Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems | School of Computing and Information Systems

5-2016

Augmenting API documentation with insights from stack overflow

Christoph TREUDE
Singapore Management University, ctreude@smu.edu.sg

Martin P. ROBILLARD

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the Software Engineering Commons

Citation

TREUDE, Christoph and ROBILLARD, Martin P.. Augmenting API documentation with insights from stack overflow. (2016). *Proceedings of the 38th IEEE International Conference on Software Engineering (ICSE), Austin, TX, USA, 2016 May 14-22*. 392-403.
Available at: https://ink.library.smu.edu.sg/sis_research/8938

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Augmenting API Documentation with Insights from Stack Overflow

Christoph Treude
School of Computer Science
University of Adelaide
Adelaide, SA, Australia
christoph.treude@adelaide.edu.au

Martin P. Robillard
School of Computer Science
McGill University
Montréal, QC, Canada
martin@cs.mcgill.ca

ABSTRACT

Software developers need access to different kinds of information which is often dispersed among different documentation sources, such as API documentation or Stack Overflow. We present an approach to automatically augment API documentation with "insight sentences" from Stack Overflow—sentences that are related to a particular API type and that provide insight not contained in the API documentation of that type. Based on a development set of 1,574 sentences, we compare the performance of two state-of-the-art summarization techniques as well as a pattern-based approach for insight sentence extraction. We then present SISE, a novel machine learning based approach that uses as features the sentences themselves, their formatting, their question, their answer, and their authors as well as part-of-speech tags and the similarity of a sentence to the corresponding API documentation. With SISE, we were able to achieve a precision of 0.64 and a coverage of 0.7 on the development set. In a comparative study with eight software developers, we found that SISE resulted in the highest number of sentences that were considered to add useful information not found in the API documentation. These results indicate that taking into account the meta data available on Stack Overflow as well as part-of-speech tags can significantly improve unsupervised extraction approaches when applied to Stack Overflow data.

CCS Concepts

•**Information systems** → **Information extraction;** •**Software and its engineering** → **Documentation;** •**Computing methodologies** → *Supervised learning;*

Keywords

API documentation, Stack Overflow, insight sentences

1. INTRODUCTION

While much of the information needed by software developers is captured in some form of documentation, it is often not obvious where a particular piece of information is stored. Different documentation formats, such as wikis or blogs, contain different kinds of information, written by different individuals and intended for different purposes [38]. For instance, API documentation captures information about functionality and structure, but lacks other types of information, such as concepts or purpose [18]. Some of the most severe obstacles faced by developers learning a new API are related to its documentation [32], in particular because of scarce information about the API's design, rationale [31], usage scenarios, and code examples [32].

On the other hand, "how-to" questions [35] (also referred to as "how-to-do-it" questions [10]) are the most frequent question type on the popular Question and Answer (Q&A) site Stack Overflow, and the answers to these questions have the potential to complement API documentation in terms of concepts, purpose, usage scenarios, and code examples. While a lot of research has focused on finding code examples for APIs (e.g., [17], [33]), less work has been conducted on improving or augmenting the natural language descriptions contained in API documentation.

To fill this gap, we compare techniques for automatically extracting sentences from Stack Overflow that are related to a particular API type and that provide insight not contained in the API documentation. We call these sentences *insight sentences*. The idea is related to update summarization [4], which attempts to summarize new documents assuming that the reader is already familiar with certain old documents. Update summarization is often applied to summarizing overlapping news stories. Applied to API documentation and content from Stack Overflow, the idea is to create a summary of the discussions on Stack Overflow as they relate to a given API type, assuming that the reader is already familiar with the type's API documentation.

Our research is guided by two main questions:

RQ1: To what extent are unsupervised and supervised approaches able to identify meaningful insight sentences?

RQ2: Do practitioners find these sentences useful?

To answer the first research question, we applied two state-of-the-art extractive summarization techniques—LexRank [13] and Maximal Marginal Relevance (MMR) [4]—to a set of API types, their documentation, and related Stack Overflow threads. These techniques assign a numeric value to each sentence and return the top-ranked sentences as a summary. We found these summarization techniques to perform poorly on our data, mainly because sentences on Stack Overflow are often not

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE '16, May 14-22, 2016, Austin, TX, USA

© 2016 ACM. ISBN 978-1-4503-3900-1/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2884781.2884800

Table 1: Motivating Examples

API type	Stack Overflow sentence	Stack Overflow id
java.sql.DriverManager	Normally you use DriverManager when you just want a connection for one time while with DataSource you get other features such as connection pooling and distributed transactions.	10868758
java.net.URL	URLEncoder is meant for passing data as parameters, not for encoding the URL itself.	724043
java.lang.Thread	join() waits for something meaningful while sleep() just sits there doing nothing.	4561951

Table 2: Regular Expressions for filtering Stack Overflow Threads

question part	pattern
body	`.*(^|[a-z]+ |[\.!?] |[\(<])TypeName([>\)\.,!?$]| [a-z]+).*` *(non-qualified API type surrounded by lower case words or punctuation marks)*
title or body	`(?i).*\bPackageName\.TypeName\b.*` *(fully-qualified API type, case-insensitive)*
body	`.*<code>.*\bTypeName\b.*</code>.*` *(non-qualified API type in code)*
body	`.*<a.*href.*PackageName/TypeName\.html.*>.*</a>.*` *(link to the official API documentation)*
title	`(?i).*\b(a |an )TypeName\b.*` *(non-qualified API type prefixed with "a" or "an", case-insensitive)*

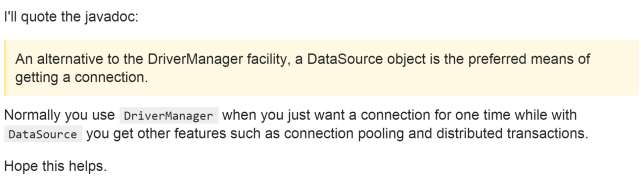

Figure 1: Stack Overflow Example

meaningful on their own without their surrounding code snippets or the question that prompted a given answer. We also applied a pattern-based approach that has been successfully used to detect and recommend fragments of API documentation potentially important to a programmer who has already decided to use a certain API element [7], with similar results. We then developed a novel machine learning approach called SISE (Supervised Insight Sentence Extractor), which uses as features the sentences themselves, their formatting, their question, their answer, and their authors as well as part-of-speech tags and the similarity of a sentence to the corresponding API documentation. With SISE, we were able to achieve a precision of 0.64 and a coverage of 0.7 on our development set.[1] In addition to the similarity of a sentence to its corresponding API documentation, characteristics of the user asking the question, the score and age of the answer, question characteristics, and the part-of-speech tags at the beginning of a sentence were among the features with the highest information gain.

