

Evaluating the Use of Semantics for Identifying Task-relevant Textual Information

Arthur Marques

Department of Computer Science
University of British Columbia, Canada
msarthur@cs.ubc.ca

Gail C. Murphy

Department of Computer Science
University of British Columbia, Canada
murphy@cs.ubc.ca

Abstract—The information a developer seeks to aid the completion of a task typically exists across a range of artifacts. For example, for a task that requires upgrading to a new version of an API component, a developer may seek information in the API’s official documentation, check community discussions about the newer version, and so on. To aid developers in locating the portion of the text that might be useful in these artifacts, prior work has used syntactic properties of the text, and an artifact’s meta-data, to automatically identify relevant text for particular kinds of artifacts. Although effective, these techniques rely on assumptions about an artifact’s structure or content that prevent applying them across the different types of artifacts that a developer may come across in their daily work. In this paper, we investigate whether techniques building on approaches to interpret the meaning, or semantics, of the text help to overcome these limitations. Particularly, we introduce six semantic-based techniques and evaluate that they can identify up to 58% of the text that developers deem relevant to Android development tasks. When compared to a state-of-the-art approach, we find that our techniques achieve comparable recall values, identifying 63% of the small fraction of the task-relevant text of Stack Overflow artifacts, but without the need for artifact-specific information.

Index Terms—Empirical Software Engineering, Software Tasks, Natural Language Artifacts

I. INTRODUCTION

Performing a task on a software system, like fixing a bug or adding a feature, typically requires a developer to consult a number of different kinds of artifacts, such as API documentation, bug reports, community forums and web tutorials. When consulting these artifacts, a developer must identify, from the large amount of text in these documents, just the fraction of text relevant to the task-at-hand.

Unfortunately, developers tend to struggle with identifying the relevant text and when they are unable to locate all, or most, of it, they may produce incomplete or incorrect solutions [1]. To aid developers in this situation, researchers have proposed various techniques and tools to identify likely relevant text. For example, *Krec* [2] identifies text that a developer cannot afford to ignore when reading an API document. Although effective, this and other techniques (e.g., [3] or [4]) target specific types of artifacts, limiting their use across the many different kinds of artifacts developers encounter daily in their work.

If one technique could identify relevant text across all kinds of artifacts a developer encounters, the technique could apply

in all situations and may be more adoptable in industry as a result. Guided by recent success using techniques that interpret the meaning, or *semantics*, of text for a variety of development activities, such as for finding who should fix a bug [5], searching for comprehensive code examples [3], or assessing the quality of information available in bug reports [6], we consider whether semantic-based techniques might also aid in identifying task-relevant text in software artifacts.

To investigate this hypothesis, we introduce six possible techniques that incorporate the semantics of words and sentences. The first two use word embeddings [7] to identify likely relevant text via semantic similarity, *word2vec*, and via a deep neural network, *BERT*. Sentence-level techniques are filters that we apply (or not) on top of the word embedding techniques generating four additional techniques. These techniques use frame semantics [8] to obtain entities (frames) that summarize a sentence’s meaning. One of our sentence-level techniques, *frame-elements*, finds task-relevant text based on a set of key frames. The other, *frame-associations*, matches frames found in a task and an artifact to determine the text’s relevance.

Evaluating these techniques requires a dataset that comprises software tasks with associated artifacts and an identification of the text within those artifacts that is pertinent for completing the task. As no such dataset is available, we created one that comprises 50 software tasks about Android development for which human annotators identified pertinent text per task across a variety of kinds of software artifacts, including API documentation, Stack Overflow answers, GitHub issue discussions and other web pages for a total of 133 distinct artifacts.

Using this dataset, we then sought to answer two questions: 1) how do these semantic-based techniques compare to a state-of-the-art technique that is artifact-specific and 2) which of these six techniques provides the best results? To answer the first question, we restrict our comparison to a subset of the dataset restricted to Stack Overflow artifacts, which is the target of the *AnswerBot* technique to which we compare. To answer the second question, we consider a range of software artifacts in the dataset. From this assessment, we find that *BERT* with no filters and *BERT* with *frame-associations* have recall comparable to *AnswerBot*, identifying 63% of the small fraction of the text in the

artifacts that human annotators indicated as relevant to a task. We also find that, for the different types of artifacts in this dataset, semantic-based techniques can identify up to 58% of the text identified as relevant to a task in a software artifact.

In summary, this paper makes three contributions:

- it provides a unique dataset [9] with 50 Android tasks and associated textually-based artifacts that includes annotations of which text in the artifacts is deemed pertinent to these tasks;
- it reports on the precision and recall of the six semantic techniques we introduce using the newly created dataset; and
- it shows that semantic approaches have recall comparable to state-of-the-art approaches tailored specifically to certain kinds of artifacts; recall is the preferred measure to ensure as much of the relevant text as possible is returned.

We begin by describing related work (Section II) and detailing the semantic-based approaches that we explore (Section III). We then describe our novel dataset (Section IV) and how we use it to evaluate the techniques (Section V). We conclude with threats to the validity of our study (Section VI), implications of our findings (Section VII), and a summary of our work (Section VIII).

II. RELATED WORK

Software engineering researchers have made use of the natural language text appearing in different kinds of artifacts associated with software development in many different ways. Many of the approaches to use this text rely on lexical and syntactic properties of the text. We review various studies that have been taken using these approaches, honing in on studies that help identify text relevant to a task-at-hand. More recently, researchers have also started to use semantic approaches; we also provide an overview of these approaches to provide background of how semantics can be useful to support software development activities.

A. Using Lexical and Syntactic Properties of Text

Early uses of textual information in artifacts relied on the frequency of co-occurrence of words or phrases. For example, Maarek and Smadja's used lexical relations to index software libraries [10]. Since this early use, software engineering researchers have continued to leverage advances in natural language processing (NLP) approaches, such as when De Lucia et al. applied Vector Space Model (VSM) to support identification of human labelled source code components [11].

Interpretation, manipulation and extraction of text in software artifacts have been employed for many different purposes to improve software development. We review some of the approaches to give a sense of the breadth of the approaches in terms of different kinds of artifacts that have been common targets.

A number of approaches target bug reports and GitHub issues, largely focusing on summarizing information within these artifacts. These summaries aim to represent key information that may help a developer complete their task. For

example, Rastkar et al. proposed a set of logistic regression classifiers to summarize bug reports based on textual properties inferred from the conversational nature of these artifacts [12]. As another example, Lotufo et al. propose a tool that automates the identification of sentences that a developer would first read when inspecting bug reports [13]. Others have focused on assessing the text rather than summarizing it. For example, Chaparro et al. have used a set of linguistic patterns to identify whether a new bug report contained steps to reproduce the bug or the system's expected behaviour, which they then used as a means to suggest improvements in the quality of the bug report description [14].

