Disambiguating mentions of API methods in stack overflow via type scoping

Kien LUONG

Ferdian THUNG

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

Citation

LUONG, Kien; THUNG, Ferdian; and LO, David. Disambiguating mentions of API methods in stack overflow via type scoping. (2021). *37th IEEE International Conference on Software Maintenance and Evolution (ICSME 2021)*. 679-683.

Available at: https://ink.library.smu.edu.sg/sis_research/6874

Disambiguating Mentions of API Methods in Stack Overflow via Type Scoping

Kien Luong, Ferdian Thung, and David Lo
School of Computing and Information Systems, Singapore Management University
{kiengialuong, ferdianthung, davidlo}@smu.edu.sg

***Abstract*—Stack Overflow is one of the most popular venues for developers to find answers to their API-related questions. However, API mentions in informal text content of Stack Overflow are often ambiguous and thus it could be difficult to find the APIs and learn their usages. Disambiguating these API mentions is not trivial, as an API mention can match with names of APIs from different libraries or even the same one. In this paper, we propose an approach called DATYS to disambiguate API mentions in informal text content of Stack Overflow using *type scoping*. With type scoping, we consider API methods whose type (i.e. class or interface) appear in more parts (i.e., scopes) of a Stack Overflow thread as more likely to be the API method that the mention refers to. We have evaluated our approach on a dataset of 807 API mentions from 380 threads containing discussions of API methods from four popular third-party Java libraries. Our experiment shows that our approach beats the state-of-the-art by 42.86% in terms of F_1-score.**

***Index Terms*—API linking, Mining, Disambiguation**

I. Introduction

In an effort to understand APIs[1], developers often discuss and mention them in natural language texts of online forums such as Stack Overflow. Mentions of API methods sharing the same name would be ambiguous to developers or automated tools that are looking for a specific API. Therefore, API disambiguation important when finding APIs in Stack Overflow. It supports several downstream tasks such as API recommendation [1], [2] and API mining [3], [4] since correctly associating the ambiguous API mentions to their actual APIs is necessary to properly index and link the APIs to their related information in various data sources (Stack Overflow, Javadoc, etc.).

To disambiguate API mentions in informal text content, two steps are involved: (1) API mention extraction; and (2) API mention disambiguation. API mention extraction aims to identify common words that refer to APIs. On the other hand, API mention disambiguation aims to link API mentions to APIs that they refer to. Some works [5]–[9] deal with both steps. Some [10], [11] only deal with API mention extraction. Our work deals with the second step, which is API mention disambiguation. It can be used together with any API mention extraction approaches proposed in all of the above work.

In this paper, we propose an approach named DATYS to disambiguate API method mentions via *type scoping* in informal text content of Stack Overflow. It exploits *type scoping* to resolve ambiguous mentions of Java API methods in informal text context of Stack Overflow, in the scenario where the mentions have been identified. DATYS first extracts API method candidates from input Java libraries. Given a Stack Overflow thread with identified API mentions, it scores API method candidates based on how often their types (i.e., classes or interfaces) appear in different parts (i.e., scopes) of the thread. We consider three scopes: (1) *mention scope*, which covers the mention itself; (2) *text scope*, which covers the textual content of the thread including the mentions; and (3) *code scope*, which covers the code snippets in the thread. An API candidate score is higher if its type appears in more scopes. For each API mention in the thread, API candidates are ranked based on their scores. DATYS takes the top-1 API candidate with a non-zero score as the API that the mention actually refers to. If the top-1 API candidate has a zero score, it considers the mention refers to an unknown API.

