2-2019

# SIEVE: Helping developers sift wheat from chaff via cross-platform analysis

Agus SULISTYA

Gede A.A.P. PRANA

David LO
*Singapore Management University*, davidlo@smu.edu.sg

Christoph TREUDE

## Citation

SULISTYA, Agus; PRANA, Gede A.A.P.; LO, David; and TREUDE, Christoph. SIEVE: Helping developers sift wheat from chaff via cross-platform analysis. (2019). *Empirical Software Engineering*. 1-27. Research Collection School Of Information Systems.
**Available at:** https://ink.library.smu.edu.sg/sis_research/4499

**Noname manuscript No.**
(will be inserted by the editor)

# SIEVE: Helping Developers Sift Wheat from Chaff via Cross-Platform Analysis

**Agus Sulistya · Gede Artha Azriadi Prana · Abhishek Sharma · David Lo · Christoph Treude**

Received: date / Accepted: date

**Abstract** Software developers have benefited from various sources of knowledge such as forums, question-and-answer sites, and social media platforms to help them in various tasks. Extracting software-related knowledge from different platforms involves many challenges. In this paper, we propose an approach to improve the effectiveness of knowledge extraction tasks by performing cross-platform analysis. Our approach is based on transfer representation learning and word embeddings, leveraging information extracted from a source platform which contains rich domain-related content. The information extracted is then used to solve tasks in another platform (considered as target platform) with less domain-related contents. We first build a word embeddings model as a representation learned from the source platform, and use the model to improve the performance of knowledge extraction tasks in the target platform. We experiment with Software Engineering Stack Exchange and Stack Overflow as source platforms, and two different target platforms, i.e., Twitter and YouTube. Our experiments show that our approach improves performance of existing work for the tasks of identifying software-related tweets and helpful YouTube comments.

**Keywords** Word Embeddings, Transfer Representation Learning, Software Engineering

Agus Sulistya, Gede Artha Azriadi Prana, Abhishek Sharma and David Lo
Singapore Management University
E-mail: {aguss.2014,arthaprana.2016,abhisheksh.2014,davidlo}@smu.edu.sg

Christoph Treude
University of Adelaide
E-mail: christoph.treude@adelaide.edu.au

## 1 Introduction

Software developers rely on many sources of knowledge to help them keep up-to-date on the latest technologies and to solve their problems. A study conducted by Maalej et al. [29] shows that 70 percent of developers use online resources (web search engine, public documentation, forums) as channels to access knowledge. Among those channels, some are more popular and contain richer content relevant to software engineering compared to others. For example, Xin et al. [73] conducted an observation on how developers commonly make use of a web search engine such as Google to find online resources to improve their productivity. They found that 63% of the searches on the Internet ended up with a visit to Stack Overflow, a popular question and answer (Q&A) site.

Despite the popularity of Stack Overflow, software developers also seek knowledge in other platforms, such as microblogging platforms (e.g., Twitter). A large number of software developers use Twitter frequently to support their professional activities, e.g., to share and obtain the latest technical news [56]. Another growing popular knowledge source for developers is online video sharing platforms, such as YouTube. A study by MacLeod et al. [31] found that video is a useful medium for communicating knowledge between developers, and that developers build their online personas and reputation by sharing videos through social channels. Developers as content creators will also be able to digest feedback from valuable comments given by their viewers. It will help content creators to improve their future videos [42].

Extracting software-related knowledge from different platforms requires varying levels of effort and skill. For example, on Stack Overflow, almost all of the contents are related to software development. The content is also maintained to be of high quality by collective community effort and the site's moderators. But it is more challenging to extract useful information from tweets, due to the information overload problem in Twitter's space. Twitter is a popular social media channel with about 330 million users who produce about 500 million tweets daily [64]. Singer et al. found that Twitter is popular among software developers also, who use it to *keep up with the fast-paced development landscape* [56]. Around 70% of the respondents said that they use Twitter to stay current about the latest technologies, practices, and tools, and also to learn about things that they are not actively looking for. A number of them also said that they use Twitter for community building especially around their development projects. They also found that developers face challenges while using Twitter, which relate to dealing with a huge amount of irrelevant tweets produced on Twitter, as well as the challenge of maintaining a relevant network. With a huge amount of content being produced by a large number of users, developers face a hard time in finding tweets with information relevant to software development. Twitter is a channel which is very noisy with information and users from domains other than software engineering. This gives rise to the problem of information overload for software developers who use Twitter.

In this paper, we propose `SIEVE`, an approach to utilize contents from a rich software-development-specific platform to help automated knowledge extraction tasks in other less software-development-specific platforms, based on a transfer representation learning approach. We consider two platforms, Software Engineering Stack Exchange and Stack Overflow, as the rich domain-related platforms. We build word embeddings based on the dataset collected from these platforms. We then leverage the word embeddings vectors to solve information retrieval and classification problems in two different target platforms. We experiment with two different use cases: finding tweets relevant to software development on Twitter [52], and classifying useful comments for software engineering video tutorials on YouTube [42]. We conducted experiments based on the existing golden datasets provided by Sharma et al. [52] for Twitter, and Poché et al. [42] for YouTube comments. Our experiments show the effectiveness of our proposed cross-platform analysis approach which achieves performance improvements of up to 23% and 10.3% for the first and second use case respectively. Our contributions can be summarized as follows:

1. We propose an approach based on transfer representation learning and word embeddings to solve information retrieval problems on how to use data from domain-specific platforms to help tasks in other platforms.
2. We conduct experiments to show the effectiveness of the proposed approach for two different tasks and platforms (i.e., Twitter and YouTube), and use baselines described in existing work.

The next sections in this paper are structured as follows. In Section 2, we describe background related to knowledge channels for software developers, and background on representation learning and word embeddings. In Section 3, we describe our approach on learning a knowledge representation from source platforms. We present our first use case on finding software-related tweets in Section 4. Next, we present the second use case on classifying informative comments on YouTube in Section 5. Threats to validity are discussed in Section 6. We describe related work in Section 7. Finally, we conclude and mention future work in Section 8.

## 2 Background

In this section, we first discuss the knowledge sources used by developers which we have considered in our current work. Next, we discuss some background on transfer representation learning and word embeddings.

### 2.1 Knowledge Sources for Software Developers

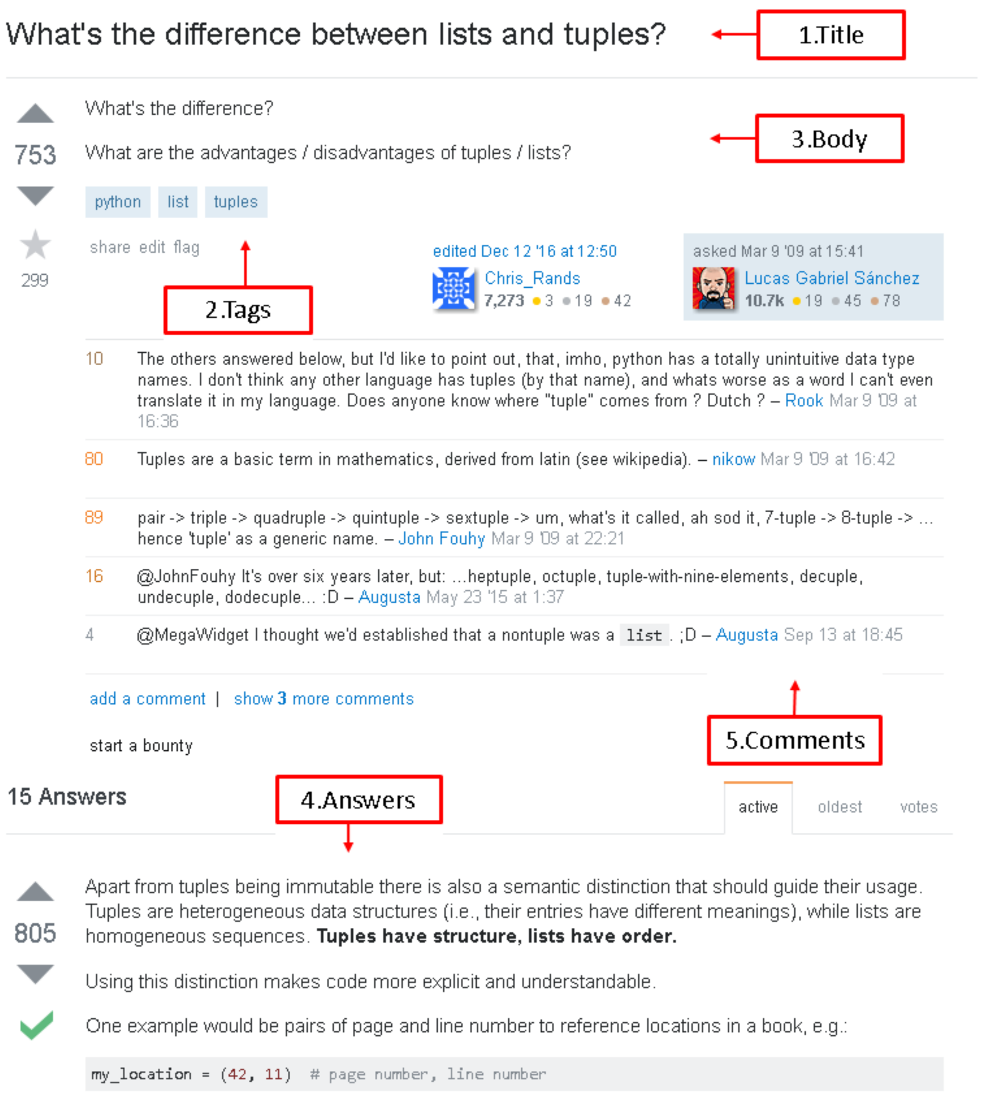
Storey et al. found that software developers use many communication tools and channels in their software development work [59, 60]. In our current work we focus on learning word embeddings from software-development-specific channels such as Software Engineering Stack Exchange and Stack Overflow (which are popular software discussion forums), and use the learned embeddings to improve the performance of information retrieval and classification tasks related to the extraction of software-development-related knowledge from open domain channels such as Twitter (a microblogging site) and YouTube (video sharing). In the subsequent paragraphs we give background on these channels.