Other approaches target web artifacts in the form of tutorials or question and answer sites, such as Stack Overflow¹, often combining information from web sources with other artifacts. For example, Ponzanelli et al. [15] take the text available in the code and documentation that a developer inspects in an IDE and use lexical similarity approaches to identify Stack Overflow answers that might help a developer understand that code. As other examples, Nadi and Treude defined a set of linguistic patterns to identify relevant information in a Stack Overflow answer [16] while Jiang et al. propose FITSEA [17] to augment API documentation with information extracted from tutorials, where the tool finds segments of the tutorial that mention an API element based on textual matching and term frequencies.

While many of the approaches described above rely on largely lexical aspects or patterns in text, researchers have also made use of structured textual information in the artifacts [18]–[20]. For example, recognizing the value of structured data available on Stack Overflow, Ponzanelli et al. proposed a summarization technique that mixes natural language text and structured data to produce more accurate summaries for Stack Overflow answers [18]. Using similar data, Treude and Robillard proposed SISE, a tool that augments API documentation with relevant information extracted from Stack Overflow answers [20].

A smaller number of approaches have focused on the problem that is the focus of this paper, namely identifying text in artifacts related to a specific task-at-hand. Chhetri and Robillard have created a tool, called Krec, that identifies relevant information, such as caveats or threats, in API documentation applying a series of NLP approaches, such as part of speech tagging and shallow parsing, to determine which of the sentences in the API documentation are relevant [2].

Many researchers have since proposed tools (e.g., [3] or [21]) that use information such as whether a post has an accepted answer or the number of votes a post has to generate answers to a developer's task. Among these, our work is closest to AnswerBot's relevant and salient text selection algorithm [4]. AnswerBot uses this algorithm to identify relevant text based on the content of the text, of the task and the structured data available on each of the paragraphs of answers in a Stack Overflow post. Although this is one

¹ www.stackoverflow.com

of the three steps that AnswerBot uses to produce an answer summary to a developer’s task, such algorithm also applies to identifying task-relevant textual information, as we show as part of our evaluation (Section V).

All of these existing approaches are specific to one kind of artifact and use properties of the kind of artifact. In contrast, the techniques we investigate in this paper seek to identify task-relevant text across different artifact types.

B. Interpreting the Semantics of Text

We use the term semantics to refer to having access to a meaning for words, phrases or sentences appearing in text. Semantic-based approaches build a meaning by establishing relationships between terms, for instance words, based on the context in which these terms appear. For example the terms ‘car’ and ‘automobile’ are likely to co-occur in different phrases with terms like ‘motor’ and ‘wheel’ and thus, semantic approaches indicate that these terms are likely similar in meaning [22].

As an example of the usage of semantics in software engineering researcher, Marcus and Maletic applied Latent Semantic Indexing (LSI) to help cluster software components to aid program comprehension of a software systems [23]. Ye et al.’s study on the usage of word embeddings [24] was among the first to use the language models in the software engineering domain [7].

Researchers have also developed approaches to interpret the meaning of sentences in software engineering documents. Maalej and Robillard have developed a knowledge taxonomy [25] using grounded theory to analyze documentation in open-source systems and then, they validated their taxonomy on documentation units sampled from the Java SDK 6 and .NET 4.0. Arya et al.’s have determined information types in open-source issue discussions [26], Di Sorbo et al. have created an approach to classify emails based on a developer’s intentions [27] and Marques et al. have investigated the use of frame semantics [8]. The techniques we propose in this paper build on these earlier findings.

III. SEMANTIC TECHNIQUES

Each semantic technique takes as input a task description and an artifact. The technique then identifies sentences in the artifact pertinent to the task. The output from the technique is a ranked set of sentences (from the artifact) from most likely to be helpful for solving the task to the least helpful.

We introduce and assess six semantic techniques. Three of the techniques use word2vec [7] as a base for determining a ranked list of similar sentences between an artifact and a task description and three use BERT [28]. One of the techniques using word2vec simply returns the results of word2vec; a second technique based on word2vec filters the ranked artifact sentences—with a filter we call *frame-elements*—by applying frame semantics to each sentence and retaining only those that include certain frames; the third technique filters—with a filter we call *frame-associations*—by keeping sentences with certain co-occurrences of frame semantics between the task

description sentences and the artifact sentence. The same set of filters—no filter, frame-meanings and frame-associations—are used to produce three techniques using BERT, as shown in Table I. We describe word2vec, BERT and the filters in turn to explain the six techniques.

A. Identifying Task-relevant Text with word2vec

To form the base of the first three techniques, we use word semantics via word2vec in conjunction with information retrieval approaches. Information retrieval techniques are usually applied to look for documents relevant to a query prompted by a user [22], where the relevance of a document to a query is based on some function that indicates how similar the document’s content is to that query [29]. Given the right *similarity function*, information retrieval techniques can be also applied to identify sentences within a document, artifact in our problem formulation, that are relevant to a task.

We use the *Skip-gram* model [7], also referred to as *word2vec*, to define such a similarity function. The *Skip-gram* model exploits Harris’ distributional hypothesis [30]—which states that words that appear in a similar context tend to have similar meanings—and builds vector representations, namely *word embeddings*, for each of the words in a text corpus. With a significantly large text corpus, the model associates similar vector embeddings to words that are similar in meaning [24].

Since *word2vec* represents words through a vector space, the similarity of two words i and j can be obtained computing the cosine similarity [29] between the corresponding vectors of each word, i.e., w_i and w_j :

$$\text{sim}(w_i, w_j) = \frac{w_i^T w_j}{\|w_i\| \|w_j\|} \quad (1)$$

In turn, the similarity of two sentences can be obtained by first computing a vector representation for the entire sentence and then, by measuring the cosine angle between the obtained sentence vectors. To obtain sentence vectors, we follow Conneau et al.’s guidelines [31] and we average the sum of the embeddings for each word in a sentence.

Following these procedures, we use a Skip-gram model with word embeddings trained for the software engineering domain [32] to compute the semantic similarity, or relevance, of the sentences $\{a_1, a_2, \dots, a_n\}$ within a pertinent artifact A and the sentences $\{t_1, t_2, \dots, t_m\}$ in a task T . As both entities have multiple sentences, the semantic similarity of a sentence $a_i \in A$ is the maximum value obtained for this sentence and each of the sentences in a task.

After we compute the semantic similarity of all the sentences in an input artifact with regards to an input task, we sort the obtained values from highest to lowest, outputting the top- n sentences with highest similarity as the ones likely relevant to an input software task.

B. Identifying Task-relevant Text with BERT

In the Skip-gram model, context refers to the sentences used to train the model. This approach to context does not allow the model to disambiguate words based on their surrounding text.