APIReal is the state-of-the-art approach in resolving ambiguous mentions of Python APIs in the textual content of Stack Overflow thread [9]. It leverages mention-mention similarity, mention-entry similarity, and scope filter. The mention-mention similarity checks if there are similar mentions in the thread that are unambiguous. The mention-entry similarity computes the similarity between the thread and the API documentation. The scope filter shrinks the number of API candidates by checking the existence of their types in the code snippets and their library names in the thread's tags or title. Different than APIReal, our approach does not use a similarity metric. Also, instead of filtering, our approach leverages scopes for scoring API candidates. APIReal's scopes are only tags, URLs, title, and code snippets. Ours are more exhaustive. It covers all the different parts of the thread (i.e., title, tags, URL, text, and code snippets), allowing us to capture the types that APIReal would have missed. This exhaustiveness, paired with our scoring mechanism, can rank the types based on its relevancy within the scopes of the thread.

We evaluate DATYS and compare it with APIReal on a dataset collected from Stack Overflow. Because there is no available dataset for disambiguating Java API mentions in textual content of Stack Overflow (i.e., APIReal dataset is on Python while datasets used in [5]–[8] are not from Stack Overflow), we collect our own dataset. Our dataset contains API mentions in informal texts of Stack Overflow threads discussing APIs from 4 Java libraries. The dataset is labeled by participants with at least 2 years of experience in Java. Hard to disambiguate cases that are discussed carefully by both participants until a consensus is reached. Additionally, we

[1]In this paper, we use the term API and API method interchangeably.

perform an ablation study where we remove one of the three scopes for scoring API candidates and observe its impact on *F_1-score*. We observe that removing any scope reduces the *F_1-score*, demonstrating the utility of considering different scopes.

The contributions of this paper are:

- We propose DATYS, an approach that uses type scoping to disambiguate mentions of Java API methods in informal text content of Stack Overflow.
- We build a ground truth dataset containing 807 Java API mentions from 380 threads in Stack Overflow and evaluate our approach on the ground truth dataset and achieves an *F_1-score* of 0.760, which beats the state-of-the-art baseline by 42.86%.

II. Approach

The overview of DATYS is presented in Figure 1. DATYS takes as inputs a Stack Overflow thread, a recognized API mention (i.e., an already identified API mention), and Java libraries. It extracts API method candidates from the Java libraries and filters out the candidates whose simple name does not match the mention. Code snippets from the thread are then passed into the *Possible Types Extraction* step to extract a list of possible types (i.e. class or interface) that the method referred by the mention may have. The recognized API mention, the list of possible types, the content of the thread, and the API method candidates are then fed into the *Type Scoping*. *Type Scoping* scores an API method candidate based on the occurrences of its type (i.e., class or interface) in various scopes of the thread. An API method candidate with the highest score is more likely to be the method that the mention refers to. *Type Scoping* returns a ranked list of API method candidates based on their scores. The API method candidate with the highest non-zero score is chosen as the API method referred to by the mention. If more than one candidate holds the highest non-zero score, we randomly pick the candidate as the API method referred to by the mention. If the top-1 API candidate has a zero score, the mention is not associated with any API method in the candidates.

We describe in detail how DATYS extracts possible types from code snippets in Section II-A and how it scores API method candidates with type scoping in Section II-B.

A. Possible Types Extraction

In order to extract possible types of API mention from code snippets, we analyze the code elements by considering the two cases below. Since the code snippet is often unparsable, we capture the code elements using regular expressions.

Static method import. We consider the class of the imported static method as a possible type of API mention having the same name as the imported static method. For example, for imported static method `org.mockito.Mockito.mock`, `org.mockito.Mockito` is a possible type for API mention *mock*. For methods imported by wildcard imports, we first check if the class of the imported methods is the same as the class of any API method candidate. If it does, we consider the class as a possible type of API mention matching the name of any of the imported static methods.