*Software Engineering Stack Exchange*: Stack Exchange[1] is a network of question and answer (Q&A) websites, where each website focuses on a specific topic. On any of the websites each of which is related to a particular domain, its users can ask questions related to that domain and other users can provide answers to these questions. The motivation for users to answer questions comes from the points that they can gain when other users in the same community upvote or accept their answers. These points help them to build a reputation in the domain (and the related community), which the Stack Exchange website is focused on. The Stack Exchange community has been the focus of many studies such as [7, 49]. In this work as we are interested in improving the performance of information retrieval and classification tasks related to software engineering, we focused on Stack Exchange communities focused on software engineering and programming, which are Software Engineering Stack Exchange[2] and Stack Overflow[3] respectively. The difference between these two sites is that *Stack Overflow* is focused only on specific programming tasks and problems, whereas *Software Engineering Stack Exchange* allows more general questions related to software development and engineering such as discussions about various libraries, methodologies etc. The latter has about 50,655 questions and 260,361 users. The intuition behind using Software Engineering Stack Exchange is that models trained on the general nature of content may achieve different performance on the task of filtering information from open domain websites such as Twitter and YouTube.

*Stack Overflow*: Stack Overflow[3] is a programming question and answer website founded in 2008 with a focus on software development. It is an online forum where anybody facing a programming issue can post a question describing the problem they face. The questions posted are public on the forum, so any other user on the forum can post their solutions as answers to the posted questions. The original asker can then mark an answer as accepted if it solved the problem. Other users can also upvote an answer if they think it is the right method to solve the programming challenge being addressed. Thus Stack Overflow helps developers in getting answers to their problems with the help of the crowd. It is one of the most used websites by software developers in the world having more than 9,000,000 registered users, more than 16,000,000 questions and an Alexa Rank of 70[4]. As Stack Overflow contains rich software development and software engineering content, it has been immensely popular among software engineering researchers in recent years, where it has been used

[1] https://stackexchange.com/
[2] https://softwareengineering.stackexchange.com/
[3] https://stackoverflow.com/
[4] https://en.wikipedia.org/wiki/Stack_Overflow

to discover topics and trends [6], generate API call rules [3], explore knowledge networks [77], build information filtering models [52] etc. More related work is discussed in detail in Section 7.



What's the difference between lists and tuples?

753

What's the difference?

What are the advantages / disadvantages of tuples / lists?

python list tuples

share edit flag

299

edited Dec 12 '16 at 12:50
Chris_Rands
7,273 ●3 ●19 ●42

asked Mar 9 '09 at 15:41
Lucas Gabriel Sánchez
10.7k ●19 ●45 ●78

10 The others answered below, but I'd like to point out, that, imho, python has a totally unintuitive data type names. I don't think any other language has tuples (by that name), and whats worse as a word I can't even translate it in my language. Does anyone know where "tuple" comes from ? Dutch ? – Rook Mar 9 '09 at 16:36

80 Tuples are a basic term in mathematics, derived from latin (see wikipedia). – nikow Mar 9 '09 at 16:42

89 pair -> triple -> quadruple -> quintuple -> sextuple -> um, what's it called, ah sod it, 7-tuple -> 8-tuple -> ... hence 'tuple' as a generic name. – John Fouhy Mar 9 '09 at 22:21

16 @JohnFouhy It's over six years later, but: ...heptuple, octuple, tuple-with-nine-elements, decuple, undecuple, dodecuple... :D – Augusta May 23 '15 at 1:37

4 @MegaWidget I thought we'd established that a nontuple was a `list`. ;D – Augusta Sep 13 at 18:45

add a comment | show **3** more comments

start a bounty

15 Answers

active oldest votes

805

Apart from tuples being immutable there is also a semantic distinction that should guide their usage. Tuples are heterogeneous data structures (i.e., their entries have different meanings), while lists are homogeneous sequences. **Tuples have structure, lists have order.**

Using this distinction makes code more explicit and understandable.

One example would be pairs of page and line number to reference locations in a book, e.g.:

```
my_location = (42, 11)  # page number, line number
```

**Fig. 1** A Sample question-answer-thread on Stack Overflow with tags (Thread ID 626759)

Figure 1 shows a sample question-and-answer thread from Stack Overflow. Each thread generally contains five types of information: title, tags, body, answers, and comments. The title of a thread is a summary of the question asked. The tags represent the metadata related to the question being asked and are entered by the person who asked the question. Whenever somebody asks a question on Stack Overflow, they receive a recommendation to attach at least three tags to the question. The body part of the thread contains the description of the question. Whenever a question is answered, the answer appears in the answers section of the thread. Other developers can also ask further clarifying questions or comment on the question or answers posted up to that point.

*Twitter and Software Engineering*: Twitter is currently one of the most popular microblogging sites in the world. On Twitter, a user can post short messages (a.k.a. *tweets*) broadcasted to all other Twitter users who are following the user. Twitter allows a user to *follow* another user, which means the latter subscribes to all the tweets of the user he/she is following. Users also have an option of reposting the tweets posted by others – an activity known as *retweeting*. Twitter also allows users to mark favorite tweets, which conveys their interest in the content of a tweet.

By virtue of its simple design and easy-to-use functionality, Twitter has become a powerful medium for information sharing and dissemination. It started as a social networking medium but has nowadays become one of the important sources of information for people to keep up-to-date with the latest news and information about their domains of interest, to share and promote knowledge, and to keep in touch with their family and friends [26]. Twitter influences many communities including the software engineering community as highlighted by many prior studies [56,9,71,61]. Various techniques have been proposed recently to mine software engineering relevant information from Twitter [52,54,72,23].

*YouTube and Software Engineering*: YouTube is a website where anybody can share videos [17]. It has over 1 billion users and generates billions of views daily [79]. YouTube has also evolved into a knowledge sharing resource, where people can share informational videos, follow other users and comment on videos. Thus it provides people with resources to share information, learn new knowledge, as well as get and provide feedback.

Software developers also use YouTube for sharing information as well as learning [31,44]. MacLeod et al. found that developers share videos detailing information they wished they had found earlier [31]. The videos mainly relate to sharing knowledge about development experiences, implementation approaches, design pattern application, etc. Other work focuses on extracting relevant information for developers from YouTube, which is a challenging task given the large size of videos. Tools to help developers find relevant content from software engineering videos have been proposed in [45,75]. Poché et al. analyzed user comments related to software engineering videos posted on YouTube [42] and proposed a technique for finding relevant comments.

### 2.2 Transfer Representation Learning and Word Embeddings

Representation learning can be described as learning representations of data that make it easier to extract useful information when building classifiers or other predictors [8]. In the field of Natural Language Processing (NLP) applications, distributed word representations are one of the applications of representation learning. Distributed word representations, i.e., word embeddings, have been widely applied in various text mining and natural language processing tasks.