TABLE I: Summary of semantic-based approaches for automatically identifying task-relevant text

Identifier	Sentence filters	Description
word2vec	<i>no filter</i>	Uses the Skip-gram model to identify relevant sentences as the sentences most semantically similar to a task
	<i>w/ frame-elements</i>	Modifies the output of the word2vec base approach according to whether sentences contain meaningful frame elements
	<i>w/ frame-associations</i>	Modifies the output of the word2vec base approach according to whether sentences contain meaningful task-artifact frame pairs
BERT	<i>no filter</i>	Fine-tunes BERT to predict the sentences that are likely relevant to an input task
	<i>w/ frame-elements</i>	Modifies the output of the BERT base approach according to whether sentences contain meaningful frame elements
	<i>w/ frame-associations</i>	Modifies the output of the BERT base approach according to whether the sentences contain meaningful task-artifact frame pairs

In other words, a Skip-gram model will have a single vector representation for a word such as ‘company’ even when it can have different meanings, i.e., a business organization or being with someone. In contrast, the **Bidirectional Encoder Representations from Transformers** (BERT) [28] provides different representations for the same word based on the sentence in which a word appears. This additional layer allows the model to perform more complex operations, leading to state-of-the-art results in several tasks [28].

Typically, BERT is trained on a massive amount of data. During training, a percentage of the tokens in a sentence—usually 15%—are replaced with a special token and the model is optimized to predict these replaced, or *masked*, tokens. The model relies on a mechanism, called *attention* [33] that correlates and weights all non-replaced tokens in the input so that the model maximizes predictions. To fully train a BERT model, one first creates a base model using a large amount of text and the general token prediction task. Then, this base model is fine-tuned to specific tasks, such as text classification, using a dataset specific to the fine-tuning steps. The procedures of *training* and *fine-tuning* allow using the model even when training data is scarce and the model transfers what it learned during training steps to the fine-tuning steps [28].

We posit that BERT’s attention mechanism and fine-tuning procedures can be used to train the model to classify sentences in a natural language artifact as likely relevant to software task.

The second group of three techniques use a BERT model to classify if a sentence is relevant or not to a task. To this end, we take the sentences $\{a_1, a_2, \dots, a_n\}$ within a pertinent artifact A and the sentences $\{t_1, t_2, \dots, t_m\}$ in a task T and we feed them to the model as task-artifact input pairs alongside binary labels representing whether that sentence is relevant. Using training data, the model’s attention mechanism will learn associations between each pair such that it can predict if a sentence is relevant for the task provided. We use up to 10 epochs to train BERT, setting both *batch size* and a *sequence length* to 64. Cross-Entropy is our loss function, and the model is trained to minimize it using the Adam optimizer at a learning rate of $1e-5$ with early stopping.

Following these procedures, we take a fully trained model and we compute the relevance of a sentence $a_i \in A$ as the most commonly predicted label for this sentence when we pair it to each of the sentences in a task T . Similar to the word2vec approach, we predict the relevance of all the

sentences in an input artifact with regards to an input task and we use prediction probabilities to select the top- n sentences with highest probability as the relevant ones.

C. Filtering Task-relevant Text with Frame Semantics

Although word2vec and BERT have provided significant improvements to a diverse set of natural language tasks, they still capture semantics at the word level. If we seek to infer a sentence’s meaning, an alternative would be to consider *frame semantics* [8].

Frame semantics is based on how readers comprehend the roles that words take in a sentence with respect to events of interest. These events are defined as *frames* while the parts of a sentence that assist a reader in understanding each event are defined as *frame elements*. For instance, Figure 1 shows the results of a frame analysis for a sentence giving instructions on how to push user notifications in an Android app—*You need to have implement a BroadcastReceiver and declare it in the AndroidManifest.xml*². In the figure, frames are highlighted (in gray) while frame elements (*fe*), and the parts of the sentence associated to them, are enclosed by square brackets. Through frame analysis, we can use the frame ‘*Required event*’ as a proxy to the sentence’s meaning.

You [need] **Required event** [to have implement a BroadcastReceiver and declare it in the AndroidManifest.xml] **fe:Required situation**

Fig. 1: Frame analysis example

We define two filters using frame semantics that can be applied with either the word2vec or BERT base techniques as a post-processing step to identify sentences potentially relevant to a task based on the frame elements that the *SEFrame* tool identifies [34]. For that, we implement a set of filters that can be applied to the previous techniques as a post-processing step [29]. A filter takes the output of a technique O and produces output O' where a sentence must match the filter’s relevance criteria. For both filters, we use *SEFrame* [35] to assign frames relevant to software engineering text.

Frame-elements: This filter considers sentences identified with frames giving instructions, describing required events, explaining system procedures, and others as likely to contain information of interest. The rationale for the set of frames

²<https://stackoverflow.com/questions/24313539>

identified is based on findings from related work that have shown that the meaning of the text is an indicator of its relevance to a software task [34]. This filter takes a sentence $a_i \in A$ and checks whether any of the frame elements of this sentence match one meaningful frame $f \in F$ from a pre-established set³.

$$F = \{\text{being obligated, causation, } \dots \text{required event, using}\} \quad (2)$$

$$\text{frame-elements}(a_i) = \begin{cases} 1, & \text{if } \text{frames}(a_i) \cap F \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Frame-associations: This filter considers that the relevance of a sentence depends on the intentionality of a task. For example, for a task that requires diagnosing an error, sentences in a bug report that describe success or failure of an action are likely relevant. This rationale is represented by a set of associated frame pairs, where each element represents a frame originating from a task and another frame originating from a sentence. This filter checks if a task-artifact frame pair obtained from an input artifact and input task appear in a pre-established set of pairs³.

$$P = \{(\text{execution, being obligated}), \dots (\text{questioning, using})\} \quad (4)$$

$$\text{frame-associations}(a_i, T) = \begin{cases} 1, & \text{if } \text{pair}(a_i, T) \cap P \neq \emptyset \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

We identify frame-pairs $p \in P$ using association rule mining [36], which identifies frame elements that co-occur in a task's title and in the text marked as relevant in the artifacts associated to each task. We mine pairs using the apriori algorithm [37] with a minimum support level—the minimum number of times the frame element pairs must appear across the data—of 0.10. Section V further details the data used to identify rules.

IV. DATASET

Our evaluation requires a dataset that contains software tasks, associated artifacts and text in the artifacts marked by human annotators as relevant to the tasks considered. We create this dataset because, to the best of our knowledge, existing datasets (e.g., [16], [34] or [38]) comprise tasks with only a single, or a small number, of types of artifacts.

Figure 2 summarizes the process we use to create the dataset, which we call DS-android. We randomly sample a set of 50 Android development tasks from two common task sources, GitHub and Stack Overflow. For each of these tasks, we use the Google search engine to find potential artifacts that likely contain relevant information for that task. We then gather task-relevant sentences from these artifacts from experienced developers.

³See full list in our replication package [9]

We focus our dataset on *Android development* because the Android *SDK* constantly evolves due to functionality, security and performance-related improvements [39], [40]. These improvements impact its development community, requiring them to often seek information regarding changes in the SDK [41]–[43]. For example, over 35,000 developers have used Q&A forums to discuss tasks covering 87% of the classes in the Android API [44]. Section VI discusses threats related to the creation of the dataset.

