API-Related Developer Information Needs in Stack Overflow

Mingwei Liu, Xin Peng, Andrian Marcus, Shuangshuang Xing, Christoph Treude, and Chengyuan Zhao

Abstract—Stack Overflow (SO) provides informal documentation for APIs in response to questions that express API related developer needs. Navigating the information available on SO and getting information related to a particular API and need is challenging due to the vast amount of questions and answers and the tag-driven structure of SO. In this paper we focus on identifying and classifying fine-grained developer needs expressed in sentences of API-related SO questions, as well as the specific information types used to express such needs, and the different roles APIs play in these questions and their answers. We derive a taxonomy, complementing existing ones, through an empirical study of 266 SO posts. We then develop and evaluate an approach for the automated identification of the fine-grained developer needs in SO threads, which takes a thread as input and outputs the corresponding developer needs, the types of information expressing them, and the roles of API elements relevant to the needs. To show a practical application of our taxonomy, we introduce and evaluate an approach for the automated retrieval of SO questions, based on these developer needs.

Index Terms—Developer Information Need, API, Stack Overflow.

1 Introduction

Software reuse through Application Programming Interfaces (APIs) is an integral part of software development [1], but learning how to effectively use APIs can be difficult [2], often impeded by the inadequacies of API documentation. While such documentation might capture an API's structure, it tends to lack information on concepts, purposes, and usage scenarios [3]. As an alternative form of documentation, the question-and-answer forum Stack Overflow (SO) can fill this gap to some extent by providing informal "how-to" documentation in response to specific needs [4].

However, identifying and extracting information available on Stack Overflow, which is relevant to APIs in the context of a task, is challenging due to the vast amount of questions and answers. At the time of writing, Stack Overflow hosts more than 19 million questions and close to 29 million answers. Even for a particular API library, the amount of information can be overwhelming, *e.g.*, there are currently about 25,000 questions tagged with "junit". Stack Overflow only offers minimal organization of this information via its tagging mechanism which allows users to associate up to five tags and a title with each question. This tagging mechanism is predominantly used to indicate the technologies relevant to a question [5]. As a result, all discussions relevant to an API library are often grouped under a single tag, not doing justice to developers who have task-specific information needs [6]. The title of SO posts summarizes them better than the tags, however it often does not cover all pertinent information for the question. For example, the following title "*How to split a string in Java*"[1] only reflects one of the two goals of the questioner: "*I have a string [..] that* ***I want to split*** *into two strings. [...]* ***I also want to check*** *if the string has '-' in it.*" If developers only look at the title, they might ignore discussions related to their needs.

People asking questions on SO have various backgrounds (*e.g.,* students, professional developers, *etc.*), yet we will refer to them simply as *developers*. With regard to the goals of the questioners, we refer to them as *developer information needs* or simply *developer needs*.

Developer information needs have been the subject of many studies (*e.g.,* [7], [8]) and several researchers analyzed and categorized the SO questions on some of these needs ([4], [5], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]). Beyer *et al.* [9] categorized SO questions in seven high-level categories (*i.e., API Usage*, *Conceptual*, *Discrepancy*, *Errors*, *Review*, *API Change* and *Learning*), which subsume categories defined in previous work.

We conjecture that the categories defined by Beyer *et al.* [9] are too coarse-grained to reason about specific developer needs and require further refinement. For example, the "*API Usage*" category is defined as "*A how type of questions asks for ways to achieve a goal*". As Beyer *et al.* , we found that many SO questions (51.9% in our data - Section 2.2.1) refer to more than one developer information need, showing that many questions are complex. For example, the following SO question[2] includes information (highlighted in bold) expressing at least three different information needs: (1) implementing a functionality; (2) error handling; (3) comparing APIs.

> *I was trying to* ***load a file in a webapp****, and I was* ***getting a FileNotFound exception when I used FileInputStream****. However, using the same path, I was able to* ***load the file when I did getResourceAsStream(). What is the difference between the two methods****, and why does one work while the other doesn't?*

- *M. Liu, X. Peng, S. Xing and C. Zhao are with the School of Computer Science and Shanghai Key Laboratory of Data Science, Fudan University, and the Shanghai Institute of Intelligent Electronics & Systems, China.*
- *X. Peng is the corresponding author (pengxin@fudan.edu.cn).*
- *A. Marcus is with the University of Texas at Dallas.*
- *C. Treude is with the University of Melbourne.*

1. https://stackoverflow.com/questions/3481828

2. https://stackoverflow.com/questions/2308188

Although it seems that only the last API comparison question is the question directly raised by the questioner, other developers with similar goals to the first one and second one can also benefit from this discussion. At the same time, the answers to a question often involve multiple APIs, each having a different role in the answer, an aspect not captured in existing taxonomies.

We argue that we need a more fine-grained categorization of developer information needs and expressed in SO question, which would help users to understand and find easier complex questions and answers.

In this paper we focus on identifying and classifying fine-grained developer needs expressed in sentences of *API-related SO questions*, as well as the specific information types used to express such needs, and the different roles APIs play in these questions and their answers. We derive a fine-grained taxonomy, complementing existing ones, through an empirical study of 266 SO posts (Section 2). We then develop and evaluate an approach for the automated identification of these fine-grained developer needs in SO threads (a thread includes a question post with the corresponding answer posts), which takes a thread as input and outputs the corresponding developer needs, the types of information used to express them, and the roles of the pertinent API elements from the thread (Section 3.3). The evaluation indicates that our approach can accurately (83.6% precision and 85.4% recall, in average) identify developer needs, relevant information types, and API roles, in SO threads.