To answer the second research question on whether practitioners find these sentences useful, we conducted a comparative study in which we asked eight software developers to rate sentences extracted with all four approaches (LexRank, MMR, patterns, and SISE). These sentences were related to 20 Java API types. The study showed that sentences extracted by SISE were considered significantly more meaningful and resulted in the most sentences that added useful information not contained in the API documentation. These results indicate that taking into account Stack Overflow meta data as well as part-of-speech tags can significantly improve existing unsupervised approaches when applied to Stack Overflow data.

The main contributions of this work are the conceptualization of insight sentences as sentences from one documentation source that provide insight to other documentation sources and a comparative study of four different approaches for the extraction of such insight sentences, as well as a list of factors that can be used to distinguish insight sentences from sentences that are not meaningful or do not add useful information to another documentation source.

2. MOTIVATING EXAMPLES

Table 1 shows, for three Java API types, a sentence taken from Stack Overflow that contains useful information about this type that is not stated in the type's API documentation. The goal of our work is to automatically extract such sentences and use them to augment API documentation. The first example is taken from our development set while the other two examples were automatically identified by SISE.

Figure 1 shows the source of the sentence in the first example. In an answer to a question about *"DataSource and DriverManager on J2SE"*, the Stack Overflow user first cites a statement from the API documentation, but then elaborates on it further than the API documentation does. This is an example of the API documentation lacking information on purpose [18] since it only states which alternative is preferred without explaining why. In contrast, the sentence added by the Stack Overflow user clearly distinguishes the roles of the API types discussed and explains which one should be used in which situation. The second example shows a case where a Stack Overflow user clarifies the relationship between types with similar names in an answer

[1]Precision measures the fraction of sentences extracted by an approach that are meaningful while coverage measures the ratio of API types for which the approach extracts at least one sentence.

to *"HTTP URL Address Encoding in Java"*. In the third example, a user again compares two alternatives and explains which one to use in a particular situation, this time in an answer to *"Thread sleep and thread join"*.

In the next sections, we describe our investigation of the means to automatically identify sentences on Stack Overflow that are meaningful and add useful information not contained in the API documentation.

3. LINKING DOCUMENTATION

First, we describe how we identify Stack Overflow threads related to a given API type as well as our development set used for comparing different extraction approaches.

3.1 Linking

Our linking approach identifies Stack Overflow threads that are related to an API type in a two-step process: (1) we perform a full-text query for the non-qualified type name using the Stack Overflow API relying on its "relevance" ordering, and (2) we reject all results that do not match at least one of a number of regular expressions that are applied to the question that started the thread (see Table 2). The advantage of using the Stack Overflow API over the Stack Overflow data dump used in previous research such as that of Bacchelli et al. [2] is that sentences extracted by our linking approach always reflect the latest content available on Stack Overflow. However, the querying mechanism offered by the Stack Overflow API is limited, thus warranting the additional filtering in step (2).

We manually created a benchmark to measure the relevance of the threads that the filter identifies in step (2). For 40 randomly selected types of the Java SE 7 API (20 types with one-word identifiers, such as `List`, and 20 types with multi-word identifiers, such as `ArrayList`), we selected the first five threads that the Stack Overflow API returned for each non-qualified type, and we manually annotated whether the thread actually mentioned the API type. We evaluated our linking approach separately for one-word types and multi-word types. For one-word types, precision was 0.82 and recall was 1.0 (F1-measure: 0.90), and for multi-word types, precision was 0.92 and recall was 0.97 (F1-measure: 0.94).

3.2 Annotated Data

We created a development set by manually annotating all 1,574 sentences that belong to the top ten answers (as indicated by their scores) from the first ten threads that our linking approach associated with ten Java API types. We did not remove false positives (threads not mentioning a target type) to ensure that the development set is a realistic representation of the sentences used as input to the extraction approaches. To sample API types, we identified the three most commonly used types from each of the ten most commonly used Java packages, as indicated by the Apatite tool [12], see Table 3. We chose this stratified sampling strategy to ensure a wide coverage of commonly used types while not focusing on rarely used ones. We then divided the data into the development set and the test set for the comparative study (see Section 6) as follows: The second most commonly used type from each package was used to construct the development set, and the most commonly used type as well as the third most commonly used type was used for the comparative study.

Table 3: Three most commonly used types in the ten most commonly used Java packages. Types in bold were used for sampling sentences in the development set and the remaining types were used to sample sentences for the comparative study.

package	type
java.applet	Applet, **AudioClip**, AppletContext
java.awt	Event, **Image**, Component
java.beans	PropertyChangeListener, **PropertyChangeEvent**, PropertyDescriptor
java.io	File, **Serializable**, InputStream
java.lang	Object, **String**, Thread
java.net	URL, **URLClassLoader**, Socket
java.security	AccessController, **SecureClassLoader**, Principal
java.sql	Connection, **DriverManager**, ResultSet
java.util	List, **ArrayList**, Map
javax.swing	JComponent, **JPanel**, JFrame

Table 4: Development Set

type	meaningful	not meaningf.
ArrayList	58	235
AudioClip	7	45
DriverManager	14	68
Image	12	151
JPanel	8	129
PropertyChangeEvent	3	101
SecureClassLoader	3	62
Serializable	48	111
String	35	437
URLClassLoader	8	39
sum	**196**	**1,378**

The first author annotated each of the 1,574 sentences belonging to the second most commonly used type in each of the ten packages with a yes/no rating to indicate whether it was meaningful on its own.[2] 196 sentences were rated as meaningful. Table 4 shows the number of sentences considered meaningful for each of the ten API types.

As an example, the first author annotated the following three sentences related to Java's `ArrayList` as being meaningful: *"The list returned from asList has fixed size"*, *"There is one common use case in which LinkedList outperforms ArrayList: that of a queue"*, and *"It's worth noting that Vector also implements the List interface and is almost identical to ArrayList"*. Examples for sentences that are not meaningful and related to the same API type are: *"See the next step if you need a mutable list"*, *"They serve two different purposes"*, and *"Use the Javadocs"*. All of these sentences make sense in the context of an entire question-and-answer thread, but they do not convey meaningful information on their own.

4. UNSUPERVISED APPROACHES

In this section, after outlining our preprocessing steps, we present the results of using state-of-the-art text summarization and pattern-based approaches for the extraction of insight sentences.

[2]The complete coding guide is available in our online appendix at http://cs.mcgill.ca/~swevo/insight/.

Table 5: Precision and Coverage for different LexRank Configurations

	precision	coverage
only first sentence	0.40	1.0
first two sentences	0.25	1.0
first three sentences	0.23	1.0
first four sentences	0.23	1.0
first five sentences	0.24	1.0
score at least 0.005	0.14	0.9
score at least 0.010	0.13	0.8
score at least 0.015	0.16	0.4
score at least 0.020	0.21	0.2

We had developed a set of techniques for preprocessing software documentation in previous work [36, 37]. We summarize them here for completeness.