A. Software Tasks

We start dataset creation by identifying software tasks for which a developer will likely benefit from the use of additional information to complete.

Two common places where Android tasks can be found are: 1) the description of an issue (e.g., a bug or feature request) reported in an issue tracking system; or in 2) a post in a community forum, development mailing lists, and others. Several studies have used such sources for software tasks [4], [16], [45] and, following the lead of these studies, we select GitHub issues and Stack Overflow posts as the two sources for the tasks in our dataset.

To select tasks from GitHub (e.g., a bug or feature request), we are guided by studies that use stars as a proxy for a project's popularity [46], [47]. We selected 14 projects by filtering the list of top-starred projects in GitHub to those with the *Java* and *Android* tags. These projects range from mail clients⁴ to development frameworks⁵ and we randomly selected 25 distinct issues originating from them as the GitHub tasks of our dataset. While selecting issues, we took care to check that they had at least one follow-up comment and that the issue title did not contain certain words (e.g., *test* or *ignore*), that indicate issues typically created automatically by scripts or bots—a common pitfall that researchers must be aware of when mining GitHub [48].

We randomly select 25 Stack Overflow posts from a curated list about Android development [38]. This list was built by Baltes et al. using the Stack Overflow dump published on June 5, 2018 [49], [50] and it contains 209,536 unique posts with the *Java* and *Android* tags. We consider Stack Overflow posts as software tasks because to answer a post, a developer often needs to provide references supporting their answer [51]. Finding these references in a timely manner and writing the key information that helps a user understand the provided solution encompasses many of the activities found in a developer's daily work, e.g., work-related browsing, coding, debugging, reading, and writing [52].

B. Artifact Selection

Next, we require a set of artifacts with textual information that helps a developer complete each of the tasks selected. To select artifacts pertinent to a task in our dataset, we seek to simulate everyday practices on how developers search the

⁴<https://github.com/k9mail/k-9>

⁵<https://github.com/libgdx/libgdx>

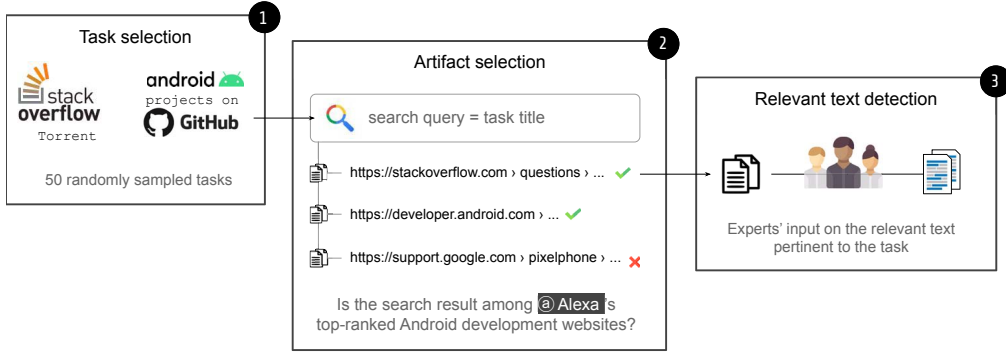


Fig. 2: Summary of procedures for dataset creation

web [53], [54]. Hence, we use a search engine to retrieve artifacts that are pertinent to each task.

Since the total number of artifacts that may assist a developer is unknown and unbounded, we focus our selection to well-known and studied artifact types [55], [56], namely Android and Java SE API documentation, GitHub issues, Stack Overflow answers, and web tutorials or blog posts on Java and Android development.

We use a task’s title (i.e., Stack Overflow question or GitHub issue title) as the seed to search for pertinent artifacts within our selected artifact types. We use `googlesearch` API [57] to get the results of a query adding `site:domain` to search for artifacts only in (or outside) a given web domain⁶—procedures similar to [4].

We request up to 5 resources per query, but we only include up to one API document, one GitHub issue discussion, one Stack Overflow answer, and two miscellaneous web pages in the final artifact set for a task. When selecting results, we exclude any entry that does not appear in the Amazon Alexa [58] web traffic statistics for Java and Android development in the period from April 2020 to March 2021. We apply this filter to include software development artifacts and remove results such as a tutorial about “*stock swap*” operations which was initially fetched for a task discussing “*left and right-hand swap*”.

Limiting the number of artifacts up to a maximum of 5 artifacts per task relates to the time-consuming [59] and cognitively demanding [60] nature the final step in the dataset creation, i.e., asking annotators to carefully read, understand and identify the text within the fetched artifacts that relevant to a given task.

C. Relevant Text Detection

The final step in the dataset creation requires determining which of the text in the gathered artifacts could provide information that assists a developer in solving her task. To produce this *golden data*, we ask experienced developers to mark the text that they deem useful and that provides information for tasks assigned to them [2], [16], [34]. We

⁶See query templates in our replication package

TABLE II: Android dataset summary

	# of sentences			# of sentences marked		
	total	mean	stdv	total	mean	stdv
API documentation	4,915	109.22	96.77	327	9.62	7.88
GitHub issues	1,890	43.95	33.69	146	4.87	3.37
Stack Overflow answers	1,420	28.40	28.48	330	7.33	5.08
Miscellaneous pages	4,176	70.78	56.64	590	12.55	9.88
Overall	12,401	62.95	66.25	1393	8.93	7.78

recruited 3 graduate students with professional programming experience to produce *golden data* for our dataset. Annotators had to have experience with Java development and they also had to be familiar with the types of artifacts they would encounter throughout the annotation process. On average, annotators self-reported 3.0 years of professional programming experience (stdv 1.63, ranging from 1 to 5 years).

Annotation was facilitated by a tool we created for this purpose. First annotators performed two tasks, other than the ones in our dataset, to familiarize themselves with the tool and annotation procedures. We discussed results from these two tasks with all annotators to ensure consistency over responses to any of the questions annotators had. No changes arose from the two introductory tasks. After this step, we presented the tasks to the annotators, dividing them into batches of 10 tasks each. For each task, annotators had the task description and links to artifacts pertinent to the task at their disposal. We asked annotators to write a short (250 words max) set of instructions a developer could follow to complete the task successfully. The purpose of the plan was to ensure that annotators built enough context about the task. While perusing artifacts, annotators also used our tool to manually highlight sentences that they deemed useful and that provided information that assisted task completion—instructions similar to the ones used for the creation of the data in Nadi and Treude’s study [16] or Marques et al.’s study [34].

D. Dataset Summary

Overall, the resultant `DS-android` dataset has 50 tasks and 133 artifacts, divided as 33 API documents, 45 Stack Overflow answers, 20 GitHub issues, and 35 other web pages. Tasks in the dataset have an average of 3 associated artifacts

each and we discriminate the text marked by each annotator per artifact. Overlap between the text marked by the annotators represents 20% of the entire data and *Krippendorff's alpha* [61] indicates good reliability [62] over the sentences marked as relevant (or not) by the annotators ($\alpha = 0.69$).