Word embeddings represent words in a low dimensional continuous space, to convey semantic and syntactic information [28]. One of the most popular word embedding techniques is Word2Vec, which uses a shallow neural network to reconstruct contexts of words. Mikolov et al. [32,33] proposed two word embedding models Continuous Bag-of-Word (CBOW) and the skip-gram model which have been widely adopted due to their effectiveness and efficiency. For CBOW, a neural network is trained to predict a word based on its surrounding words. In this architecture, the continuous value vector for a word is the vector that is input to the last layer in the network after we input its surrounding words to the network. For skip-gram, a neural network is trained to predict surrounding words based on the current word. In this architecture, the continuous value vector for a word is the vector that is output by the first layer in the network. It has been shown that the embedding vectors produced by these models preserve the syntactic and semantic relations between words under simple linear operation. For example, the resultant vector of the following arithmetic operation (vector of brother - vector of man + vector of woman) is similar to the vector of sister. This is related to analogical reasoning where brother is to sister as man is to woman, which is encoded in the vector representation learned by Word2Vec.

In machine learning, many methods perform well under the common assumption that the training and test data are drawn from the same feature space and the same distribution. In many contexts, this assumption may not hold. For example, we attempt to solve a classification problem in a domain that does not have enough training data, but we have sufficient data in other related domains. In this case, knowledge transfer or transfer learning would be useful to solve the classification problem [38]. In the context of representation learning, transfer representation-learning is where rich representations are learnt in a source platform with the aim of transferring them to different target platforms [1].

## 3 Representation Learning from Sofware-Development-Specific Platforms

Our work is related to transfer representation-learning, where rich representations are learnt from a software-development-specific platform, and leveraged in a different target platform. To represent knowledge in the source platform, we build a word embedding model that represents each word as a low-dimensional vector such that words that are similar in meaning are associated with similar vectors. Word embedding models have successfully been applied in various natural language processing (NLP) tasks, such as in [78, 74, 16].

A recent finding by Mou et al. [35] shows that the transferability of neural NLP models depends largely on the semantic relatedness of the source and target tasks. Therefore, since our target tasks are related to the extraction of knowledge relevant to software development, we need to define a source platform that contains rich software engineering content. In this work, we choose two source platforms (Software Engineering Stack Exchange and Stack Overflow), and compare the performance of models built from the two platforms.

Our approach consists of two stages: representation learning from a source platform, and model building for a target platform. In the first stage, we learn a word embedding representation from a software-development-specific platform. In the second stage, we leverage the word embedding model built from the platform to resolve tasks in the target platform. Figure 2 shows the overall architecture of our proposed framework.

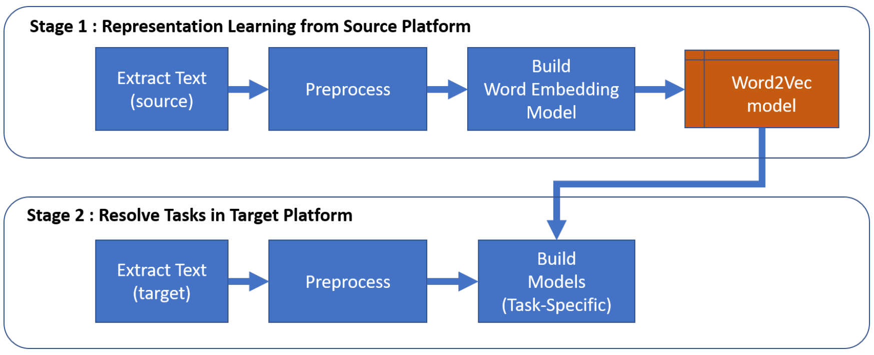

Stage 1 : Representation Learning from Source Platform
Extract Text (source)
Preprocess
Build Word Embedding Model
Word2Vec model
Stage 2 : Resolve Tasks in Target Platform
Extract Text (target)
Preprocess
Build Models (Task-Specific)

**Fig. 2** Overall approach

We describe each stage of our proposed approach as follows:

**Stage 1: Representation Learning from Source Platform**

While most research done on Q&A sites is based on Stack Overflow data (e.g. [52, 80]), we believe that Software Engineering Stack Exchange (StackExchange-SE) is also a good source for software engineering related terms. Therefore, we use text data extracted from the two sites, and build two different models: `SIEVE_SO` which is based on *Stack Overflow* and `SIEVE_SE` which is based on *Stack Exchange* data.

The StackExchange-SE dataset is publicly available on the Stack Exchange data dump site.[5] We use the following two files: Posts.7z and Comments.7z. Posts.7z contains the title and body of posts (i.e., questions and answers)

[5] http://archive.org/download/stackexchange

that appear on Stack Exchange. Comments.7z contains comments that users give to the questions and answers on Stack Exchange. Our Stack Exchange dataset contains a total of 149,478 posts, 409,740 comments, and 46,246 titles generated in a time period spanning from September 2010 to August 2017. We combined all of the posts, comments and titles for learning word embeddings from this dataset.

We used the Stack Overflow data dump provided by previous work by Sharma et al. [52]. The data was also taken from the data dump site.[5] They extracted the questions and answers from the Posts.7z file, and user's comments from Comments.7z. These files contain content posted on Stack Overflow from September 2008 to September 2014. There are a total of 7,990,787 titles, 21,736,594 posts (questions and answers), and 32,506,636 comments. Since there are too many posts and comments to efficiently process the data, to reduce the time it takes to learn a model, they randomly selected 8 million posts and comments from the data dump. We use this randomly selected data and combine all of the posts, comments and titles.

Before we build the word embeddings model, we performed the following text preprocessing for both datasets:

1. Parse the posts into sentences, since we want to train word embeddings at sentence-level. We use NLTK's *punkt* tokenizer[6] for sentence splitting.
2. Remove all HTML tags since they do not contain useful information for word embeddings.
3. Remove all special characters (e.g., symbols, punctuations, etc.) and words that contain only numbers.
4. Change all words to their lower case.

We chose the continuous skip-gram Word2Vec model proposed by Mikolov et al. [32] We use the Word2Vec implementation in Gensim[7]. We set the parameters according to Mikolov et al. [32]: context windows size to 5, dimension to 300, batch size to 50, negative sampling to 10, minimum word frequency to 5 and iterations to 5. The output of the model is a dictionary of words, each of which is associated with a vector representation. Table 1 includes statistics on the generated word embeddings learned from the datasets.

**Table 1** Statistics of datasets and word embeddings extracted from Stack Overflow (`SIEVE_SO`) and StackExchange-SE (`SIEVE_SE`)

| | **StackExchange-SE** | **Stack Overflow** |
|---|---:|---:|
| Number of Posts+Comments | 605,464 | 8,000,000 |
| Number of Sentences (after preprocessing) | 1,884,959 | 5,007,411 |
| Size of Vocabulary in word vector | 232,953 | 275,103 |

### Stage 2: Model Building for Target Platform

[6] http://www.nltk.org
[7] https://pypi.org/project/gensim/

Our goal is to leverage knowledge extracted from software-development-specific platforms and apply it to open-domain platforms. In order to examine the learned word embeddings representation in stage 1, we utilize the word embeddings in two different use cases. In the first use case, we aim to resolve the task of finding tweets related to software engineering. In the second use case, we leverage the word embeddings to classify user comments on YouTube coding tutorial videos. We discuss each of the use cases further in the next sections.