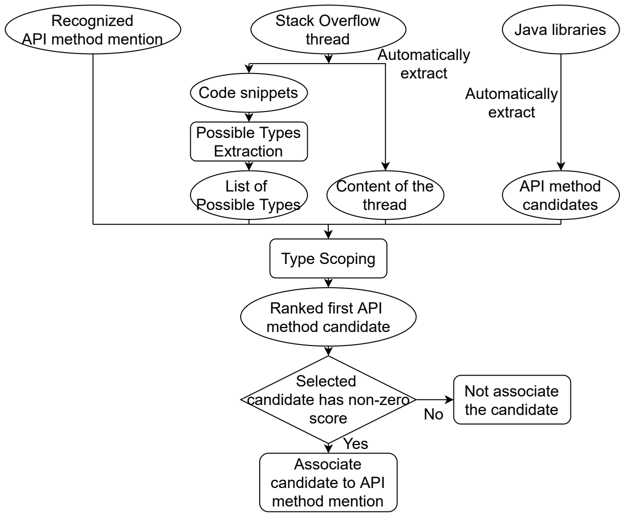

Recognized API method mention
Stack Overflow thread
Java libraries
Automatically extract
Automatically extract
Code snippets
Possible Types Extraction
List of Possible Types
Content of the thread
API method candidates
Type Scoping
Ranked first API method candidate
Selected candidate has non-zero score
No
Not associate the candidate
Yes
Associate candidate to API method mention

Fig. 1. Overview of DATYS

Method invocation. Given a method invocation in the form of $\langle prefix \rangle.\langle method \rangle$, we check if *prefix* term (i.e., we call it *method caller*) starts with a capital letter, which indicates that the caller is a class. We then check the class import statements. If the fully qualified name of the imported class ends with the class name, we consider it as the fully qualified name of the class. At this point, we consider the fully qualified name as a possible type of API mention that matches the name of the invoked method. For example, an invocation `Mockito.mock(...)` means that `Mockito` is a class name. If we have a class import statement such as `import org.mockito.Mockito;`, in which the fully qualified name of the imported class name ends with `Mockito`, it is the fully qualified name of `Mockito`.

We also check if the method caller is declared in any of the variable declaration statements. If so, we resolve the caller to its type as defined in the statement. Next, we would try to resolve the fully qualified name of the caller type by checking the class/interface import statements. If the fully qualified name of the imported class/interface ends with the caller type, we consider it as the fully qualified name of the caller type. At this point, we consider either the caller type or its fully qualified name (if resolved through class/interface import statements) as a possible type of API mention that matches the name of the invoked method.

B. Scoring with Type Scoping

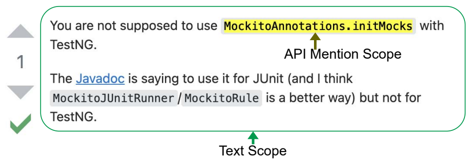
Type scoping scores an API method candidate based on three types of scopes. Each scope contributes to an API method candidate $APIMethodCandidate$ a score of either 0 or 1. Therefore, the maximum score of an API method candidate is 3. Algorithm 1 shows the scoring algorithm. $APIMention$, $PTypeList$, $APIMethodCandidate$ stand for the API mention, the list of possible types extracted

Algorithm 1 Scoring an API Candidate with Type Scoping

```
Input: $ApiMention, PTypesList, APIMethodCandidate,$
       $ThreadContent$
Output: $CandScore$
 1: $CandScore$ = 0
 2: $CandType = getType(APIMethodCandidate)$
 3: if $hasPrefix(ApiMention)$ then
 4:     $Prefix = getPrefix(ApiMention)$
 5:     if $endsWith(Prefix, CandType)$ then
 6:         $CandScore = CandScore + 1$
 7:     end if
 8: end if
 9: $Tokens = tokenize(ThreadContent)$
10: if $CandType$ in $Tokens$ then
11:     $CandScore = CandScore + 1$
12: end if
13: for $PType$ in $PTypesList$ do
14:     if $isSameType(PType, CandType)$ then
15:         $CandScore = CandScore + 1$
16:     end if
17: end for
18: return $CandScore$
```

from code snippets of the thread (see Section II-A), the API method candidates the mention may refer to, respectively. $ThreadContent$ is the thread's content including the text, the title, and the tags. The score of the API candidate $CandScore$ is first initialized to zero (Line 1) and the type of the API candidate $CandType$ is extracted (Line 2). $CandScore$ is increased by one if $CandType$ appears in these scopes:

