassan, A. E. (2014). What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*.

Bazelli, B., Hindle, A., and Stroulia, E. (2013). On the Personality Traits of StackOverflow Users. In *2013 IEEE International Conference on Software Maintenance*, pages 460–463.

Beyer, S., Macho, C., Pinzger, M., and Di Penta, M. (2018). Automatically classifying posts into question categories on stack overflow. In *Proc. of the 26th Conference on Program Comprehension*, pages 211–221.

Csuvik, V., Kicsi, A., and Vidács, L. (2019). Source code level word embeddings in aiding semantic test-to-code traceability. In *10th International Workshop at the 41st International Conference on Software Engineering (ICSE) – SST 2019*. IEEE.

Dai, A. M., Olah, C., and Le, Q. V. (2015). Document Embedding with Paragraph Vectors.

Deerwester, S., Dumais, S., and Landauer, T. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science and Technology*, 41(6):391–407.

DeFronzo, R. A., Lewin, A., Patel, S., Liu, D., Kaste, R., Woerle, H. J., and Broedl, U. C. (2015). Combination of empagliflozin and linagliptin as second-line therapy in subjects with type 2 diabetes inadequately controlled on metformin. *Diabetes Care*, 38(3):384–393.

Ginsca, A. L. and Popescu, A. (2013). User profiling for answer quality assessment in Q&A communities. In *Proc. of the 2103 workshop on Data-driven user behavioral modelling and mining from social media*, pages 25–28.

Guo, J., Cheng, J., and Cleland-Huang, J. (2017). Semantically Enhanced Software Traceability Using Deep Learning Techniques. In *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017*, pages 3–14. IEEE.

Kaushik, N., Tahvildari, L., and Moore, M. (2011). Reconstructing Traceability between Bugs and Test Cases: An Experimental Study. In *2011 18th Working Conference on Reverse Engineering*, pages 411–414. IEEE.

Kicsi, A., Tóth, L., and Vidács, L. (2018). Exploring the benefits of utilizing conceptual information in test-to-code traceability. *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pages 8–14.

Kicsi, A., Vidács, L., Beszédes, A., Kocsis, F., and Kovács, I. (2017). Information retrieval based feature analysis for product line adoption in 4gl systems. In *Proceedins of the 17th International Conference on Computational Science and Its Applications – ICCSA 2017*, pages 1–6. IEEE.

Le, Q. V. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents.

Marcus, A., Maletic, J. I., and Sergeyev, A. (2005). Recovery of Traceability Links between Software Documentation and Source Code. *International Journal of Software Engineering and Knowledge Engineering*, pages 811–836.

Mathieu, N. and Hamou-Lhadj, A. (2018). Word embeddings for the software engineering domain. *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*, pages 38–41.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems*, 2:3111–3119.

Nasehi, S. M., Sillito, J., Maurer, F., and Burns, C. (2012). What makes a good code example?: A study of programming q a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34.

Nguyen, T. D., Nguyen, A. T., Phan, H. D., and Nguyen, T. N. (2017). Exploring API embedding for API usages and applications. In *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017*, pages 438–449. IEEE.

Philipp Bouillon, Jens Krinke, Nils Meyer, F. S. (2007). EzUnit: A Framework for Associating Failed Unit Tests with Potential Programming Errors. In *Agile Processes in Software Engineering and Extreme Programming*, volume 4536, pages 101–104. Springer Berlin Heidelberg.

Qusef, A., Bavota, G., Oliveto, R., De Lucia, A., and Binkley, D. (2011). SCOTCH: Test-to-code traceability using slicing and conceptual coupling. In *IEEE International Conference on Software Maintenance, ICSM*, pages 63–72. IEEE.

Qusef, A., Bavota, G., Oliveto, R., De Lucia, A., and Binkley, D. (2014). Recovering test-to-code traceability using slicing and textual analysis. *Journal of Systems and Software*, 88:147–168.

Rehurek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50.

Rompaey, B. V. and Demeyer, S. (2009). Establishing traceability links between unit test cases and units under test. In *European Conference on Software Maintenance and Reengineering, CSMR*, pages 209–218. IEEE.

Saini, T. and Tripathi, S. (2018). Predicting tags for stack overflow questions using different classifiers. In *2018 4th International Conference on Recent Advances in Information Technology*, pages 1–5.

Sneed, H. (2004). Reverse engineering of test cases for selective regression testing. In *European Conference on Software Maintenance and Reengineering, CSMR 2004*, pages 69–74. IEEE.

Tufano, M., Watson, C., Bavota, G., Di Penta, M., White, M., and Poshyvanyk, D. (2018). Deep learning similarities from different representations of source code. *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*, 18:542–553.

Wang, S., Tang, J., Aggarwal, C., and Liu, H. (2016). Linked Document Embedding for Classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*, pages 115–124, New York, New York, USA. ACM Press.

Wang, W., Malik, H., and Godfrey, M. W. (2015). Recommending posts concerning api issues in developer q a sites. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 224–234.

White, M., Tufano, M., Vendome, C., and Poshyvanyk, D. (2016). Deep learning code fragments for code clone detection. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*, pages 87–98.

Yang, D., Martins, P., Saini, V., and Lopes, C. (2017). Stack overflow in github: Any snippets there? In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 280–290.

Yang, X., Lo, D., Xia, X., Bao, L., and Sun, J. (2016). Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports. In *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pages 127–137. IEEE.

Ye, X., Shen, H., Ma, X., Bunescu, R., and Liu, C. (2016). From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, pages 404–415, New York, New York, USA. ACM Press.

Zhao, T., Cao, Q., and Sun, Q. (2018). An Improved Approach to Traceability Recovery Based on Word Embeddings. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, volume 2017-Decem, pages 81–89. IEEE.

Zhu, Z. and Hu, J. (2017). Context Aware Document Embedding.