We remove markup from the HTML files and preprocess the resulting text files to account for the unique characteristics of software documentation not found in other texts, such as the systematic use of incomplete sentences and the presence of code terms. In particular, we prefix sentences that start with a verb in present tense, third person singular, such as *"returns"* or *"computes"*, with the word *"this"* to ensure correct parsing of partial sentences, such as *"Returns the next page number"*. In addition, we prefix sentences that start with a verb in present participle or gerund, such as *"adding"* or *"removing"*, immediately followed by a noun, with the word *"for"* to ensure correct parsing of partial sentences, such as *"Displaying data from another source"*.[3] We further configure the Stanford NLP parser [19] to automatically tag all code elements as nouns. In addition to code elements tagged with `tt` or `code` tags in the original source, all words that match one of about 30 hand-crafted regular expressions are treated as code elements.[4] The resulting sentences are then parsed using the Stanford NLP toolkit.

These preprocessing steps are identical for all the approaches described below.

4.1 LexRank

LexRank is a text summarization technique that conceptualizes a text as a graph where each node represents a sentence and the weight of each edge corresponds to the cosine similarity of the sentences it connects.[5] The importance of a sentence is then given by the eigenvector centrality of the corresponding node [13]. We chose LexRank because it is the best-known graph-based method for summarization [24].

We re-implemented LexRank in Java and calculated the eigenvector centrality of each sentence in our development set, separately for each API type.[6] For each API type, we considered all related sentences in our development set as the text to be summarized. The result is a numeric score for each sentence. There is no clear rule as to how many sentences should be considered for a summary or what a good threshold for eigenvector centrality is. Thus, we experimented with different settings and evaluated the performance of different configurations on our development set. Table 5 shows the results in terms of precision and coverage. We define coverage as the ratio of API types for which there is at least one sentence. We focus on coverage instead of recall because our goal is the extraction of useful insight sentences and not the identification of all possible insight sentences. If we only consider the sentence with the highest eigenvector centrality to be the insight sentence for each API type, the average precision across the ten API types in our development set is 0.40. The precision drops if we consider more sentences as insight sentences. We also explored the effects of different eigenvector centrality thresholds. As Table 5 shows, the precision remains low and coverage drops as well.

[3]Using other prepositions, such as *"by"*, does not significantly change the results.

[4]The regular expressions are available in our online appendix at http://cs.mcgill.ca/~swevo/insight/.

[5]similarity$(A, B) = |A \cap B|/(\sqrt{|A| \times |B|})$, where A and B are the tokens of the respective sentence.

[6]We implemented our own version of LexRank to be able to modify it if needed. However, all results in this paper are based on an unmodified implementation following Erkan and Radev [13].

Table 6: Precision and Coverage for different MMR Configurations

	precision	coverage
only first sentence	0.20	1.0
first two sentences	0.10	1.0
first three sentences	0.10	1.0
first four sentences	0.15	1.0
first five sentences	0.16	1.0
score at least 0.005	0.13	0.8
score at least 0.010	0.14	0.5
score at least 0.015	0.11	0.3
score at least 0.020	0.14	0.1

4.2 Update Summarization

The goal of update summarization is to produce a summary of a new document under the assumption that the reader is already familiar with the content of a given set of old documents. Applied to insight sentence extraction, the idea is to create a summary of Stack Overflow threads related to an API type under the assumption that the reader is already familiar with the type's API documentation.

As previously done by Boudin et al. [4], we adopted the concept of Maximal Marginal Relevance (MMR) in our implementation of update summarization, using the LexRank score as a baseline for calculating the MMR scores (see previous section). We subtracted from each sentence's LexRank score the maximum cosine similarity between that sentence and any sentence in the API type's documentation. In other words, if the similarity between a sentence and each sentence in the API documentation is 0, the MMR score is identical to the one assigned by LexRank. However, sentences that are similar to at least one sentence in the API documentation receive a score lower than the one assigned by LexRank.

Table 6 shows precision and coverage for different configurations of our MMR implementation when applied to the development set. The results are worse than those for LexRank.

4.3 Knowledge Patterns

Previous research has successfully used knowledge patterns to detect and recommend fragments of API documentation potentially important to a programmer using an API element. In their work, Chhetri and Robillard [7] categorize text fragments in API documentation based on whether they contain information that is indispensable, valuable, or neither. From the fragments that contain potentially im-

Table 7: Features used for SISE

source	feature	type
sentence	sentence	string
	part-of-speech tags	string
sentence	the number of tokens in the sentence	numeric
	whether the sentence is a codeblock (i.e., surrounded by `pre` tags)	boolean
	the position of the sentence in the answer	numeric
	the position of the API element in the sentence (or 0)	numeric
	whether the sentence starts lower case	boolean
	the number of characters that are code (as indicated by `pre` or `code` tags)	numeric
	whether sentence contains HTML tag (code, pre, a, strong, em, i, b, h1, h2, h3, sup, strike)	boolean
	the percentage of tokens tagged (code, pre, a, strong, em, i, b, h1, h2, h3, sup, strike)	numeric
question	whether the question title or body contain the API element	boolean
	question attributes: score, favorites, views, answer count, size, age	numeric
	question attributes: whether it was edited, whether it contains a code block	boolean
	question user attributes: reputation, acceptrate	numeric
	whether the question user is registered	boolean
answer	answer attributes: score, time difference to question, size, age	numeric
	answer attributes: whether it was accepted or edited, whether it contains a code block	boolean
	answer user attributes: reputation, acceptrate	numeric
	whether the answer user is registered	boolean
	relative rank of the answer among all answers to that question by score and age	numeric
similarity	cosine similarity between sentence and most similar sentence in API documentation	numeric
	average cosine similarity between sentence and all sentences in API documentation	numeric

portant knowledge, they extract word patterns to automatically find new fragments that contain similar knowledge in unseen documentation. In a study involving independent human participants, indispensable knowledge items recommended for API types were judged useful 57% of the time and potentially useful an additional 30% of the time.

Each knowledge pattern consists of a small number of words, such as *"specify, should, argument"* or *"not, exceed, number, should"*. To match a knowledge pattern, a sentence must contain all of the words in the pattern, but not necessarily in the order specified by the pattern. Instead of a word, a pattern can contain a special wildcard for code elements which indicates that only sentences containing a code element can match the pattern. An example is given by the pattern *"must, not, `<code element>`, null"*.