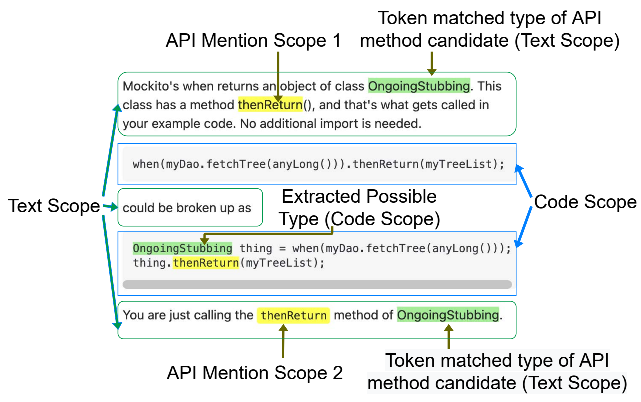
1) *Mention Scope (Lines 3-8).* The algorithm increases $CandScore$ when $CandType$ appears within the scope of the mention itself. Given API mention in the form of $\langle prefix \rangle.\langle method \rangle$, *Prefix* helps associate the API mention with the right API among the candidates. *Prefix* can be either a variable or a type. Having *Prefix* which matches $CandType$ would increase the likelihood of correctly linking the mention. Hence, by utilizing *Prefix*, the algorithm increases $CandScore$ by one when *Prefix* and $CandType$ are matched (i.e., $CandType$ ends with *Prefix*). Figure 2 shows an example of API mention `MockitoAnnotations.initMocks`. The Mention Scope finds a prefix in the scope of the API mention, i.e., it gets the prefix `MockitoAnnotations` from the mention. The scores of API method candidates having the type `MockitoAnnotations` are then increased by one.
2) *Text Scope (Lines 9-12).* The algorithm increases $CandScore$ when $CandType$ appears within the scope of textual content of the thread, including the mention. It first tokenizes the content of $ThreadContent$ by splitting it into alphanumerical tokens $Tokens$. If $CandType$ appears in $Tokens$, the algorithm increases $CandScore$ by one. Consider an example in Figure 3, in the Text Scope, due to the occurrence of the term `OngoingStubbing` in the thread, the scores of the API method candidates having the type `OngoingStubbing` are increased by one.
3) *Code Scope (Lines 13-17).* The algorithm increases $CandScore$ when $CandType$ appears within the scope of the code snippet in the thread. It iterates the list of possible types for an API mention $PTypeList$, which is


1

You are not supposed to use `MockitoAnnotations.initMocks` with TestNG.

The Javadoc is saying to use it for JUnit (and I think `MockitoJUnitRunner` / `MockitoRule` is a better way) but not for TestNG.

Fig. 2. Example 1 from thread *32065666* in Stack Overflow


Mockito's when returns an object of class OngoingStubbing. This class has a method thenReturn(), and that's what gets called in your example code. No additional import is needed.

```
when(myDao.fetchTree(anyLong())).thenReturn(myTreeList);
```

could be broken up as

```
OngoingStubbing thing = when(myDao.fetchTree(anyLong()));
thing.thenReturn(myTreeList);
```

You are just calling the `thenReturn` method of OngoingStubbing.

API Mention Scope 1
Token matched type of API method candidate (Text Scope)
Text Scope
Extracted Possible Type (Code Scope)
Code Scope
API Mention Scope 2
Token matched type of API method candidate (Text Scope)

Fig. 3. Example 2 from thread *26026018* in Stack Overflow

extracted from code snippet as described in Section II-A. If $CandType$ matches one of the possible types $PType$ in $PTypeList$, the algorithm increases $CandScore$ by one. Consider the example in Figure 3, from the code snippets, we can perform *Possible Type Extraction* and get the possible type `OngoingStubbing` for API mention *thenReturn*. The scores of API method candidates with type `OngoingStubbing` are then increased by one.