## 4 Finding Relevant Tweets Using Word Embeddings

In this section, we show how our approach can be used for the task of finding tweets related to software engineering. Researchers have found that developers use Twitter to support their professional activities by sharing and discovering various information from microblogs, e.g., new features of a library, new methodologies to develop a software system, opinions about a new technology or tools, etc. [52] However, due to various topics posted on Twitter, it becomes a challenge to find interesting software-related information on Twitter. To overcome this problem, Sharma et al. [52] proposed a language-model based approach and used the model to rank tweets based on their relevance to software engineering. We will use the proposed model as a baseline, along with other baselines. We aim to answer the following research question:

**RQ1. How effective is our approach at the task of finding software related tweets?**

### 4.1 Approach

Figure 3 shows an instance of our proposed approach for the task of finding software development-related tweets, by utilizing word embeddings trained from a source platform. In general, we formulate the task of finding software-related tweets as a ranking problem, i.e., ranking the tweets in the order of their similarity scores with selected sentences from the source platforms. We follow these steps:

**Step 1: Instance Selection**
In our approach, selecting instances (i.e., sentences) from the source platform is an important task, since we will use these selected sentences to rank the tweets based on a similarity measure. Sentences extracted from the source platform (StackExchange-SE/Stack Overflow) are considered as software-related. However, some of the sentences may have different characteristics with Twitter. Therefore, we use the following heuristic methods to select suitable sentences from a source platform:

1. We select sentences that have a length of no more than 140 characters which corresponds to a tweet's maximum length.

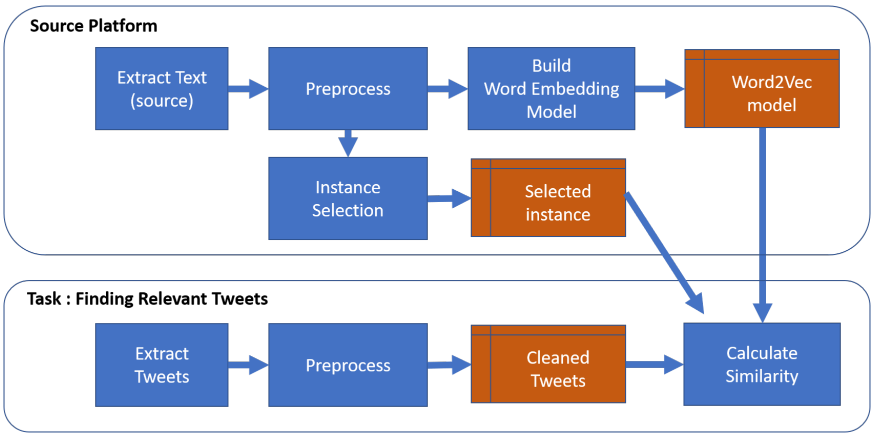


**Fig. 3** Our approach for finding software-related tweets

2. Among these selected sentences, we randomly sample sentences. By default, we sample 1000 sentences. We believe that this sampled set should be enough to represent sentences that contain software-related terms.

### Step 2: Preprocess Tweets

We use the Twitter dataset provided by Sharma et al. [52]. The tweets are preprocessed by removing punctuation marks and URLs, and all words are changed into lowercase.

### Step 3: Calculate Similarity

To measure similarity between tweets and selected sentences taken from the source platforms, we need to implement the same representation for both texts. Because sentences (or tweets) have different lengths, we need to use a fixed-length vector to represent them. To build the vector representation, we leverage the word embeddings learned from a source platform. The model consists of word vectors that have a dimension of 300 as mentioned in Section 3. We follow these steps:

1. For each sentence or tweet in the dataset, we tokenize it into words.
2. For each word, we look up its weight from the word embeddings model. If a word does not exist in the model, we can either ignore that word, use a vector whose values are all 0 to represent it, or use the average of the embeddings from words having the lowest frequency in the model. By default, we ignore the word that not exist in the model. The result is a 300-dimension vector of real values taken from the word embeddings model.
3. We represent the sentence into a fixed-length vector. There are different ways to obtain text representation from word embeddings. The most common methods use the maximum, minimum, or average of the embeddings of all words (or just the important words) in a sentence [57]. In this case, we take the average of the word embeddings of all words within the text,

following [25]. At the end, we have a word vector of real values with dimension of 300 for each tweet or sentence.

For each tweet, we calculate similarity between the vector representations of tweets and the vector representations of each of selected sentences in the source platform. We then rank the tweets based on their similarity scores. The higher the scores, the more likely the tweet contains software-related contents. To calculate similarity between two word vectors, we use cosine similarity. Cosine similarity is a measure of similarity between two vectors (in this case, vector of text representation) that measures the cosine of the angle between them. Given a tweet $T$ and a selected sentence $S$, that are represented by two word vectors $wv_{tweet}$ and $wv_{so}$, we define their semantic similarity as the cosine similarity between their word vectors:

$$similarity(T, S) = \frac{wv_{tweet}^T.wv_{so}}{||wv_{tweet}||||wv_{so}||}$$

## 4.2 Dataset and Baselines

**Dataset.** For the Twitter dataset, we use the same dataset used by Sharma et al. [52]. The dataset consists of around 6.2 million tweets downloaded through the Twitter REST API. To collect tweets, they first obtained a set of microbloggers that are likely to generate software-related contents. They started with a collection of 100 seed microbloggers who are well-known software developers. Next, they analyzed the follow links of these microbloggers to identify other Twitter accounts that follow or are followed by at least 5 seed microbloggers. After they had identified the target microbloggers, they downloaded tweets that were generated by these individuals. They then performed preprocessing on the collected tweets such as removing punctuation marks and URLs, and changed all words into lowercase.

**Baselines.** We used several baselines to show the effectiveness of our approach. First, we compared our proposed approach against NIRMAL [52], since we used the same dataset as their work. Next, since our models are trained on software-development-specific platforms, we compared the models with a within-platform model trained from the target platform (Twitter). To show the effectiveness of the Word2Vec-based models that we use, we compared the models with a model that uses Term Frequency – Inverse Document Frequency (td-idf) vectors generated from a source plarform. Tf-idf technique has been widely used in other software-engineering-related information retrieval tasks such as in [19, 37]. We briefly describe the baselines as follows:

1. **NIRMAL by Sharma et al.** We used this approach as the main baseline, since the approach is the state-of-the-art in the task of ranking software-related tweets. This approach builds an N-gram language model by using SRILM [58], a language modeling toolkit. NIRMAL learns a language

model from Stack Overflow data. NIRMAL then uses the learned model to compute the perplexity score of each tweet. The lower the perplexity score, the more likely the tweet is software related. NIRMAL then ranks the tweets in ascending order of their perplexity scores and returns a ranked list.

2. **Term Frequency – Inverse Document Frequency (*tf-idf*).** In this approach, instead of using vectors generated by word embeddings, we used td-idf vectors generated from a source plarform. We built two variants of tf-idf vectors: one from Stack Overflow posts, and one from Stack Exchange posts. Term frequency (tf) is the number of times a word occurs in a given sentence, accompanied with a measure of the term scarcity across all the sentences, known as inverse document frequency (idf). Before constructing the vectors, we performed stemming using Porter Stemmer [48] and removed English stopwords. To remove stopwords, we used stopwords listed in the Python NLTK library[8].
3. **Word2Vec trained on Twitter.** We consider this approach as a *within-platform* baseline, since we leverage knowledge extracted from Twitter itself as the target platform. We trained a skip-gram word embeddings model using the set of parameters advised by Mikolov et al. [32]: context windows size of 5, dimensions of 300, batch size of 50, negative sampling to 10, minimum word frequency of 5 and iterations of 5 - the same set of parameters that are used by our proposed approach.

## 4.3 Experiments and Results

**Experiments Setting.** We conducted experiments to answer RQ1 and evaluated the effectiveness of our approach as compared to the baselines. After following the steps in our proposed approach, we ranked the tweets based on similarity scores between the tweets and selected instances taken from a source platform. We investigated three different word embeddings models trained from Stack Overflow, StackExchange-SE and Twitter, and one non-word embeddings model (tf-idf).

As an evaluation metric, we used accuracy@K, which is defined as the proportion of tweets in the top-K positions that are software-related. We manually evaluated the top-K tweets ranked by their similarity scores. We asked two labelers who have master's degrees in Computer Science to manually label the tweets, either as *"relevant"* or *"not relevant"* to software engineering. For our final ground truth, we labeled a particular tweet as "relevant" only if both labelers agreed that the tweet is software-development-related. We used Cohen's Kappa to measure inter-rater reliability for the labeling task. We obtained a Kappa value of 0.78 for labeling `SIEVE_SE` and a Kappa value of 0.68 for labeling `SIEVE_SO` – following Landis and Koch's interpretation [27], this value indicates substantial agreement.

[8] https://www.nltk.org/

**Results.** The results of our experiments are shown in Table 2 and Figure 4. Overall, the word embeddings model trained on Stack Exchange performed best, except for accuracy@10, where the tf-idf based approach performed best. Word embeddings models trained on platforms that contain rich software-development-related knowledge (Stack Exchange and Stack Overflow) performed better as compared to the baselines.