Table II provides summary statistics for the dataset's content over the artifacts inspected as well as on an artifact type basis according to the text marked by any annotator. Based on individual annotations, the text deemed useful to a software task in the artifacts inspected comprises an average of 8.93 sentences per artifact per annotator. This usually represents 15% of the text in an artifact. We also observe that the highest number of sentences marked originate from miscellaneous types of web pages while the lowest come from GitHub issue discussions. The high number of sentences marked in the former might relate to the more didactic nature of web tutorials [63], [64]. For the latter, the content on GitHub may be too project-specific which might prevent a developer who is performing a task in a different context from benefiting from most of the content found in this type of artifact. We discuss threats that might arise from these differences in Section VI.

V. EVALUATION

Our aim is to create a technique that can apply to multiple kinds of software artifacts so that software developers can rely upon one technique instead of many, one for each artifact type. To be useful, the one technique that applies to multiple artifacts must perform similarly to techniques specialized to particular kinds of artifacts.

We first evaluate whether any of the six techniques we introduce performs similarly to a technique specialized to one kind of artifact. We chose to compare the performance of our techniques to that of AnswerBot on Stack Overflow answers because, as Section II details, the tool's relevant and salient text selection is the technique closest to our work. This evaluation helps answer: *How do our six semantic-based techniques compare to a state-of-the-art technique that is specific to a particular artifact type?*

We then consider whether the semantic-based techniques can perform equivalently well across multiple artifact types. This evaluation helps answer: *Which semantic-based technique provides the best results?*

We use *precision* and *recall* metrics [65] to measure what portion of the text identified by human annotators the techniques automatically identify. In this context, we believe recall to be the most important metric since failure to identify text that is relevant to a task means that a developer will have an incomplete or partial view of the information needed, what can lead to faults [1].

A. Experimental Setup

To evaluate the six techniques and determine how much of the task-relevant text in the DS-android dataset they are able to identify, we use the following experimental setup.

1) *Techniques' Output*: We configure each technique to identify a target number of 10 relevant sentences for each task-artifact pair. This decision is based on the fact that no more than 15% of the content of any artifact in our dataset is deemed relevant to a task, which, on average, accounts for 8.93 sentences. Researchers have also used the same target number of 10 sentences when evaluating techniques (e.g., [4] or [13]) able to identify relevant text for certain kinds of artifacts, what will also facilitate comparing our results to related work.

2) *Training & Testing Data*: In addition to configuring the techniques' output, two of our techniques require training data for fine-tuning purposes (BERT) and to derive task-artifact frame pairs (frame-associations). We ensure that no data used to evaluate these techniques is also used to train them by splitting the dataset into two portions, one for training and another for testing, each with an equal number of tasks. Due to our comparison with AnswerBot, we also ensure that all tasks used for testing purposes have associated Stack Overflow artifacts. That is, we create our test set randomly selecting 25 tasks in the dataset that contain a Stack Overflow artifact among its associated artifacts.

3) *Metrics*: We compute values for *precision* and *recall* metrics based on the sentences deemed relevant to a task by *at least two* human annotators. To minimize risks from a scenario where one of the techniques and their underlying approaches has a peak or bottom performance due to training procedures or due to factors beyond our control, we compute results for each technique over 10 distinct executions over the test data, reporting the average of each metric over all runs.

Precision measures the fraction of the sentences identified that are relevant over the total number of target sentences identified, as shown in Equation 6.

$$Precision = \frac{TP}{TP + FP} \quad (6)$$

Recall represents how many of all the annotated sentences are identified by a technique (Equation 7).

$$Recall = \frac{TP}{TP + FN} \quad (7)$$

Precision means identifying only text that is relevant, whereas recall means identifying all relevant text. Ideally, we would aim for a technique with high precision and high recall. Unfortunately, this is often not possible and we must reach a compromise. As described earlier, our goal is to support developers to locate text that might be relevant to their task, and not locating all the relevant text may lead to incomplete or incorrect solutions, thus the reason why we favour recall.

B. Comparison to AnswerBot

For a fair comparison between our techniques and AnswerBot (AnsBot), we must ensure that we obtain measurements for all the approaches under the same circumstances. This could mean applying our techniques to the tasks and artifacts used in AnswerBot's original evaluation or applying both our semantic-based techniques and AnswerBot to our test data.

TABLE III: Comparison to AnswerBot

technique	no filter		w/ frame-elements		w/ frame-associations	
	precision	recall	precision	recall	precision	recall
AnsBot	0.59	0.63	-	-	-	-
word2vec	0.43 [*]	0.48 [*]	0.49 [*]	0.45 [*]	0.45 [*]	0.50 [*]
BERT	0.58	0.63	0.53 [*]	0.58 [*]	0.58	0.62

^{*} AnsBot performs better with a large effect size ($\alpha = 0.1$, *Cohen's D* ≥ 0.8);

It would be interesting to report results for both scenarios, but golden data in AnswerBot represents how human evaluators judged the target sentences outputted by the tool. This comprises only a portion of the entire text available in a Stack Overflow answer, which can have more text that these evaluators could have judged as relevant. Since we do not have access to AnswerBot's original judges to construct this data, we evaluate our techniques and AnsBot only on the tasks and Stack Overflow artifacts in the portion of the DS-android dataset that we use for testing.

Since AnsBot uses both textual properties and meta-data, we also evaluate how much of the tool's accuracy relies on properties such as the number of votes an answer got on the platform or whether the answer has been accepted. For that, we disable any of the features used by AnsBot that rely on meta-data and we compute evaluation metrics once more. We refer to this configuration in our results as AnsBot_{text}.

Results: Table III shows values of precision and recall for AnsBot and the semantic techniques that we explore when identifying text annotated as relevant for each artifact of each task. In the table, rows provide details about the base technique while columns show which post-processing filters were applied as well as precision and recall values. We also use a paired Wilcoxon-Mann-Whitney test [66] to check if the results between each technique and AnsBot are statistically different, marking which technique performs better.

Overall, we observe that AnsBot achieves 0.59 precision and 0.63 recall—values explainable by the fact that the tool is tailored specifically to Stack Overflow. When we compare AnsBot to our techniques, we observe that the base BERT approach and the one using frame-associations achieve result comparable to AnsBot. Interestingly, the frame-elements filter does not provide significant improvements to the base approach. The lack of differences may be explained by the fact that, to determine relevance, the attention mechanism used by BERT already correlates the text in a task and in an artifact (Section III-B), and that the vector representations in the model are able to infer contextual information, which may serve as an implicit way to identify a sentence's meaning. For word2vec, we observe that the technique's results are significantly lower than AnsBot, which suggests that it fails to identify relevant text identified by AnsBot.