III. Dataset and Experimental Settings

A. API candidates from top-4 popular Java libraries

Top-4 most used libraries from Maven Repository[2] are selected as the source of our API method candidates. These libraries are Guava, Mockito, AssertJ, and Fastjson. Fully qualified names of all public methods in the libraries are extracted. There is a total of 38 134 API methods extracted where Guava, Mockito, AssertJ and Fastjson have 10 485, 4 033, 21 785, and 1 831 unique API methods, respectively.

B. Selected Stack Overflow threads

As there is no available ground truth dataset of Java API method mentions in Stack Overflow, we collected the dataset ourselves. The Stack Overflow data used in this study is a subset of Stack Exchange Data Dump published on 09/2020. The Stack Overflow posts tagged with at least a library in the top-4 popular libraries are extracted. Also, answer posts referring to the same question post are grouped into a thread. Next, threads having accepted answer are kept while the ones without accepted answer (including those having no answer) are discarded. We also remove the non-accepted answers from the threads. This is because we assume that threads with accepted answer are of higher quality since they

[2]https://mvnrepository.com/

are deemed useful by the users posting the questions. Non-accepted answers may also be wrong.

To exclude irrelevant threads (i.e., threads not containing mentions of API methods in the 4 libraries), we select threads for each API method in our API method candidates (see Section III-A) by searching if the method name appears in the thread. Furthermore, a ranked list of relevant threads is established for each method in order to select threads that are more likely to contain mentions of our API method candidates. The threads are first fit into a vector space model [12]. For each API method candidate, we use the fully qualified name of the method as a query and select top-5 relevant threads based on their score. The reason of selecting the top-5 relevant threads is to filter out threads not containing any mention of API methods and limit the number of threads to label. Note that a thread might appear in the top-5 relevant threads of more than one query. Therefore, we take only of the unique threads that appear in the top-5 relevant threads of any method. Moreover, since manual labeling is costly, in this preliminary evaluation, we sample 20% of the unique threads. Therefore, we further remove threads discussing non-Java code (e.g., Scala, Groovy) since they can import Java libraries and end up with 380 threads.

For the 380 selected threads, we search the textual content of the threads for terms that match the name of API method and manually label whether the terms are API mentions. For terms that are mentions, we further label them with the names of API methods from the API method candidates. In case the mentions do not refer to any method in the candidates, we label the mention to refer to an unknown method. There are a total of 807 labelled method mentions. The mentions are labelled by four participants with at least 2 years of experience in Java. They discuss cases which are difficult to disambiguate with each other until they reach an agreement.

C. Evaluation Metrics

We use *precision*, *recall*, and F_1*-score* to measure the performance of our techniques. In detail, each API method mention is either linked to a unique API method from API method candidates or not linked to any candidates. We define ground truth as a candidate in the candidate list which the mention actually refers to. The predicted API method for a mention is a True Candidate Prediction (TCP) if it is the same as the ground truth. A False Candidate Prediction (FCP) case occurs whenever the predicted API method refers to an API method that is different from the ground truth. A False Non-linking Prediction (FNN) occurs whenever the predicted API method is an unknown API and the ground truth is a method in the candidates. We then calculate $Precision = \frac{TCP}{TCP+FCP}$, $Recall = \frac{TCP}{TCP+FNN}$, and $F_1\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall}$.

D. Baseline

APIReal [9] works with Python APIs in the textual content of Stack Overflow. Thus, we have to adapt it when using it as a baseline. APIReal uses type mentions to disambiguate method mentions. Since our dataset does not have type mentions, we find terms matching the type names and manually label whether they are type mentions. Moreover, APIReal has a knowledge base containing various information about APIs. We fill the knowledge base with different forms of method names of our API method candidates: simple names and fully qualified names.