**Table 2** Accuracy@K results of different approaches in our experiments (best results are in bold).

| **Approach** | **acc@10** | **acc@50** | **acc@100** | **acc@150** | **acc@200** |
|---|---|---|---|---|---|
| `Nirmal` | 0.900 | 0.820 | 0.720 | 0.707 | 0.695 |
| `TF-IDF (Stack Overflow)` | **1.000** | 0.540 | 0.730 | 0.693 | 0.575 |
| `TF-IDF (Stack Exchange)` | **1.000** | 0.560 | 0.400 | 0.347 | 0.310 |
| `Word2Vec (Twitter)` | 0.400 | 0.220 | 0.260 | 0.260 | 0.235 |
| `SIEVE_SO` | 0.900 | 0.880 | 0.870 | 0.847 | 0.800 |
| `SIEVE_SE` | 0.900 | **0.980** | **0.970** | **0.940** | **0.925** |

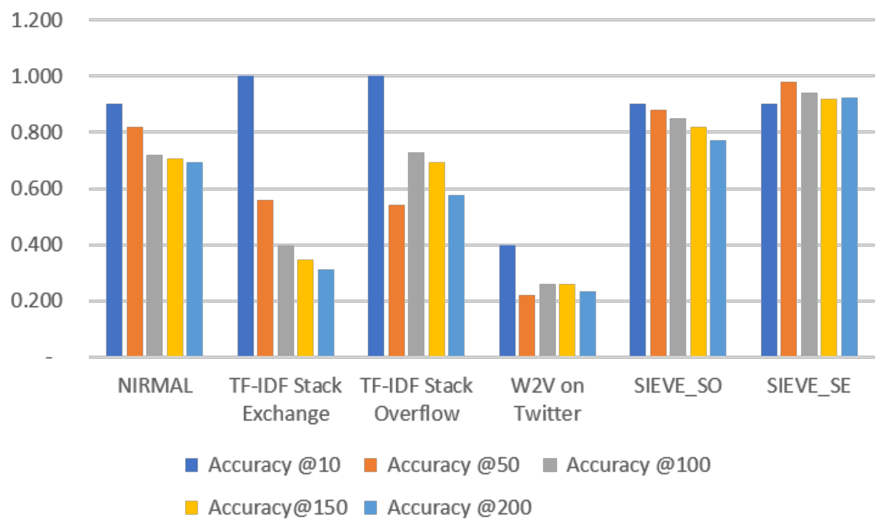


**Fig. 4** Comparison of Accuracy@K achieved by different approaches

Based on our experiments, the performance of Word2Vec trained on Twitter was lower than the other Word2Vec models. The low scores achieved by the Word2Vec model trained on Twitter data can be attributed to the content of tweets that are mostly not related to software development.

While the tf-idf-based approach performed best when ranking the top-10 most relevant tweets, the performance degrades significantly when ranking top 50, and fluctuates when ranking the top 100, 150 and 200 tweets. Figure 5 shows a box-plot diagram, describing the word count of the top 200 tweets returned by various approaches. We found that, in the top-200 tweets ranked by the tf-idf approach, the tweets mostly contain word stem ”use”, such as *”What is the use of c”*, *”Java MySQL Insert Record using Jquery”*, *”@shayman I used to work there”*. On the other hand, the top 200 tweets returned by SIEVE_SE contain more diverse vocabulary and tend to be lengthy, such as *”I still very much admire all the work put into TinyMCE Building a RTE is one of the most gruesome things you can do in a browser,”* *”@Youdaman yes angular is very minimal in the amount of code and glue you need to do specially if you use a RESTful service”*. This finding highlights the benefit of leveraging word embedding models to learn feature representation from a rich software-related platform.

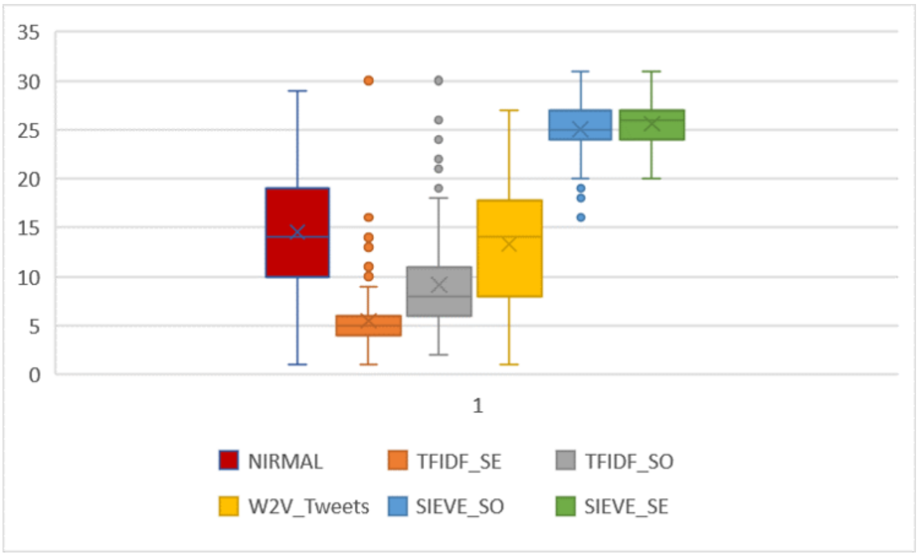


**Fig. 5** A box plot diagram representing word count of tweets returned by various approaches

RQ1: Two variants of SIEVE (`SIEVE_SE` and `SIEVE_SO`) are able to find software-related tweets with accuracy of 0.800 - 0.980.

## 5 Finding Informative Comments on YouTube Using Word Embeddings

The objective of this task is to analyze user comments for YouTube coding tutorial videos. Important users’ questions and concerns can then be automatically classified in order to help content creators to better understand the needs and concerns of their viewers, as described in work by Poché et al. [42] They categorized the comments into two general categories: informative vs. non-informative (which corresponds to other miscellaneous comments). We aim to answer the following research question:

**RQ2. How effective is our approach at the task of finding informative comments on YouTube?**

## 5.1 Approach

Figure 6 shows our proposed approach for the task of finding informative comments on YouTube, by utilizing word embeddings models trained on the source platforms (StackExchange-SE and Stack Overflow). We formulate this task as a binary classification problem, where a comment can be either informative or non-informative with regards to the video content. In order to build a classifier for this task, we need to represent the YouTube comments into a feature representation. We leverage the word embeddings learned from a source platform. The model consists of word vectors that have 300 dimensions as mentioned in Section 3. We build vectors to represent the comments, by following these steps:

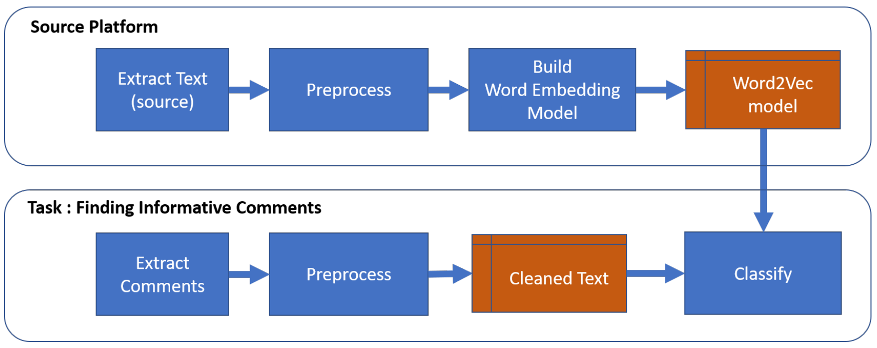


**Fig. 6** Approach for finding relevant comments on YouTube

1. For each comment, we tokenize it into words, remove words that contain only numbers, and change all words into lowercase.
2. Next, for each word in the comment, we look up its vector value taken from the word embeddings model. We ignore a word if it does not exist in the word embeddings model. We take the average of the word embeddings of all words within the text, following [25]. At the end, we have a word vector of real values with dimension of 300 for each comment.

### 5.2 Dataset and Baselines

**Dataset.** We used the dataset provided by Poché et al.[9] The dataset consists of 6,000 YouTube comments sampled from 12 different coding tutorial videos. The data was collected on Sep 6, 2016. They collected a total of 41,773 comments from all videos. They used YouTube Data API3[10] to retrieve the comments. This API extracts comments and their metadata, including the author's name, the number of likes, and the number of replies. Finally, 500 comments were sampled from the videos. Based on a manual classification process, around 30% of the comments were found to be informative, meaning that the majority of comments are basically not related to the content.