Without meta-data, results (Table IV) indicate that all BERT techniques achieve significantly higher recall than AnsBot_{text}. This suggests that, to determine relevancy, a neural network might implicitly use some of the properties of

TABLE IV: Comparison to AnswerBot_{text}

technique	no filter		w/ frame-elements		w/ frame-associations	
	precision	recall	precision	recall	precision	recall
AnsBot _{text}	0.53	0.53	-	-	-	-
word2vec	0.43 [*]	0.48 [*]	0.49 [*]	0.45 [*]	0.45 [*]	0.50 [*]
BERT	0.58 [†]	0.63 [†]	0.53	0.58 [†]	0.58 [†]	0.62 [†]

[†] BERT performs better with a large effect size ($\alpha = 0.1$, *Cohen's D* ≥ 0.8);

^{*} AnsBot_{text} performs better with a large effect size ($\alpha = 0.1$, *Cohen's D* ≥ 0.8);

text that appears in highly voted or accepted answers.

Results from our comparison to a state-of-the-art approach, namely AnswerBot, indicate that text-based semantic techniques achieve comparable accuracy when identifying relevant information in Stack Overflow artifacts.

C. Evaluation Across all Artifact Types

Provided that we can identify task-relevant textual information with accuracy comparable to a state-of-the-art approach, we measure how much of the text that is relevant to a task (within an artifact) can our semantic-based techniques identify. To this end, we compute precision and recall metrics over all the artifacts in the test data. To assist comparisons, we also report results for the text identified by a standard VSM lexical similarity approach. Several studies [34], [67], [68] have shown that developers often use keyword-matching as a first search strategy to locate text that might contain information relevant to their tasks. Thus, this baseline might assist us in interpreting how much of the task-relevant text a developer would identify by themselves when inspecting an artifact at first glance.

We also consider precision and recall metrics per type of artifact, i.e., API documentation, Stack Overflow answers, GitHub issues, and miscellaneous web pages. We measure how much each metric varies across these artifacts reporting their standard deviation. The less variation there is, the more a technique consistently identifies task-relevant textual information regardless of an artifact type.

Results: Table V summarizes the average of precision and recall metrics when identifying task-relevant textual information for all of the test data. Based on the overall results for each technique, we observe that recall scores range from 0.43 to 0.58. We also note that applying sentence-level filters improves precision and recall values for word2vec. Notably, word2vec with the frame-elements filter achieves up to 0.49 recall. For BERT, recall values range from 0.56 to 0.58 and, similar to our evaluation with Stack Overflow artifacts, we observe that sentence-level filters do not provide substantial changes.

When we compare evaluation metrics to a baseline using VSM, we find that the baseline achieves precision and recall scores of 0.30 and 0.33 for the same data. Although this result is not surprising, it suggests that a developer using keyword-matching could miss much of the task-relevant textual information in an artifact.

Table V also details evaluation metrics per type of artifact. Certain techniques, such as word2vec with

TABLE V: Evaluation Metrics Over All Artifacts and Artifact-type Wise

	artifact	<i>no filter</i>		<i>w/ frame-elements</i>		<i>w/ frame-associations</i>	
		precision	recall	precision	recall	precision	recall
word2vec	API documentation	0.47	0.39	0.47	0.40	0.53	0.52
	GitHub issues	0.40	0.36	0.60	0.54	0.44	0.36
	Stack Overflow answers	0.43	0.48	0.49	0.45	0.45	0.50
	Miscellaneous pages	0.49	0.49	0.50	0.45	0.50	0.47
	overall	0.45	0.43	0.51	0.46	0.48	0.46
	standard deviation	0.03	0.05	0.05	0.05	0.03	0.06
BERT	API documentation	0.52	0.55	0.51	0.57	0.52	0.55
	Github issues	0.52	0.56	0.53	0.55	0.52	0.56
	Stack Overflow answers	0.58	0.63	0.53	0.58	0.58	0.62
	Miscellaneous pages	0.52	0.56	0.51	0.54	0.53	0.57
	overall	0.54	0.58	0.52	0.56	0.54	0.58
	standard deviation	0.02	0.03	0.01	0.01	0.02	0.02

frame-elements, have better results in specific types of artifacts, such as GitHub issues. However, word2vec results are significantly lower for other types, such as API documentation or Stack Overflow answers. Without filters, BERT performs better at Stack Overflow. For this technique, we also find that filters decrease the amount of variation in the evaluation metrics. Notably, BERT with frame-associations is the technique with the highest recall and second lowest standard deviation suggesting that the technique performs well across different types of artifacts.

When we compare techniques without any filters, differences between word2vec and BERT may be explained by how each of these two approaches compute relevance. That is, all the words in a sentence contribute equally to word2vec’s identification of task-relevant text via its semantic similarity function. If an artifact lacks much of the text that appears in a task description, this technique may not be able to identify the text deemed relevant. BERT shortens this gap because relevance is computed via its attention mechanism, which might assign more weight to words key to determining that the text is relevant to a task thus, explaining why the technique has better evaluation metrics.

With sentence-level filters, there are improvements to most of the types of artifacts evaluated. However, in certain techniques, the filters decreased the text identified, as when using the frame-elements filter in Stack Overflow artifacts. It is possible that the text in these artifacts lacks any of the frames that we defined as meaningful, and thus, the reason why this filter reduced the amount of text identified.

Results from our evaluation across all artifact types indicate that semantic techniques can find up to 58% of the task-relevant textual information in an artifact. Some of our techniques also show consistent results over different types of artifacts, suggesting that semantic techniques may generalize across a variety of natural language software artifacts.

VI. THREATS

We rely on the golden data for DS-android to evaluate techniques, which impacts the conclusions we draw. This golden data in turn relies on the human annotations of text in artifacts that is related to a task-at-hand. There is a possibility that this text marked by the annotators might not contain information associated with the solution for the task, or that an annotator missed highlighting text that they deemed relevant. We minimize this threat by considering only the sentences marked by at least two annotators as the golden data used in the evaluation of techniques. We also asked annotators to write a short set of instructions for the tasks they annotated and provided them an in-house tool that streamlined annotation procedures. These procedures help mitigate risks to the validity of our conclusions. We note that there are no objective means to quantify which text assists task-completion [69], [70].

The internal validity of the evaluation may be impacted by the differing number of types of artifacts in the data set. While most of the tasks in the data set have associated Stack Overflow artifacts, not as many have artifacts from GitHub. These differences may affect the training and testing of each technique. These risks are mitigated in that the content of Stack Overflow and GitHub artifacts is similar in length in the data set (Table II).

The selection of tasks in the Android development domain could affect the generalizability of our work. Most notably, aspects such as programming languages, frameworks, associated technologies, and others [45] influence the information sought by a developer as well as what they find relevant. The three annotators that we recruited to construct the DS-android data set are also not representative of the entire developer population. We leave the investigation of software tasks in other programming languages or domains—and their associated natural language software artifacts—to future work.