We applied the 361 patterns for indispensable and valuable knowledge items extracted from the reference documentation of the Java 6 SE SDK to our development set and calculated precision and coverage. To ensure that the matching of sentences to patterns is not sensitive to different grammatical forms, we applied stemming to each word in the patterns and the sentences using Porter's stemmer [27] before calculating the matches. Similar to the attempts of using text summarization techniques for the extraction of insight sentences, the application of knowledge patterns to our development set did not produce encouraging results—we obtained a precision of 0.15 and a coverage of 0.8. This can be explained by the patterns' reliance on the systematic writing style of reference documentation which is not used in informal documentation formats such as Stack Overflow.

5. SISE: A SUPERVISED APPROACH

Considering the poor results achieved with traditional summarization approaches, we developed SISE (Supervised Insight Sentence Extractor), a novel approach specifically for extracting insight sentences from Stack Overflow. A supervised approach efficiently supports considering a large amount of potential factors available for each sentence on the Q&A website, such as the reputation of the user authoring a sentence or the score of the corresponding answer.

After the preprocessing steps described in Section 4, we used the features shown in Table 7 for each sentence as input for machine learning algorithms. We designed the feature set to cover all meta data available on Stack Overflow as well as basic syntactic and grammatical features. In addition, we included two features for the similarity of a sentence to the corresponding API documentation, following the idea of update summarization [4]. Most of the features are either *boolean* or *numeric*. For the two *string* features, we used WEKA's [15] `StringToWordVector` filter to turn the corresponding text into separate features for single words, bigrams, and trigrams. For example, the simple sentence *"`List` is slower than `Arrays`"* would result in one feature for each lemmatized word (*"`<code element>`"*, *"be"*, etc.), four features for the bigrams (*"`<code element>` be"*, *"be slow"*, etc.), and three features for the trigrams (*"`<code element>` be slow"*, *"be slow than"*, etc.). For the part-of-speech tags feature set, the same number of features would be produced based on the part-of-speech text that corresponds to the sentence, which in the example is *"`NN VBZ JJR IN NN`"* (a noun followed by a verb in third person singular, a comparative adjective, a preposition, and another noun).

We tested different machine learning algorithms that are commonly used for text classification in software engineering (e.g., [10]) on our development set: k-nearest neighbour (Ibk) [1], decision trees (J48) [28], Naive Bayes [16], random forest [5], and support vector machines (SMO, the sequential minimal optimization implementation in WEKA) [23]. Apart from J48 and random forest, all classifiers belong to different classes, ensuring a wide coverage of different pos-

Table 8: Features with highest Information Gain

#	feature
1	cosine similarity between sentence and most similar sentence in API documentation
2	average cosine similarity between sentence and all sentences in API documentation
3	question user acceptance rate
4	answer score
5	answer age
6	answer time difference to question
7	question score
8	question favorites
9	question user reputation
10	question views
11	number of nouns followed by verb in present tense, third person singular in sentence
12	question age
13	sentence starts with noun followed by verb in present tense, third person singular
14	number of tokens in sentence
15	position of API element in sentence (or 0)
16	number of occurrences of API element in sentence
17	answer score
18	answer size
19	number of nouns in sentence
20	sentence starts with noun
21	number of characters that are code
22	number of occurrences of the verb "be" in sentence

sible algorithms. We used WEKA's default setting for each classifier except for the k-nearest neighbour classifier which we instantiated with values of one, five, and ten for k.[7] Because of the large number of features generated by our treatment of *string* features, we calculated the information gain of each feature and used attribute selection [14, 40] to reduce the data set in order to improve classifier performance.

To calculate the precision and coverage of each classifier, we applied what we call "ten-type cross validation", i.e., we trained the algorithms on sentences belonging to nine API types in our development set, and we tested them on the tenth type. Each type was used once as the test type. We used five different settings for filtering out features based on low information gain: no filtering as well as filtering at a 0.01, 0.02, 0.03, and 0.04 threshold, respectively.

Only four cases achieved a precision of above 0.5: the random forest classifier without attribute selection achieved a precision of 0.60 with a coverage of 0.7 (i.e., the classifier produced at least one sentence for seven out of the ten types in our development set). The support vector machine classifier showed a similar performance in terms of precision at a value of 0.64 when combined with an information gain filter for features at thresholds of 0.02 and 0.03, with the same coverage. Finally, the random forest classifier with an attribute selection threshold of 0.03 achieved perfect precision, but only covered a single type in the development set. Balancing precision and coverage using the harmonic mean, we conclude that the most promising performance was shown by the support vector machine classifier at information gain thresholds of 0.02 and 0.03. We use the 0.02 setting for the remainder of the paper.

[7]Tuning machine learning parameters is part of our future work.

Table 9: Sentences in the Comparative Study

	Lex	MMR	KP	SISE	(unique)
Applet	1	1	1	1	(3/4)
AppletCont.	1	1	1	2	(4/5)
Event	1	1	6	0	(8/8)
Component	1	1	3	7	(11/12)
PropertyCh.	1	1	3	1	(6/6)
File	1	1	4	4	(10/10)
PropertyD.	1	1	1	0	(2/3)
Object	1	1	12	4	(18/18)
InputStream	1	1	7	4	(13/13)
URL	1	1	4	2	(8/8)
Thread	1	1	11	5	(18/18)
AccessContr.	1	1	7	2	(10/11)
Socket	1	1	2	2	(6/6)
Connection	1	1	3	1	(5/6)
Principal	1	1	4	1	(7/7)
List	1	1	11	15	(28/28)
ResultSet	1	1	4	0	(5/6)
JComponent	1	1	1	0	(3/3)
Map	1	1	21	22	(43/45)
JFrame	1	1	3	0	(5/5)
sum	20	20	109	73	(213/222)

Table 8 shows the features in this setting. The features are diverse, ranging from a sentence's similarity to the API documentation and part-of-speech tags to attributes of the answer, question, and the user asking the question.

In answering our first research question regarding the ability of different approaches to identify meaningful insight sentences, we conclude that only the supervised approach was able to identify insight sentences with reasonable precision and coverage.

6. COMPARATIVE STUDY

To investigate our second research question, i.e., whether practitioners find these insight sentences useful, we conducted a comparative study. We selected the most commonly used type and the third most commonly used type from each of the ten most commonly used Java packages, as indicated by Apatite [12] (cf. Table 3). The motivation for this stratified sampling was to cover a wide range of types while avoiding ones that are rarely used.