**Baselines.** Since we used the same dataset and experiment setting as Poché et al.'s work, we used their approach as the first baseline. To show the effectiveness of our models that are trained on software-development-specific platforms, we compared the models with a within-platform model trained from YouTube comments, and another cross-platform pretrained model that was learned from more general contents. We briefly describe the baselines as follows:

1. **Normalized Term-Frequency (as proposed by Poché et al. [42]).** In order to automatically identify content-relevant comments, Poché et al. investigate the performance of two classification algorithms: Naive Bayes (NB) and Support Vector Machines (SVM). They performed text preprocessing on the dataset, by stemming and removing stopwords. They also remove words that appear in one comment only since they are highly unlikely to carry any generalizable information. As feature representation, they use normalized term frequency (tf) of words in their documents. They found that their SVM classifier performs better than Naive Bayes. They also experimented with different combinations of data preprocessing such as stemming and removing stop-words, and found that the best result was achieved without stemming and stop-word removal.
2. **Word2Vec trained from YouTube Comments.** We consider this baseline as a *within-platform* baseline, since we leverage knowledge extracted from the target platform itself (i.e., YouTube comments). We built a skip-gram word embeddings model from this dataset with the same set of parameters used by our proposed approach.
3. **Pretrained Word2Vec on GoogleNews.** We used a pretrained word embedding model on GoogleNews[11] which is an alternative cross-platform pretrained model as another baseline. The model contains 300-dimensional vectors for 3 million words and phrases, which was trained on part of Google News dataset (about 100 billion words).

[9] http://seel.cse.lsu.edu/data/icpc17.zip
[10] https://developers.google.com/youtube/v3/
[11] https://code.google.com/archive/p/word2vec/

### 5.3 Experiments and Results

**Experiments Settings.** We conducted experiments to answer RQ2 and evaluated the effectiveness of our approach as compared to the baselines. We used Support Vector Machines (SVM) as the classification algorithm, since this algorithm performs better in Poché et al.'s work [42]. To enable a fair comparison, we used the same implementation of SVM (inside Weka[12]) for classification. For the kernel function, in Poché et al's work, the best results were obtained using the universal kernel. Therefore, we also used the universal kernel in our experiment. To validate the result, we used 10-fold cross validation. With this technique, the dataset was first partitioned randomly into 10 partitions of equal size. Afterwards, one of the partitions was selected as validation set while the remaining partitions are used for training. The process was repeated 10 times with a different partition being selected as validation set, ensuring that the entire dataset was used for both training and validation, and each entry in the dataset was used for validation exactly once.

To measure the effectiveness of our approach, we used the same metrics as Poché et al.'s study (i.e., Precision, Recall and F-measure). F-measure is the harmonic mean of precision and recall, and it is used as a summary measure to evaluate if an increase in precision (recall) outweighs a reduction in recall (precision). These metrics are calculated based on four possible outcomes of each comment in an evaluation set: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP corresponds to the case when a comment is correctly classified as an informative comment; FP corresponds to the case when a non-informative comment is wrongly classified as an informative comment; FN is when a comment is wrongly classified as a non-informative comment; TN is when a non-informative comment is correctly classfied as such. The formulas to compute precision, recall, and F-measure are shown below:

$$Precision = \frac{\#TP}{\#TP + \#FP}$$

$$Recall = \frac{\#TP}{\#TP + \#FN}$$

$$FMeasure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

**Results.** The results of our approach as compared to the baselines are shown in Table 3 and Figure 7. The results showed that the best performance (in terms of precision, recall, and F-measure) was achieved by using word embeddings model trained on StackExchange-SE data.

The results also showed that by using word embeddings as feature representation, the performance of the classifiers can be improved by up to 10.3%

[12] https://www.cs.waikato.ac.nz/ml/weka/

**Table 3** Performance of different approaches for classifying informative comments.

| Approach | Precision | Recall | F-Measure |
|---|---|---|---|
| `NTF (Poché et al.)` | 0.790 | 0.750 | 0.770 |
| `Word2Vec (GoogleNews)` | 0.859 | 0.861 | 0.859 |
| `Word2Vec (YouTube Comments)` | 0.829 | 0.833 | 0.830 |
| `SIEVE_SO` | 0.868 | 0.870 | 0.869 |
| `SIEVE_SE` | **0.872** | **0.874** | **0.873** |

in terms of F-measure, as compared to the normalized-tf based approach proposed by Poché et al. Among the four word embeddings models used in our experiment, models trained on Stack Exchange and Stack Overflow performed best. This finding justifies the importance of choosing a source platform that is more relevant to a target task. Even though the corpus' size is less as compared to GoogleNews data, Stack Exchange and Stack Overflow data contains more software-development-specific contents than GoogleNews, and this explains the improved performance.

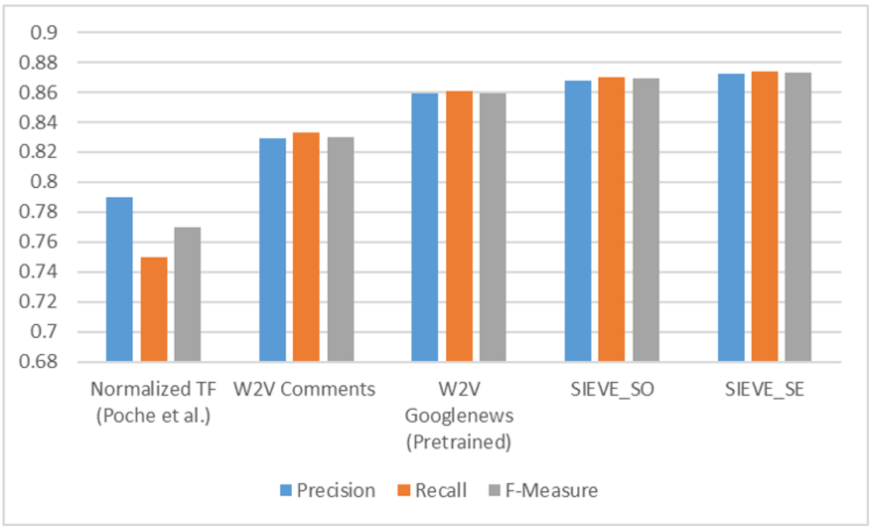


**Fig. 7** Comparison of Precision, Recall and F-measure achieved by different approaches

RQ2: Two variants of SIEVE (`SIEVE_SE` and `SIEVE_SO`) performed better than various baselines with F-measure score of 0.874 and 0.869 respectively

## 6 Threats to Validity

We present the potential threats to the validity of our findings. The threats include threats to internal, external, and construct validity.

**Threats to internal validity.** These threats are related to potential errors that may have occurred when performing the experiments and labelling. Internal threats might stem from the tools we used in our analysis. We used Gensim[13], a popular Python module for machine learning to build word embeddings that has also been used in many previous studies related to word embeddings. For machine learning and classification tools, we used Weka[14] which has been extensively used in the literature and has been shown to generate robust results for various applications. Potential errors might also occur when labelling our dataset. To label tweets as software related or not, we asked two labelers with experience in programming, and with degrees in Computer Science. We believe the labelers have enough expertise to judge if a tweet is software-related or not.

**Threats to external validity.** These threats refers to the generalizability of our results. To mitigate these threats, we have considered two source domains (Software Engineering Stack Exchange and Stack Overflow), two target domains (Twitter and YouTube), two tasks (relevant tweet identification and informative comment classification), and two settings (ranking and classification).

**Threats to construct validity.** These threats are related to the suitability of the evaluation metrics that we use for analyzing the result. We use the same evaluation metrics used to evaluate previous studies [52,42] to enable fair comparisons (i.e., Accuracy@k, Precision, Recall and F-measure). Therefore, we believe that the threat to construct validity is minimal.

## 7 Related Work

In this section, we describe work related to our study.

### 7.1 Developer Information Channels

In the past few years, there has been a substantial amount of work which has analyzed tools or channels used by software developers. Storey et al. found that software developers use many communication tools and channels in their software development work [59,60]. They found that a lot of knowledge is embedded in these tools and channels which also encourages a participatory culture in software development. Their work also highlights the challenges faced

[13] https://pypi.org/project/gensim/
[14] https://www.cs.waikato.ac.nz/ml/weka

by developers while using these channels. In the paragraphs below, we discuss work related to channels such as Software Engineering Stack Exchange and Stack Overflow (software question-and-answer sites), Twitter, and Youtube, as these are the domains we have considered in this work.