VII. DISCUSSION

Our results show that it is possible to create semantic-based techniques for identifying task-relevant information with

comparable performance to a state-of-the-art technique that is more targeted. Although comparable in performance, artifact-specific techniques may still perform better overall raising the question of whether a general technique is needed. We explore this question as well as discussing potential improvements to the semantic-based techniques. We also introduce potential tools that can leverage the broader applicability of the semantic-based techniques.

A. Specificity or Generality?

From the evaluation of techniques we conducted, we observe that techniques specific to a particular kind of artifact demonstrate the best recall value. If such techniques, such as AnswerBot, can already identify most of the text considered relevant in the artifacts that they target, one may ask what is the place of more general approaches.

We argue that techniques that might apply to more than one kind of artifact may be more robust to a variety of kinds of tasks that occur during development and to new kinds of artifacts that are introduced. In particular, studies have shown that the information sources sought by a developer vary according to a task's type [56]. On the one hand, the variety of task types, the variety of artifacts needed, and the introduction of new artifact types pose challenges to techniques that are artifact specific and may favor general techniques. On the other hand, research on the web search behaviour of developers suggests that certain sources are well-established within many development communities and thus, artifact-specific techniques tailored to these sources might identify task-relevant textual information more accurately. A good path forward may be to use a complement of artifact-specific and general approaches.

B. Limitations & Potential Improvements

We have shown that semantic-based approaches identify task-relevant textual information across different types of artifacts without relying on assumptions about the nature of an artifact or its structured data. However, we observe that the techniques with the best accuracy require training data. Producing this training data might require significant manual effort and this could be considered as an impediment for the use of such supervised techniques.

Although we acknowledge that the need for training data is a limitation inherent to supervised techniques, the fact that artifact-specific techniques, such as AnswerBot, have accuracy comparable to semantic-based approaches provides a unique opportunity for the creation of synthetic data that one might use for training purposes. In a similar manner to how researchers created word embeddings for the software engineering domain [32], one might gather significantly large amounts of data from online resources such as Stack Overflow and use techniques, such as AnswerBot, that apply to such resources to create the data needed for training.

One requirement to using such a bootstrapping mechanism is that one must show that techniques trained on synthetic data do not simply replicate the underlying heuristics of the

techniques used for bootstrapping. That is, we have to show that a technique trained in this manner correlates data from a task and an artifact in ways that extend beyond what a technique used for bootstrapping already provides. For example, showing that it can identify task-relevant textual information in a different type of artifact or showing that its accuracy is significantly better than the technique used for bootstrapping. We leave this investigation for future work.

C. Tool Ideas

We describe two potential tools based on semantic techniques to assist developers in locating information associated with the solution for a software task.

A first tool, in the form of a web browser plugin, could assist developers with real-time navigational data. A developer could provide to the tool a link to their current task and, whenever a developer navigated to a new web page, the tool could highlight text automatically identified as relevant to the developer's task.

A second tool takes inspiration from Hipikat—a tool that implicitly builds a group memory for a software system based on the artifacts produced during a task [71]. While Hipikat makes recommendations at the artifact level, Cubranic et al. had already acknowledged that natural language text could be used as a recommendation unit. For this type of tool, developers could use a feedback mechanism to confirm if the text identified by a technique was indeed relevant to their tasks, saving this knowledge in the project memory.

VIII. CONCLUSION

Given that only a fraction of the text of an artifact contains information associated with the solution of a developer's task, researchers have proposed specialized techniques (and tools) to help developers locate text relevant to a task-at-hand. Although effective, such techniques do not apply to all the different types of artifacts that a developer may come across in their daily work, what limits their usage.

As an alternative, this paper proposes six possible techniques that incorporate semantics of words and sentences to identify task-relevant text across a range of natural language artifacts. We compare our proposed techniques to a state-of-the-art technique, AnswerBot, specific to Stack Overflow artifacts and we evaluate them over different types of artifacts.

Evaluation results show that semantic-based techniques achieve recall comparable to AnswerBot, but without the need for artifact-specific data, and that some of our proposed techniques perform equivalently well across multiple artifact types. In the future, we plan to investigate how to embed our techniques into practical tools, and how developers benefit from such tools. We publicly share the dataset produced as part of our work to help future research in the field [9].

ACKNOWLEDGMENTS

The authors would like to thank Alison Li, Katharine Kerr, and Tarcísio Teixeira for annotating the dataset. This research was funded in part by a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant.

REFERENCES

- [1] G. C. Murphy, M. Kersten, M. P. Robillard, and D. Čubranić, "The emergent structure of development tasks," in *ECOOP 2005 - Object-Oriented Programming*, A. P. Black, Ed., 2005, pp. 33–48.
- [2] M. P. Robillard and Y. B. Chhetri, "Recommending reference api documentation," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1558–1586, Dec. 2015.
- [3] R. F. Silva, C. K. Roy, M. M. Rahman, K. A. Schneider, K. Paixao, and M. de Almeida Maia, "Recommending comprehensive solutions for programming tasks by mining crowd knowledge," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 358–368.
- [4] B. Xu, Z. Xing, X. Xia, and D. Lo, "AnswerBot: Automated generation of answer summary to developers' technical questions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 706–716.
- [5] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun, "Combining word embedding with information retrieval to recommend similar bug reports," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 127–137.
- [6] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshvanyk, and V. Ng, "Assessing the quality of the steps to reproduce in bug reports," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2019, pp. 86–96.
- [7] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS)*, 2013, pp. 3111–3119.
- [8] C. J. Fillmore, "Frame semantics and the nature of language," *Annals of the New York Academy of Sciences*, vol. 280, no. 1, pp. 20–32, 1976.
- [9] *Evaluating the Use of Semantics for Identifying Task-relevant Textual Information Supplementary Material*. Zenodo, Oct. 2021. [Online]. Available: <https://doi.org/10.5281/zenodo.5826124>
- [10] Y. S. Maarek and F. Z. Smadja, "Full text indexing based on lexical relations: an application: Software libraries," *SIGIR Forum*, vol. 23, no. SI, pp. 198–206, May 1989.
- [11] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using ir methods for labeling source code artifacts: Is it worthwhile?" in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 2012, pp. 193–202.
- [12] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: A case study of bug reports," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, 2010, pp. 505–514.
- [13] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the hurried bug report reading process to summarize bug reports," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 430–439.
- [14] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng, "Detecting missing information in bug descriptions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2017, pp. 396–407.
- [15] L. Ponzanelli, "Holistic recommender systems for software engineering," in *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 2014, pp. 686–689.
- [16] S. Nadi and C. Treude, "Essential sentences for navigating stack overflow answers," in *2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2020, pp. 229–239.
- [17] H. Jiang, J. Zhang, X. Li, Z. Ren, and D. Lo, "A more accurate model for finding tutorial segments explaining apis," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 157–167.
- [18] L. Ponzanelli, A. Mocci, and M. Lanza, "Summarizing complex development artifacts by mining heterogeneous data," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 401–405.
- [19] C. Chen, S. Gao, and Z. Xing, "Mining Analogical Libraries in Q&A Discussions – Incorporating Relational and Categorical Knowledge into Word Embedding," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 338–348.
- [20] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from stack overflow," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 392–403.
- [21] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "Api method recommendation without worrying about the task-api knowledge gap," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 293–304.
- [22] G. Bavota, "Mining unstructured data in software repositories: Current and future trends," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, 2016, pp. 1–12.
- [23] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, 2003, pp. 125–135.
- [24] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 404–415.
- [25] W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," in *IEEE Transactions on Software Engineering*, vol. 39, no. 9, 2013, pp. 1264–1282.
- [26] D. Arya, W. Wang, J. L. Guo, and J. Cheng, "Analysis and detection of information types of open source software issue discussions," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 454–464.
- [27] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "Development emails content analyzer: Intention mining in developer discussions (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 12–23.
- [28] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [29] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2009.
- [30] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.
- [31] A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni, "What you can cram into a single vector: Probing sentence embeddings for linguistic properties," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, Jul. 2018, pp. 2126–2136.
- [32] V. Efstathiou, C. Chatzilenas, and D. Spinellis, "Word embeddings for the software engineering domain," in *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 38–41.
- [33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 6000–6010.
- [34] A. Marques, N. C. Bradley, and G. C. Murphy, "Characterizing task-relevant information in natural language software artifacts," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 476–487.
- [35] A. Marques, G. Viviani, and G. C. Murphy, "Assessing semantic frames to support program comprehension activities," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, 2021, pp. 13–24.
- [36] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.
- [37] "apriori-python," <https://pypi.org/project/apriori-python/>, verified: 2021-10-11.
- [38] S. Baltes, C. Treude, and M. P. Robillard, "Contextual documentation referencing on stack overflow," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.
- [39] L. Li, J. Gao, T. Bissyand, L. Ma, X. Xia, and J. Klein, "Characterising deprecated android apis," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, 2018, pp. 254–264.
- [40] B. G. Mateus and M. Martinez, "On the adoption, usage and evolution of kotlin features in android development," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020.