We then used all four approaches (LexRank, MMR, patterns, and SISE) to extract insight sentences for these 20 API types. For LexRank and MMR, we used the configuration that achieved the highest precision on the development set (i.e., we considered only the sentence with the highest score). Table 9 shows the number of sentences that each approach extracted. LexRank and MMR extracted exactly one sentence per API type, while patterns extracted between 1 and 21 sentences per API type and SISE extracted between 0 and 22 sentences per API type.[8] As the last column shows, the overlap between sentences extracted by different approaches was very low: 213 of the 222 sentences selected by all approaches for all API types were unique. Seven of the nine overlaps occurred between LexRank and MMR, one occurred between LexRank and patterns, and one involved patterns and SISE. To keep the number of sentences man-

[8]This variation is explained by the length of the Stack Overflow threads linked to each API type. Devising algorithms for ranking sentences is part of our future work.

Table 10: Participants in the Study

job title	exp.	area
Technical Consultant	4 years	automation, mobile
Software Engineer	4 years	web
Research Assistant	10+ years	embedded, system
Postdoc Researcher	10+ years	web, systems
Software Engineer	5 years	data engineering
Student	2 years	web
Android Developer	3 years	mobile
Software Developer	3 years	web, systems

Table 11: Comparative Study Results

	meaningf., added inf.	meaningf., no added inf.	more context	no sense
Lex	9	5	15	11
	22.5%	12.5%	37.5%	27.5%
MMR	13	4	17	6
	32.5%	10.0%	42.5%	15.0%
KP	34	21	46	23
	27.4%	16.9%	37.1%	18.6%
SISE	38	18	12	12
	47.5%	22.5%	15.0%	15.0%

ageable for a study, we randomly selected up to four sentences per approach and per API type. This resulted in at most ten sentences per API type (one for LexRank, one for MMR, at most four for patterns, and at most four for SISE). Duplicate sentences (selected by more than one approach) were only shown to participants once.

We recruited eight participants from GitHub, randomly selecting from the 68,949 GitHub users who had made at least one contribution in the previous twelve months, used Java in at least one of their projects, and had published their email address. We randomly selected email addresses in batches of ten. It took 40 emails to recruit these eight participants (response rate 20%). The study was conducted using Google Forms and there were no time constraints. To minimize bias, we did not explain the research goal to participants. Each participant was shown sentences belonging to five API types, leading to a maximum of 50 sentences per participant. All sentences were rated by exactly two participants. We asked each participant whether developing software was part of their job, about their job title, for how many years they had been developing software, and what their area of software development was. Table 10 shows the answers. All participants indicated that developing software was part of their job.

For each pair of an API type and a sentence, we asked the participants to choose one of the following options:

- The sentence is meaningful and adds useful information not found in the API documentation.
- The sentence is meaningful, but does not add useful information to the API documentation.
- The sentence requires more context to understand.
- The sentence does not make sense to me.

These options were motivated by Binkley et al.'s [3] observation that summaries should be judged based on their usefulness rather than their quality alone.

Table 11 shows the results of the comparative study. SISE resulted in most ratings indicating a meaningful sentence, both in absolute numbers and relatively. In total, 70% of the sentences identified by SISE were considered meaningful (the first two answer options), compared to 44% for patterns, 43% for MMR, and 35% for LexRank. When comparing SISE to each of the other approaches, the difference between meaningful sentences and not meaningful sentences (the last two answer options) is statistically significant (Pearson's chi square, $p < 0.005$). In addition, SISE resulted in the highest number of sentences which were considered to add useful information not found in the API documentation.

Table 12: Inter-rater Agreement

	(1)	(2)	(3)	(4)
(1) meaningf., added inf.	27	12	16	8
(2) meaningf., no added inf.	–	8	12	6
(3) req. more context	–	–	17	22
(4) no sense	–	–	–	6

Table 12 shows the inter-rater agreement. Out of a total of 134 pairs of ratings, 58 (43%) were in perfect agreement. The highest number of disagreements (22) was related to sentences that either require more context to understand or make no sense.

In answering our second research question on the usefulness of insight sentences as perceived by practitioners, we conclude that our participants found more sentences which contained useful information in the output of SISE compared to the output of other approaches.

7. DISCUSSION

This section discusses the implications of our work, in particular related to user interface design for insight sentence presentation and to the role that meta data on Stack Overflow can play for the extraction of insight sentences. In addition, we discuss the inter-rater agreement from our comparative study in more detail and review the threats to validity.

7.1 Sentence Meta Data

This work shows that the large amount of meta data on Stack Overflow can be used for the extraction of insight sentences. Compared to state-of-the-art summarization techniques or pattern-based techniques which do not take any meta data into account, SISE achieved higher precision and usefulness. Out of the 22 features with the highest information gain used in the classifier, half of them (and eight out of the first ten) represent Stack Overflow meta data, such as the number of views on a question or the score of an answer (cf. Table 8). Interestingly, the features with the highest information gain also suggest that the meta data of the person asking the question is possibly more important than the meta data of the person authoring the answer. For example, the feature with the third highest information gain is the acceptance rate of the person asking the question. We hypothesize that the acceptance rate of a user reflects the kinds of questions that such a user would ask, and that insight sentences are more likely to come from answers to questions that ask about basic information instead of specific use cases. Future work will have to be conducted to investigate this hypothesis.

Another interesting finding is that the two features that represent the similarity of a potential insight sentence to the corresponding API documentation were the features with

the highest information gain. This finding suggests that there is an advantage to interpreting sentences on Stack Overflow in the context of other documentation sources. Our current hypotheses regarding the positive correlation between sentence similarity and meaningful insights is that a "somewhat similar" sentence combines content from the API documentation with new information, while less similar sentences contain information completely unrelated to an API type.

7.2 User Interface

The results of our evaluation suggest that the context of sentences will play an important role when complementing API documentation with sentences from Stack Overflow. In fact, only 15% of the ratings for sentences extracted by SISE indicated that the sentence did not make sense. Another 15% of the ratings indicated that more context was required for the sentence to be understandable. Since this context (e.g., surrounding code snippets, the complete answer, or the corresponding question) is available on Stack Overflow, it would be possible to display it along with an insight sentence. For example, each insight sentence could be accompanied by an expandable widget which shows the entire thread on Stack Overflow from which the insight sentence originated. In addition, user input similar to the one we gathered as part of our comparative study could be used to continuously improve the extraction of insight sentences.

Figure 2 shows the current version of our interface for SISE. In the top left corner of the API documentation, a widget is added that shows up to five insight sentences for the API type. Upon selection of one sentence, the sentence is expanded to show the surrounding paragraph from the original source, along with a link to the corresponding Stack Overflow thread.

7.3 Inter-rater Agreement

Since the inter-rater agreement in our comparative study was relatively low, we analyzed the disagreements in more detail.

For the twelve cases where both raters agreed that the sentence was meaningful but disagreed as to whether it added useful information not contained in the API documentation, we manually verified whether that information could indeed be found in the API documentation. In nine out of the twelve cases, the information in the sentence was available in the API documentation. However, the insight sentence often summarized information in a more succinct way than the API documentation did, e.g., *"List is an ordered sequence of elements whereas Set is a distinct list of elements which is unordered"*, which was extracted by SISE for Java's `List`. In some contexts, such sentences could still be useful since they provide a more succinct version of content that is available in the API documentation.