TABLE I
EFFECTIVENESS OF OUR APPROACH

	Precision	***Recall***	***F_1-score***
DATYS	0.696	0.837	0.760
APIReal	0.367	0.967	0.532

TABLE II
CONTRIBUTION OF EACH TYPE SCOPE

	Precision	***Recall***	***F_1-score***
all Type Scopes	0.696	0.837	0.760
w/o Code Scope	0.695	0.822	0.752
w/o Mention Scope	0.682	0.834	0.75
w/o Text Scope	0.815	0.418	0.552

IV. RESULTS

Performance. Table I shows the effectiveness of DATYS compared to APIReal. Our approach and APIReal achieve F_1*-scores* of 0.760 and 0.532, respectively. When compared to APIReal, DATYS also has a higher precision, which indicates that it is better in predicting a correct API method for a given mention. While APIReal has a higher recall, it often assigns incorrect API methods to mentions. Its low precision score highlights this problem.

Ablation study. To evaluate the contribution of each type scope, we remove one of the three scopes of our API candidate scoring algorithm and observe its impact on the effectiveness of DATYS. As shown in Table II, removing each of the scopes reduces the effectiveness in terms of F_1*-score*, which indicates that each scope contributes to the overall effectiveness of our approach. Based on the reductions of F_1*-score*, the ranked list of most important scopes is text scope (most important), mention scope, and code scope (least important).

Qualitative study. We investigate cases where DATYS fails to disambiguate API mentions. The first failed case (i.e., thread 25850491 in Table III) occurs when the API mention *assertThat* refers to a method that is defined in the code snippet of the thread. Since it is not an API method from a third-party library, it should not be linked to any API method candidates. However, because of the appearance of token `Assertions` in the textual content of the thread, our approach links the API mention to a wrong type. To avoid this issue, the algorithm can learn the semantic context surrounding the API mention to determine if the method the mention refers to is defined in the thread or not. If the method is defined in the thread, the algorithm should not link it to any API method candidates.

For the second failed case (i.e., thread 30668168 in Table III), due to the lack of occurrences of the corresponding type in the scopes, our approach does not link the API mention to any method. However, we observe that some methods often co-occur together. We can exploit the co-occurrence of mentions referring to these methods.

TABLE III
FAILED CASES (API METHOD MENTIONS ARE BOLDED)

Thread ID	*Thread content*	*Ground truth*	*Prediction*
25850491	[...] **assertThat** method to extend AssertJ. [...] The compiler doesn't know whether assertThat(Iterable) ([...]Assertions.java#L194) or my method should be used.	Not linked	org. assertj. core.api. Assertions. assertThat
30668168	[...] verify(x,times(1)). doSomething(any(B.class)); //fails. **verify times(1)** fails...	org. mockito. Mockito. verify	Not linked

V. RELATED WORK

Prior works [5]–[9], [13]–[15] worked on API disambiguation. We can divide them into two groups: informal text disambiguation [5]–[9] and code snippet disambiguation [13]–[15]. The first group disambiguates API mentions in textual contents. The second group strives to find the fully qualified name of code elements available in code snippets. Our method, DATYS, belongs to the first group.

For disambiguating API mentions in informal text, there are studies that leverage IR techniques (Vector Space Model, Latent Semantic Indexing) [5]–[7] and heuristics [8]. Bacchelli et al [7] developed Miler, which utilized string matchings and IR techniques in linking emails to source code entities in software systems such as *classes* in object-oriented systems and *function* in procedural language systems. Dagenais and Robillard [8] applied filtering heuristics to link Java APIs mentioned in support channels (e.g., mailing list, forums), documents, and code snippets. Recently, Ye *et al.* [9] work on API mention recognition and disambiguation in the textual content of Stack Overflow thread. They develop three modules: mention-mention similarity, mention-entry similarity, and scope filter. We provided a comparison between our approach and APIReal in Section I.