- [41] M. Linares-Vázquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "How do api changes trigger stack overflow discussions? a study on the android sdk," in *Proceedings of the 22nd International Conference on Program Comprehension (ICPC)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 83–94.
- [42] G. Bavota, M. Linares-Vázquez, C. E. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "The impact of api change- and fault-proneness on the user ratings of android apps," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 384–407, 2015.
- [43] T. McDonnell, B. Ray, and M. Kim, "An empirical study of api stability and adoption in the android ecosystem," in *2013 IEEE International Conference on Software Maintenance*, 2013, pp. 70–79.
- [44] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow," *Georgia Institute of Technology, Tech. Rep.*, vol. 11, 2012.
- [45] S. Baltes and S. Diehl, "Usage and attribution of stack overflow code snippets in github projects," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1259–1295, 2019.
- [46] H. Borges and M. T. Valente, "Whats in a github star? understanding repository starring practices in a social coding platform," *Journal of Systems and Software*, vol. 146, pp. 112–129, 2018.
- [47] L. Xavier, F. Ferreira, R. Brito, and M. T. Valente, "Beyond the code: Mining self-admitted technical debt in issue tracker systems," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, 2020, pp. 137–146.
- [48] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 92–101.
- [49] S. Baltes, C. Treude, and M. P. Robillard, "Contextual Documentation Referencing on Stack Overflow Supplementary Material," Mar. 2019. [Online]. Available: <https://doi.org/10.5281/zenodo.2585828>
- [50] S. Baltes, C. Treude, and S. Diehl, "Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 191–194.
- [51] M. Yazdaniinia, D. Lo, and A. Sami, "Characterization and prediction of questions without accepted answers on stack overflow," in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, 2021, pp. 59–70.
- [52] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1178–1193, 2017.
- [53] N. Rao, C. Bansal, T. Zimmermann, A. H. Awadallah, and N. Nagappan, "Analyzing web search behavior for software engineering tasks," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 768–777.
- [54] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" *Empirical Softw. Engg.*, vol. 22, no. 6, pp. 3149–3185, Dec. 2017.
- [55] J. Starke, C. Luce, and J. Sillito, "Searching and skimming: An exploratory study," in *2009 IEEE International Conference on Software Maintenance*, 2009, pp. 157–166.
- [56] H. Li, Z. Xing, X. Peng, and W. Zhao, "What help do developers seek, when and how?" in *2013 20th Working Conf. on Reverse Engineering (WCRE)*, Oct 2013, pp. 142–151.
- [57] "googlesearch," <https://pypi.org/project/googlesearch-python/>, verified: 2020-10-27.
- [58] "Alexa," <https://alexa.com>, verified: 2021-04-08.
- [59] F. N. A. Al Omran and C. Treude, "Choosing an NLP library for analyzing software documentation: A systematic literature review and a series of experiments," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 187–197.
- [60] D. Piorkowski, A. Z. Henley, T. Nabi, S. D. Fleming, C. Scaffidi, and M. Burnett, "Foraging and navigations, fundamentally: Developers' predictions of value and cost," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2016, pp. 97–108.
- [61] K. Krippendorff, *Content analysis: An introduction to its methodology*. Sage publications, 2018.
- [62] A. Nenkova and R. Passonneau, "Evaluating content selection in summarization: The pyramid method," in *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2004, pp. 145–152.
- [63] D. M. Arya, J. L. Guo, and M. P. Robillard, "Information correspondence between types of documentation for apis," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4069–4096, 2020.
- [64] H. Jiang, J. Zhang, Z. Ren, and T. Zhang, "An unsupervised approach for discovering relevant tutorial fragments for APIs," in *2017 IEEE/ACM 39th International Conf. on SE*, 2017, pp. 38–48.
- [65] C. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval," *Natural Language Engineering*, vol. 16, no. 1, pp. 100–103, 2010.
- [66] H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50 – 60, 1947.
- [67] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, Dec. 2006.
- [68] L. Freund, "Contextualizing the information-seeking behavior of software engineers," *Journal of the Association for Information Science and Technology*, vol. 66, no. 8, pp. 1594–1605, aug 2015.
- [69] T. Saracevic, "Relevance: A review of the literature and a framework for thinking on the notion in information science. part iii: Behavior and effects of relevance," in *Journal of the American Society for Information Science and Technology*, vol. 58, no. 13, 2007, pp. 2126–2144.
- [70] —, "Relevance: A review of the literature and a framework for thinking on the notion in information science. part ii: nature and manifestations of relevance," in *Journal of the American Society for Information Science and Technology*, vol. 58, no. 13, 2007, pp. 1915–1933.
- [71] D. Cubranic, G. Murphy, J. Singer, and K. Booth, "Hipikat: a project memory for software development," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 446–465, 2005.