There were 28 cases where one rater indicated that more context was required to understand the sentence while the other rater indicated that the sentence was meaningful. In many of these cases, the background of the users seems to determine whether they understand a sentence or not. We found a similar situation in our previous work [36] when we asked developers to rate the meaningfulness of task descriptions that we had automatically extracted from their software documentation. In those cases, we argue that displaying such sentences does little harm if some users do not understand them while other users find them useful. An example for such a sentence from our data set is *"Yes you should close a ResultSet"*, which was extracted by the pattern-based approach for Java's `ResultSet`. Arguably, this sentence should be accompanied by a question to be more meaningful, yet the message from the sentence is understandable without the specific question.

In 14 cases, one participant indicated that a sentence did not make sense while the other participant found it meaningful. A manual inspection of those 14 cases suggests that in most cases, the problem was missing context. An example is *"I don't know what's your problem, but if you have some problems to run this code, you can try to close connection and open other to make the second query"*, a sentence that was extracted by LexRank and MMR and is related to Java's `ResultSet`. While the sentence does require more context about the questioner's problem to be understandable, it might be helpful without such context if a user is troubleshooting a connection issue related to a `ResultSet`. As mentioned before, we are addressing the context issue with a user interface that shows more context when requested.

7.4 Threats to Validity

A threat to the validity of our results is the manual construction of the development set since we did not attempt to validate the annotated data. However, it would have been practically impossible for us to annotate sentences in a way that would favour specific approaches. When the development set was constructed, we were not aware that SISE would be based on machine learning, thus our development set was not biased towards certain features.

The size of the development set is another limitation since the sentences were related to only ten Java API types. However, for each API type, we considered ten different questions on Stack Overflow, and for each question, we considered up to ten answers. In total, the 1,574 sentences originated from 309 different answers. In addition, our finding that SISE outperformed state-of-the-art summarization techniques and a pattern-based approach was confirmed in a comparative study with sentences related to another twenty Java API types.

The agreement about insight sentences between our study participants was relatively low. It is natural for software developers with different backgrounds and different experience to disagree on what information is useful. Despite the disagreements, the comparative study clearly showed that SISE produced the most meaningful and useful insight sentences.

The evaluation of the usefulness of the insight sentences was based on subjective assessment from the study participants. Although all sentences were judged by two participants to eliminate the threat of individual bias, it is nevertheless possible that the responses may be affected by collective bias. There was no mechanism to ensure participants read the API documentation before rating sentences, and we acknowledge this threat to validity.

We cannot claim that SISE generalizes beyond Java. However, none of the features used in SISE are specific to Java, and we are optimistic that we can achieve similar results for other programming languages in future work.

We also cannot make claims regarding generalizability beyond Stack Overflow. However, with more than 17 million answers, Stack Overflow is a big enough data source to war-

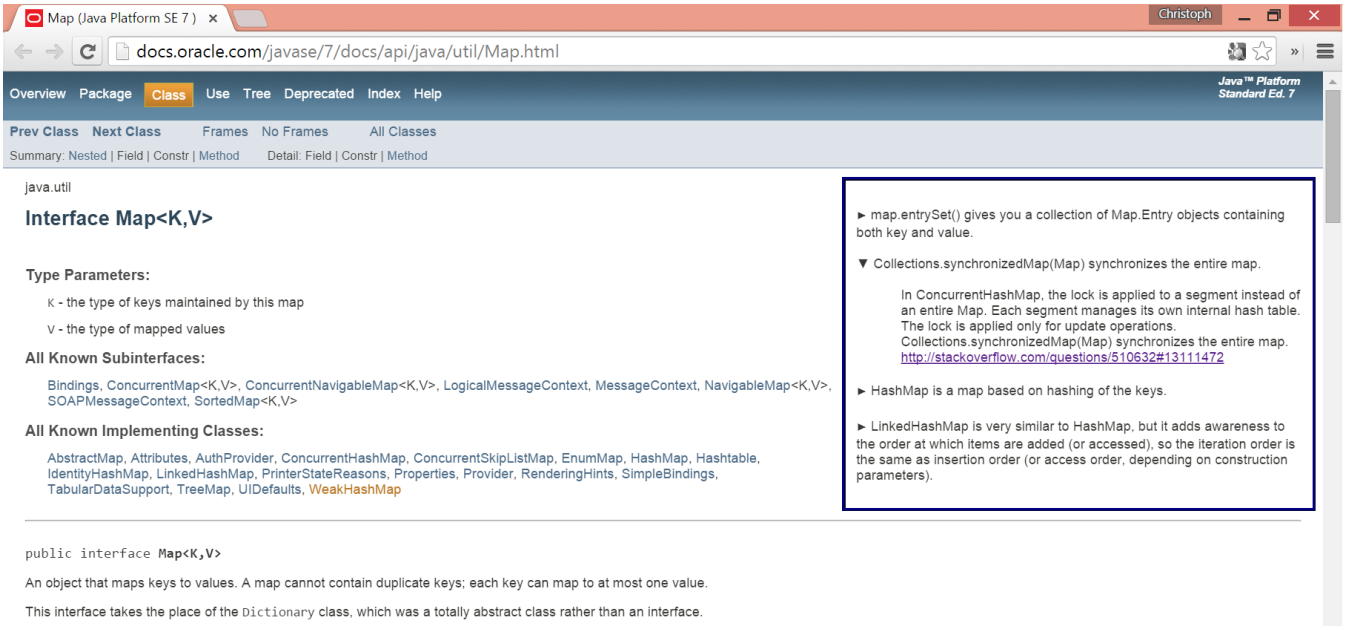

Figure 2: Screenshot of Java's `Map` documentation, with insight sentences produced by SISE

rant specialized tools to utilize its data. SISE will only work if a topic is discussed on Stack Overflow. Since all insight sentences used in this paper were obtained from sets of ten Stack Overflow threads associated with an API type, we would expect comparable results for any API type with at least ten threads on Stack Overflow. As we found in our previous work [21], 77% of the types of the Java API are discussed on Stack Overflow (Android: 87%) – thus, we do not expect library popularity to be a major limitation.

8. RELATED WORK

Work related to our approach for insight sentence extraction can be divided into work on harnessing Stack Overflow data and work on improving API documentation.