To disambiguate API mentions (i.e., code elements) in code snippets, Baker [14] iteratively performs a deductive linking technique to identify the fully qualified name of the code elements. COSTER [13] and STATTYPE [15] capture and learn the tokens around the code element and associate them to the identified element with similar tokens that they have learnt. These studies aim at disambiguating API mentions in code snippets while we focus on disambiguating API mentions in natural language text of Stack Overflow.

VI. THREATS TO VALIDITY

We may incorrectly adapt the baseline (i.e., APIReal). APIReal uses type mentions to disambiguate method mentions.To provide type mentions, we manually label terms mentioning type and feed them into APIReal. This can be considered as APIReal accurately identified all type mentions, which should benefit its performance rather than harm it. Regarding the knowledge base of APIReal, we adapted it by importing the equivalent information about the API method candidates in Java (whenever possible). Therefore, we believe that the impact of our adaptation should be minimal. We may also wrongly label API mentions. In some cases, it is difficult to determine which API a mention is referring to. To minimize labeling errors, our participants discussed the hard cases with each other.

VII. CONCLUSION AND FUTURE WORK

We present DATYS, an approach to disambiguate API mentions in informal text of Stack Overflow. We exploit the type occurrence of Java API method candidate in the various scopes of StackOverflow thread (code, text, and mention scope). Our evaluation shows that our approach is better than prior work on disambiguating API mentions in textual content of Stack Overflow. In the future, we plan to improve our approach by utilizing machine learning algorithm to learn the semantic context of text around the mention to better disambiguate it. We also plan to experiment with more threads and libraries. Also, we would like to make our approach language agnostic so it can be directly applied to different programming languages without any modifications.

Replication Package. The source code for DATYS is available at https://github.com/Kienlgk/DATYS.

Acknowledgment. This work was supported by The Ministry of Education Academic Research Fund Tier 2 (MOE2019-T2-1-193).

REFERENCES

[1] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "Api method recommendation without worrying about the task-api knowledge gap," in *ASE*. IEEE, 2018.

[2] M. M. Rahman, C. K. Roy, and D. Lo, "Rack: Automatic api recommendation using crowdsourced knowledge," in *SANER 2016*, vol. 1. IEEE, 2016.

[3] G. Uddin, F. Khomh, and C. K. Roy, "Mining api usage scenarios from stack overflow," *Information and Software Technology*, vol. 122, 2020.

[4] G. Uddin and F. Khomh, "Automatic mining of opinions expressed about apis in stack overflow," *TSE*, 2019.

[5] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *TSE*, vol. 28, no. 10, 2002.

[6] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *ICSE*. IEEE, 2003.

[7] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *ICSE*, 2010.

[8] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *ICSE*. IEEE, 2012.

[9] D. Ye, L. Bao, Z. Xing, and S.-W. Lin, "Apireal: an api recognition and linking approach for online developer forums," *Empirical Software Engineering*, vol. 23, no. 6, 2018.

[10] S. Ma, Z. Xing, C. Chen, C. Chen, L. Qu, and G. Li, "Easy-to-deploy api extraction by multi-level feature embedding and transfer learning," *TSE*, 2019.

[11] D. Ye, Z. Xing, C. Y. Foo, J. Li, and N. Kapre, "Learning to extract api mentions from informal natural language discussions," in *ICSME*. IEEE, 2016.

[12] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.

[13] C. K. Saifullah, M. Asaduzzaman, and C. K. Roy, "Learning from examples to find fully qualified names of api elements in code snippets," in *ASE*. IEEE, 2019.

[14] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *ICSE*, 2014.

[15] H. Phan, H. A. Nguyen, N. M. Tran, L. H. Truong, A. T. Nguyen, and T. N. Nguyen, "Statistical learning of api fully qualified names in code snippets of online forums," in *ICSE*. IEEE, 2018.