Stack Overflow has received much attention in recent years in the software engineering research community. Barua et al. analyzed in detail the topics and trends among discussions on Stack Overflow by applying Latent Dirichlet Allocation (LDA) [6]. They found that the topics which interest developers range from jobs to version control systems to C# syntax etc. Their analysis showed that web development, mobile applications, Git, and MySQL were the topics that were gaining the most popularity over time. Vasilescu et al. explore how the developer's use of Stack Overflow and GitHub relates to each other [68]. Many empirical studies have focused on understanding and modeling questions and/or answers on Stack Overflow. Asaduzzaman et al. found that some questions go unanswered on Stack Overflow as the questions may be short, unclear, too hard, etc. [2]. Rahman et al. developed a prediction model based on behavior, topics, and popularity of a question to determine unresolved questions [51]. Ponzanelli et al. have performed studies to analyze the quality of questions on Stack Overflow [46] and also proposed an approach to detect low-quality questions [47]. Treude et al. did an analysis of how programmers ask and answer questions on Stack Overflow [63]. How the crowd generates valuable documentation on Stack Overflow and ways of measuring it have been discussed in [40,39]. A lot of tools have also been proposed which can help developers in their usage of Stack Overflow. Many techniques have been proposed for solving the problem of tag prediction for questions on Stack Overflow [70,81,10]. Seahawk, an Eclipse plugin was proposed to integrate Stack Overflow knowledge within the IDE [43,4]. Identification of opinionated sentences from Stack Overflow data and subsequent aspect identification have been proposed recently by Uddin et al. [65,66].

*Stack Exchange* was first explored from a software engineering perspective by Begel et al. [7]. This work explored what kinds of service were provided by Stack Exchange, what challenges they face, and how people benefit from the service. Possnet et al. did an empirical analysis of user expertise on Stack Exchange websites and found that the expertise of users does not increase with time spent in the community; experts join the community as experts, and provide good answers from the beginning [49]. Vasilescu et al. analyzed how social Q&A sites such as Stack Exchange affect the knowledge sharing practices in open source communities [69].

*Twitter* also has been explored in recent years by the software engineering research community. Singer et al. did a survey involving 271 developers from GitHub and found that Twitter is used by developers to keep themselves up-to-date with the latest happenings in software development [56]. Bougie et al. did an exploratory study on understanding how Twitter is used in software engineering [9]. Wang et al. studied the usage of Twitter in Drupal open source development [71]. Tian et al. found that Twitter is also used by software developers for coordination of efforts, sharing of knowledge, etc. [62,61]. Sharma et al. have explored the categories of software engineering related tweets and events on Twitter [53]. Methods to filter software-relevant tweets and links have been proposed in Prasetyo et al. [50] and Sharma et al. [52,54]. Sharma et al. also proposed an approach to find software experts on Twitter [55]. Guzman et al. analyzed tweets on Twitter which talked about software applications and companies, and demonstrated that machine learning techniques have the capacity to identify valuable information for companies and developers of software applications [21,22]. They also proposed a technique to mine tweets for software requirements and evolution [23]. There has been other work also on mining Twitter feeds for software user requirements such as by William et al. [72]. Mezouar et al. found that tweets generated by users can help in early detection of bugs in software applications, and can help developers know about a bug which may be affecting a large user base [20].

*Software development videos* on YouTube in recent years have been studied as a repository from which software-related knowledge can be extracted. MacLeod et al. studied the developer's usage of videos (on YouTube) to document software knowledge [31,30]. They found that the main motivation for sharing videos by developers are building an online identity, to give back to community, to promote themselves, etc. Ponzanelli et al. proposed an approach to extract relevant fragments from software development video tutorials [45, 44]. Their approach splits the video tutorials into coherent fragments, which are then classified into relevant categories. These fragments are then available individually for developers to query, rather than being forced to browse the whole video. Poche et al. proposed an approach to identify relevant user comments on coding video tutorials on YouTube [42]. Parra et al. had proposed a text-mining-based approach to recommend tags for software development videos on YouTube [41]. Recently there has been work on extracting code and/or code related features also from programming tutorial videos extracted from YouTube by Yadid et al. [75] and Ott et al. [36].

### 7.2 Leveraging Word Embeddings

Harris et al. had hypothesized that words tend to have similar meaning in similar contexts [24]. Mikolov et al. proposed two neural-network-based language models to represent words as a low dimensional vector [32,33]. These vectors are commonly known as *word embeddings*. These models have shown considerable success in many NLP tasks [18,34,5]. Word embeddings has been used in software engineering for improving information retrieval tasks [78,74, 16]. Chen et al. have used word embeddings based methods to mine analogical libraries [12], assist collaborative editing [14], recommend tag synonyms [15] and recommend similar libraries [13]. In [76], word embeddings was combined with information retrieval to recommend similar bug reports. Methods similar to or based on word embeddings have also been used recently for better code retrieval [67], to find common software weaknesses [82], API recommendation [80] and sentiment analysis for software engineering [11]. Our work complements the existing work as we build a cross-platform approach that leverages word embeddings to aid software-development-specific knowledge extraction tasks. Additionally, we demonstrate the value of leveraging word embeddings built from a platform that contains rich software-development-relevant content to solve tasks in another platform.

## 8 Conclusion and Future Work

We proposed an approach to exploit knowledge from rich software-development-specific platforms, to automate knowledge seeking tasks in other less software-development-specific platforms. We first built word embeddings from text extracted from Stack Overflow and Sofware Engineering Stack Exchange, to represent software-development-related knowledge sources. We then leveraged the word embeddings to solve tasks in two different target platforms. In the first use case, we leveraged the word embeddings and sampled sentences from source platforms, to find software-related tweets. In the second use case, we used the word embeddings to classify informative comments on YouTube video tutorials. Based on our experiments conducted in both use cases, our approach improves performance of existing state-of-the-art work for software-development-specific knowledge extraction tasks in the target platforms.

In the future, we intend to perform additional experiments to evaluate the effectiveness of the approach for additional tasks. Finally, we also plan to expand the work to other platforms and knowledge sources, such as Wikipedia articles, software development blogs, README files on GitHub, and software documentation.

## References

1. Jerone TA Andrews, Thomas Tanay, Edward J Morton, and Lewis D Griffin. Transfer representation-learning for anomaly detection. ICML, 2016.
2. Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K Roy, and Kevin A Schneider. Answering questions about unanswered questions of stack overflow. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 97–100. IEEE Press, 2013.
3. Shams Azad, Peter C Rigby, and Latifa Guerrouj. Generating api call rules from version history and stack overflow posts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2017.
4. Alberto Bacchelli, Luca Ponzanelli, and Michele Lanza. Harnessing stack overflow for the ide. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, pages 26–30. IEEE Press, 2012.
5. Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 238–247, 2014.
6. Anton Barua, Stephen W Thomas, and Ahmed E Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 2014.
7. Andrew Begel, Jan Bosch, and Margaret-Anne Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE Software*, 2013.

8. Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.
9. Gargi Bougie, Jamie Starke, Margaret-Anne Storey, and Daniel M German. Towards understanding twitter use in software engineering: preliminary findings, ongoing challenges and future questions. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, pages 31–36, 2011.
10. Xuyang Cai, Jiangang Zhu, Beijun Shen, and Yuting Chen. Greta: Graph-based tag assignment for github repositories. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 1, pages 63–72. IEEE, 2016.
11. Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. Sentiment polarity detection for software development. *Empirical Software Engineering*, pages 1–31, 2017.
12. Chunyang Chen, Sa Gao, and Zhenchang Xing. Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 338–348. IEEE, 2016.
13. Chunyang Chen and Zhenchang Xing. Similartech: automatically recommend analogical libraries across different programming languages. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*, pages 834–839. IEEE, 2016.
14. Chunyang Chen, Zhenchang Xing, and Yang Liu. By the community & for the community: A deep learning approach to assist collaborative editing in q&a sites. In *Proceedings of the 21st ACM Conference on Computer-Supported Cooperative Work and Social Computing*, pages 32:1–32:21. ACM, 2018.
15. Chunyang Chen, Zhenchang Xing, and Ximing Wang. Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering*, pages 450–461. IEEE Press, 2017.
16. Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 744–755. ACM, 2016.
17. Ronald J Chenail. Youtube as a qualitative research asset: Reviewing user generated videos as learning resources. *The Qualitative Report*, 13(3):18–24, 2008.
18. Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
19. Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Labeling source code with information retrieval methods: an empirical study. *Empirical Software Engineering*, 19(5):1383–1420, 2014.
20. Mariam El Mezouar, Feng Zhang, and Ying Zou. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. *Empirical Software Engineering*, 23(3):1704–1742, 2018.
21. Emitza Guzman, Rana Alkadhi, and Norbert Seyff. A needle in a haystack: What do twitter users say about software? In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, pages 96–105. IEEE, 2016.
22. Emitza Guzman, Rana Alkadhi, and Norbert Seyff. An exploratory study of twitter messages about software applications. *Requirements Engineering*, 22(3):387–412, 2017.
23. Emitza Guzman, Mohamed Ibrahim, and Martin Glinz. A little bird told me: mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20. IEEE, 2017.
24. Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.
25. Tom Kenter and Maarten De Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1411–1420. ACM, 2015.
26. Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue B. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 591–600, 2010.