8.1 Harnessing Stack Overflow data

Seahawk by Bacchelli et al. [2] is an Eclipse plug-in that integrates Stack Overflow content into an integrated development environment (IDE). Seahawk automatically formulates queries from the current context in the IDE and presents a ranked and interactive list of results. The tool lets users identify individual discussion pieces and import code samples through drag & drop. In addition, users can link Stack Overflow discussions and source code. An evaluation of Seahawk showed that the tool can produce surprising insights that aid a developer in program comprehension and software development [25]. A related tool called Prompter was later proposed by Ponzanelli et al. [26]. Given the IDE context, Prompter automatically retrieves pertinent discussions from Stack Overflow, evaluates their relevance and notifies developers about the available help if a given threshold is surpassed.

Other approaches have focused on harnessing Stack Overflow data for explaining stack traces in the IDE. Cordeiro et al. [8] developed a tool that integrates the recommendation of Q&A web resources related to stack traces into Eclipse. Their preliminary evaluation showed that their approach outperformed a simple keyword-based approach. In a similar line of work, Rahman et al. [29] developed a context-aware IDE-based meta search engine for recommendations about programming errors and exceptions. The Stack Overflow API is one of the search APIs used in their work, and their approach captures the context in a similar fashion to the work by Cordeiro et al. In an evaluation, the authors found that the inclusion of different types of contextual information associated with an exception can enhance the accuracy of recommendations.

Arguably the work that is most similar to ours is AutoComment, the automatic comment generation approach introduced by Wong et al. [39], since it also harnesses the natural language text available on Stack Overflow. AutoComment extracts code-descriptions mappings, which are code segments together with their descriptions, from Stack Overflow, and leverages this information to automatically generate descriptive comments for similar code segments in open-source projects. The authors applied AutoComment to Java and Android projects, and they were able to automatically generate 102 comments for 23 projects. In a user study, the majority of participants found the generated comments to be accurate, adequate, concise, and useful in helping them understand the code. Our work differs from that by Wong et al. in that we focus on single sentences from Stack Overflow that are relevant to an API type instead of a code snippet.

Related to our solution for linking Stack Overflow threads to API types is the work by Rigby and Robillard [30]. Their traceability recovery approach discovers essential code elements in informal documentation such as Stack Overflow. Our linking approach for linking Stack Overflow threads to API types works the other way around. We start from an API type, and then use the Stack Overflow API as well as a number of regular expressions to find threads that are related to that API type.

In terms of using machine learning to discover content on Stack Overflow, there are some common themes between SISE and the work of de Souza et al. [10]. They developed an improved search engine for content on Stack Overflow which recommends question-and-answer pairs (as opposed to entire Q&A threads) based on a query. The ranking criteria used by their approach consists of the textual similarity of the question-and-answer pairs to the query and the quality of these pairs. In addition, their search focuses on "how-to" threads. In an evaluation of their work, the authors found that their approach was able to recommend at least one useful question-and-answer pair for most queries, many of which included a reproducible code snippet. In comparison, the goal of SISE is the extraction of insight sentences from Stack Overflow that add useful information to API documentation. Our catalogue of machine learning features is also more extensive and includes features that bridge the gap between different documentation formats.

8.2 Improving API documentation

Several researchers have contributed efforts for the improvement of API documentation. Stylos et al. [33] introduced Jadeite, which displays commonly used classes more prominently and automatically identifies the most common ways to construct an instance of any given class. eMoose by Dekel and Herbsleb [11] improves API documentation by decorating method invocations whose targets have associated usage directives, such as rules or caveats, of which authors of invoking code must be aware. In a similar effort, Pandita et al. [20] proposed to infer formal specifications from natural language text.

More closely related to SISE is the proposal for integrating crowdsourced FAQs into API documentation by Chen and Zhang [6]. They propose to connect API documentation and informal documentation through the capture of developers' Web browsing behaviour. In contrast, we connect different forms of documentation through heuristics that match Stack Overflow threads to API types, and instead of FAQs, SISE produces insight sentences.

Other work has focused on detecting and preventing API documentation errors. Zhong and Su [41] introduced DOCREF, an approach that combines natural language processing techniques and code analysis to detect and report inconsistencies in documentation. The authors successfully used DOCREF to detect more than one thousand documentation errors. Dagenais and Robillard [9] introduced AdDoc, a technique that automatically discovers documentation patterns, i.e., coherent sets of code elements that are documented together, and that reports violations of these patterns as the code and the documentation evolve.

Previous work has successfully identified natural language text that is potentially important for a programmer using a given API type. Chhetri and Robillard [7] categorized text fragments in API documentation based on whether they contain information that is indispensable, valuable, or neither, using word patterns. When we applied their patterns to content on Stack Overflow, we were not able to repeat their positive results in terms of precision and usefulness (see Section 4.3). Petrosyan et al. [22] proposed an approach to discover tutorial sections that explain a given API type. They classified fragmented tutorial sections using supervised text classification based on linguistic and structural features and they were able to achieve high precision and recall on different tutorials. Their work differs from ours in that we use Stack Overflow's meta data for SISE. In addition, unlike Petrosyan et al., we focus on the extraction of single sentences instead of entire documentation fragments.

Several researchers have focused on augmenting API documentation with code examples. For example, Kim et al. [17] proposed a recommendation system that returns API documents embedded with code example summaries mined from the Web. Their evaluation results showed that the approach provides code examples with high precision and boosts programmer productivity. In a similar effort, Subramanian et al. [34] introduced Baker, an iterative, deductive method for linking source code examples to API documentation. In contrast to these tools, SISE links natural language sentences from Stack Overflow to API documentation.

9. CONCLUSION AND FUTURE WORK

While the rise of social media and Q&A sites such as Stack Overflow has resulted in a plethora of information for software developers available in many different formats on the Web, it can be difficult to determine where a particular piece of information is stored. In an effort to bring documentation from different sources together, we presented an evaluation of different techniques for extracting insight sentences from Stack Overflow. We define insight sentences as those sentences on Stack Overflow that are related to a particular API type and that provide insight not contained in the API documentation of the type.

In a comparative study with eight software developers to evaluate the meaningfulness and usefulness of the insight sentences, we found that our supervised approach (SISE) resulted in the highest number of sentences which were considered to add useful information not found in the API documentation. We conclude that considering the meta data available on Stack Overflow along with natural language characteristics can improve existing approaches when applied to Stack Overflow data.

We believe that we are the first to investigate augmenting natural language software documentation from one source with that from another source. Our work is a first step towards a vision of presenting users with combined documentation from various sources rather than users having to look through different sources to find a piece of information. We plan to extend this work beyond the Java API and we plan to experiment with more features that capture the grammatical structure of sentences on Stack Overflow. Determining whether a sentence is meaningful on its own is non-trivial, and while our evaluation showed that a supervised approach can detect such sentences based on part-of-speech tags with a higher precision than summarization or pattern-based approaches, we expect that the precision can further be improved with a deeper understanding of each sentence and its dependencies on other sentences or code snippets. In addition, we intend on applying the idea of insight sentence extraction to other textual artifacts produced by software developers, such as bug reports, commit messages, or code comments.