27. J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.
28. Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen. Word embedding revisited: a new representation learning and explicit matrix factorization perspective. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 3650–3656. AAAI Press, 2015.
29. Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):31, 2014.
30. Laura MacLeod, Andreas Bergen, and Margaret-Anne Storey. Documenting and sharing software knowledge using screencasts. volume 22, pages 1478–1507. Springer, 2017.
31. Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. Code, camera, action: how software developers document and share program knowledge using youtube. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, pages 104–114. IEEE Press, 2015.
32. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
33. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
34. Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.
35. Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in nlp applications? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 479–489, 2016.
36. Jordan Ott, Abigail Atchison, Paul Harnack, Natalie Best, Haley Anderson, Cristiano Firmani, and Erik Linstead. Learning lexical features of programming languages from imagery using convolutional neural networks. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, pages 336–339, New York, NY, USA, 2018. ACM.
37. Fabio Palomba, Annibale Panichella, Andrea De Lucia, Rocco Oliveto, and Andy Zaidman. A textual-based technique for smell detection. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*, pages 1–10. IEEE, 2016.
38. Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.
39. Chris Parnin and Christoph Treude. Measuring api documentation on the web. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, pages 25–30. ACM, 2011.
40. Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. *Georgia Institute of Technology, Tech. Rep*, 2012.
41. Esteban Parra, Javier Escobar-Avila, and Sonia Haiduc. Automatic tag recommendation for software development video tutorials. In *Proceedings of the 26th Conference on Program Comprehension*, pages 222–232. ACM, 2018.
42. Elizabeth Poché, Nishant Jha, Grant Williams, Jazmine Staten, Miles Vesper, and Anas Mahmoud. Analyzing user comments on youtube coding tutorial videos. In *Proceedings of the 25th International Conference on Program Comprehension*, pages 196–206. IEEE Press, 2017.
43. Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. Seahawk: Stack overflow in the ide. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1295–1298. IEEE Press, 2013.
44. Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, Rocco Oliveto, Mir Hasan, Barbara Russo, Sonia Haiduc, and Michele Lanza. Too long; didn't watch!: extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th International Conference on Software Engineering*, pages 261–272. ACM, 2016.

45. Luca Ponzanelli, Gabriele Bavota, Andrea Mocci, Massimiliano Di Penta, Rocco Oliveto, Barbara Russo, Sonia Haiduc, and Michele Lanza. Codetube: extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 645–648. ACM, 2016.
46. Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, and Michele Lanza. Understanding and classifying the quality of technical forum questions. In *Quality Software (QSIC), 2014 14th International Conference on*, pages 343–352. IEEE, 2014.
47. Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, Michele Lanza, and David Fullerton. Improving low quality stack overflow post detection. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 541–544. IEEE, 2014.
48. Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
49. Daryl Posnett, Eric Warburg, Premkumar Devanbu, and Vladimir Filkov. Mining stack exchange: Expertise is evident from initial contributions. In *Social Informatics (SocialInformatics), 2012 International Conference on*, pages 199–204. IEEE, 2012.
50. Philips Kokoh Prasetyo, David Lo, Palakorn Achananuparp, Yuan Tian, and Ee-Peng Lim. Automatic classification of software related microblogs. In *ICSM*, 2012.
51. Mohammad Masudur Rahman and Chanchal K Roy. An insight into the unresolved questions at stack overflow. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 426–429. IEEE Press, 2015.
52. Abhishek Sharma, Yuan Tian, and David Lo. Nirmal: Automatic identification of software relevant tweets leveraging language model. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 449–458. IEEE, 2015.
53. Abhishek Sharma, Yuan Tian, and David Lo. What's hot in software engineering twitter space? In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 541–545. IEEE, 2015.
54. Abhishek Sharma, Yuan Tian, Agus Sulistya, David Lo, and Aiko Fallas Yamashita. Harnessing twitter to support serendipitous learning of developers. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 387–391. IEEE, 2017.
55. Abhishek Sharma, Yuan Tian, Agus Sulistya, Dinusha Wijedasa, and David Lo. Recommending who to follow in the software engineering twitter space. *ACM Trans. Softw. Eng. Methodol.*, 27(4):16:1–16:33, October 2018.
56. Leif Singer, Fernando Figueira Filho, and Margaret-Anne Storey. Software engineering at the speed of light: how developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*, pages 211–221. ACM, 2014.
57. Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.
58. Andreas Stolcke. Srilm-an extensible language modeling toolkit. In *Seventh international conference on spoken language processing*, 2002.
59. Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (r) evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*, pages 100–116. ACM, 2014.
60. Margaret-Anne Storey, Alexey Zagalsky, Leif Singer, Daniel German, et al. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, (1):1–1, 2017.
61. Yuan Tian, Palakorn Achananuparp, Ibrahim Nelman Lubis, David Lo, and Ee-Peng Lim. What does software engineering community microblog about? In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 247–250. IEEE, 2012.
62. Yuan Tian and David Lo. An exploratory study on software microblogger behaviors. In *MUD*, 2014.
63. Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web?: Nier track. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 804–807. IEEE, 2011.

64. Twitter. About twitter inc. 2017. [Online; accessed 26-July-2017].
65. Gias Uddin and Foutse Khomh. Automatic summarization of api reviews. In *ASE*. IEEE Press, 2017.
66. Gias Uddin and Foutse Khomh. Opiner: an opinion search and summarization engine for apis. In *ASE*. IEEE Press, 2017.
67. Thanh Van Nguyen, Anh Tuan Nguyen, Hung Dang Phan, Trong Duc Nguyen, and Tien N Nguyen. Combining word2vec with revised vector space model for better code retrieval. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 183–185. IEEE Press, 2017.
68. Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *Social computing (SocialCom), 2013 international conference on*, pages 188–195. IEEE, 2013.
69. Bogdan Vasilescu, Alexander Serebrenik, Prem Devanbu, and Vladimir Filkov. How social q&a sites are changing knowledge sharing in open source software communities. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 342–354. ACM, 2014.
70. Shaowei Wang, David Lo, Bogdan Vasilescu, and Alexander Serebrenik. Entagrec: An enhanced tag recommendation system for software information sites. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 291–300. IEEE, 2014.
71. Xiaofeng Wang, I. Kuzmickaja, K.-J. Stol, P. Abrahamsson, and B. Fitzgerald. Microblogging in open source software development: The case of drupal and twitter. *Software, IEEE*, 2013.
72. Grant Williams and Anas Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10. IEEE, 2017.
73. Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E Hassan, and Zhenchang Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, 2017.
74. Bowen Xu, Zhenchang Xing, Xin Xia, David Lo, Qingye Wang, and Shanping Li. Domain-specific cross-language relevant question retrieval. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 413–424. ACM, 2016.
75. Shir Yadid and Eran Yahav. Extracting code from programming tutorial videos. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 98–111. ACM, 2016.
76. Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. Combining word embedding with information retrieval to recommend similar bug reports. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 127–137. IEEE, 2016.
77. Deheng Ye, Zhenchang Xing, and Nachiket Kapre. The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow. *Empirical Software Engineering*, 22(1):375–406, 2017.
78. Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 404–415. ACM, 2016.
79. YouTube. Youtube. 2017. [Online; accessed 20-AUG-2018].
80. Jingxuan Zhang, He Jiang, Zhilei Ren, and Xin Chen. Recommending apis for api related questions in stack overflow. *IEEE Access*, 6:6205–6219, 2018.
81. Pingyi Zhou, Jin Liu, Zijiang Yang, and Guangyou Zhou. Scalable tag recommendation for software information sites. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 272–282. IEEE, 2017.
82. Hongtao Liu Zhenchang Xing Zhuobing Han, Xiaohong Li and Zhiyong Feng. Deepweak: Reasoning common software weaknesses via knowledge graph embedding. In *Software Analysis, Evolution, and Reengineering (SANER), 2018 IEEE 25rd International Conference on*, 2018.