Acknowledgements

We thank the study participants. This research was funded by NECSIS.

10. REFERENCES

[1] D. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[2] A. Bacchelli, L. Ponzanelli, and M. Lanza. Harnessing Stack Overflow for the IDE. In *Proc. of the 3rd Int'l. Workshop on Recommendation Systems for Software Engineering*, pages 26–30, 2012.

[3] D. Binkley, D. Lawrie, E. Hill, J. Burge, I. Harris, R. Hebig, O. Keszocze, K. Reed, and J. Slankas. Task-driven software summarization. In *Proc. of the 29th Int'l. Conf. on Software Maintenance*, pages 432–435, 2013.

[4] F. Boudin, M. El-Bèze, and J.-M. Torres-Moreno. A scalable MMR approach to sentence scoring for multi-document update summarization. In *Proc. of the 22nd Int'l. Conf. on Computational Linguistics (Companion volume: Posters and Demonstrations)*, pages 23–26, 2008.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] C. Chen and K. Zhang. Who asked what: Integrating crowdsourced FAQs into API documentation. In *Companion Proc. of the 36th Int'l. Conf. on Software Engineering*, pages 456–459, 2014.

[7] Y. B. Chhetri and M. P. Robillard. Recommending reference API documentation. *Empirical Software Engineering*, 20(6):1558–1586, 2014.

[8] J. Cordeiro, B. Antunes, and P. Gomes. Context-based recommendation to support problem solving in software development. In *Proc. of the 3rd Int'l. Workshop on Recommendation Systems for Software Engineering*, pages 85–89, 2012.

[9] B. Dagenais and M. P. Robillard. Using traceability links to recommend adaptive changes for documentation evolution. *IEEE Trans. on Software Engineering*, 40(11):1126–1146, 2014.

[10] L. B. L. de Souza, E. C. Campos, and M. d. A. Maia. Ranking crowd knowledge to assist software development. In *Proc. of the 22nd Int'l. Conf. on Program Comprehension*, pages 72–82, 2014.

[11] U. Dekel and J. D. Herbsleb. Improving API documentation usability with knowledge pushing. In *Proc. of the 31st Int'l. Conf. on Software Engineering*, pages 320–330, 2009.

[12] D. S. Eisenberg, J. Stylos, and B. A. Myers. Apatite: A new interface for exploring APIs. In *Proc. of the Conf. on Human Factors in Computing Systems*, pages 1331–1334, 2010.

[13] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal on Artificial Intelligence Research*, 22(1):457–479, 2004.

[14] G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal on Machine Learning Research*, 3:1289–1305, 2003.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[16] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *11th Conf. on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[17] J. Kim, S. Lee, S.-W. Hwang, and S. Kim. Enriching documents with examples: A corpus mining approach. *ACM Trans. on Information Systems*, 31(1):1:1–1:27, 2013.

[18] W. Maalej and M. P. Robillard. Patterns of knowledge in API reference documentation. *IEEE Trans. on Software Engineering*, 39(9):1264–1282, 2013.

[19] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[20] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring method specifications from natural language API descriptions. In *Proc. of the 34th Int'l. Conf. on Software Engineering*, pages 815–825, 2012.

[21] C. Parnin and C. Treude. Measuring API documentation on the web. In *Proc. of the 2nd Int'l. Workshop on Web 2.0 for Software Engineering*, pages 25–30, 2011.

[22] G. Petrosyan, M. P. Robillard, and R. de Mori. Discovering information explaining API types using text classification. In *Proc. of the 37th Int'l. Conf. on Software Engineering*, pages 869–879, 2015.

[23] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. 1998.

[24] L. Plaza and J. C. de Albornoz. Evaluating the use of different positional strategies for sentence selection in biomedical literature summarization. *BMC Bioinformatics*, 14(71):1–11, 2013.

[25] L. Ponzanelli, A. Bacchelli, and M. Lanza. Leveraging crowd knowledge for software comprehension and development. In *Proc. of the 17th European Conf. on Software Maintenance and Reengineering*, pages 57–66, 2013.

[26] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In *Proc. of the 11th Working Conf. on Mining Software Repositories*, pages 102–111, 2014.

[27] M. F. Porter. Snowball: A language for stemming algorithms, 2001. http://snowball.tartarus.org/texts/introduction.html.

[28] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, 1993.

[29] M. M. Rahman, S. Yeasmin, and C. K. Roy. Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions. In *Proc. of the Software Evolution Week: Conf. on Software Maintenance, Reengineering and Reverse Engineering*, pages 194–203, 2014.

[30] P. C. Rigby and M. P. Robillard. Discovering essential code elements in informal documentation. In *Proc. of the 35th Int'l. Conf. on Software Engineering*, pages 832–841, 2013.

[31] M. P. Robillard. What makes APIs hard to learn? Answers from developers. *IEEE Software*, 26(6):27–34, 2009.

[32] M. P. Robillard and R. Deline. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[33] J. Stylos, B. A. Myers, and Z. Yang. Jadeite: Improving API documentation using usage information. In *Extended Abstracts on Human Factors in Computing Systems*, pages 4429–4434, 2009.

[34] S. Subramanian, L. Inozemtseva, and R. Holmes. Live API documentation. In *Proc. of the 36th Int'l. Conf. on Software Engineering*, pages 643–652, 2014.

[35] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? (NIER track). In *Proc. of the 33rd Int'l. Conf. on Software Engineering*, pages 804–807, 2011.

[36] C. Treude, M. P. Robillard, and B. Dagenais. Extracting development tasks to navigate software documentation. *IEEE Trans. on Software Engineering*, 41(6):565–581, 2015.

[37] C. Treude, M. Sicard, M. Klocke, and M. P. Robillard. TaskNav: Task-based navigation of software documentation. In *Proc. of the 37th Int'l. Conf. on Software Engineering*, pages 649–652, 2015.

[38] C. Treude and M.-A. Storey. Effective communication of software development knowledge through community portals. In *Proc. of the 8th joint meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering*, pages 91–101, 2011.

[39] E. Wong, J. Yang, and L. Tan. Autocomment: Mining question and answer sites for automatic comment generation. In *Proc. of the 28th Int'l. Conf. on Automated Software Engineering*, pages 562–567, 2013.

[40] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of the 14th Int'l. Conf. on Machine Learning*, pages 412–420, 1997.

[41] H. Zhong and Z. Su. Detecting API documentation errors. In *Proc. of the Int'l. Conf. on Object Oriented Programming Systems Languages and Applications*, pages 803–816, 2013.