We conjecture that the fine-grained API-related developer needs and the API roles capture essential features of the SO threads, which help in the retrieval of SO questions, especially multiple developer information needs are involved. We developed a retrieval approach leveraging the above-mentioned identification tool (Section 5). For evaluation, we compared the retrieval performance of our approach with a state-of-the-art retrieval approach, AnswerBot [19] The results show that our approach outperforms AnswerBot on Top@1 (0.625 vs 0.484), Top@5 (0.828 vs 0.797), and Top@10 (0.859 vs 0.828) accuracy and MRR (0.698 vs 0.617). We further conducted a user study asking participants to complete programming tasks with the help of our approach or with AnswerBot. The results show that, using our approach, participants could complete tasks faster (378s vs 518s).

In summary, the contributions of this paper are:

- A fine-grained taxonomy of developer needs in SO posts, together with the information needed to express them, and the roles of the APIs in addressing the needs. The taxonomy is accompanied by an annotated data set used to derive it.
- An approach that automatically identifies fine-grained developer needs and relevant information in SO posts.
- An approach for the retrieval of API-related SO questions, based on the developer information needs.

2 Developer Needs in SO Posts

We conducted an empirical study for understanding what type of developer needs are expressed in API related SO questions and how. Fig. 1 shows the relationships between the main concepts used in the paper. An SO [*thread*] could include a [*question*] (with a title and the body) and multiple [*answers*]. The question may express multiple [*developer information needs*]. Each [*developer need*] is an instance of a [*developer need type*] and it is described by [*describing sentences*] from the [*question*]. The [*describing sentences*] contain the [*relevant information*] for expressing the [*developer need*]. Each [*relevant information*] is an instance of a [*relevant information type*]. Multiple [*APIs*] may be mentioned in the [*question*] or [*answers*], and each API plays a specific [*API role*] in describing the [*developer need*] or its solution.

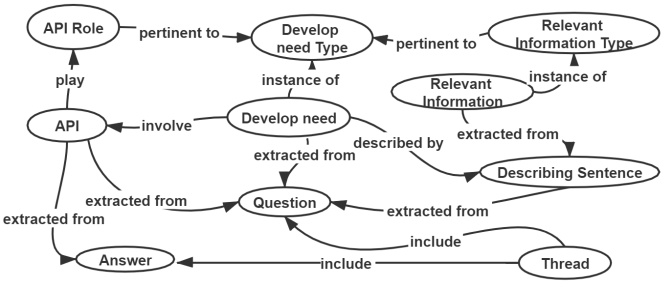

Fig. 1. Conceptual Schema of Our Taxonomy

We focus on answering the following research questions:
RQ1: *What type of developer needs are present in SO questions?*
RQ2: *What type of information is used to describe the developer needs?*
RQ3: *What roles do APIs play related to the developer needs?*

2.1 Study Design

While we considered the seven categories defined by Beyer *et al.* as a starting point to our study, we performed open coding on a set of SO threads with the goal of refining and/or redefining them, as needed. Since our focus is on API-related questions only, we limited the scope of the "*Learning*" category to "*API Usage Learning*" only.

Qualitative Analysis Method. Based on the thematic analysis framework proposed by Braun and Clarke [20], we conducted a qualitative analysis by performing open coding on API-related threads. We treat the developer need, relevant information, and API role as themes and the analysis was conducted collaboratively by following steps similar to Robillard *et al.* [21].

1. Data Collection and Familiarisation. We first collected API-related threads from SO. Then all annotators read those collected threads in order to become familiar with them, paying specific attention to patterns that occur.

2. Coding on Data. Annotators analyzed the API-related threads and coded those threads according to the coding protocol we designed. As a result, we obtained a list of codes for developer need types, relevant information types, and API role types.

3. Generating, Reviewing and Defining Codes/Themes. We first grouped codes into themes, *i.e.,* developer need, relevant information, and API role. Then we discussed the distribution of codes across threads and the relationships between themes (*i.e.,* Fig. 1) and codes (*i.e.,* Table 2 and Table 4). We reviewed once again all API-related threads, focusing on checking whether the definitions of codes/themes are appropriate, and whether the relationships between

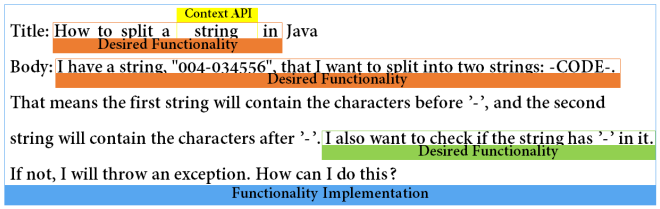

Fig. 2. An Example of Coding of a SO Question

themes and codes are correct. We repeated this process until the results were stable.

Next, we elaborate on our data collection and coding protocol.

Data Collection. We selected threads related to JDK and Android APIs for this empirical study. We chose JDK and Android APIs because they are popular [22] and we are familiar with them. To obtain the data, we first selected threads tagged with "java" and an accepted answer from the SO data dump [23] and removed those that did not contain any qualified name or aliases of APIs from JDK 1.8[3] or Android API 27[4] in the title, question body, or accepted answer. The aliases of APIs were derived from the qualified name (*e.g.*, `StringBuffer` is one derived alias for `java.lang.StringBuffer`). To further ensure the quality of threads, we ranked the threads by the number of question votes and retained the top 500 voted threads. Then, we manually removed threads that were not about APIs. The manual removal was conducted by two students independently (one PhD and one MS student, each with more than five years Java and Android experience). One of the authors was assigned to resolve any conflicts, although the agreement between students was near perfect (*i.e.*, Cohen's Kappa coefficient [24] of 0.92). After this step, we obtained 266 threads about APIs. Unlike our data set, Beyer *et al.* [9] used 500 randomly sampled posts with the tag "android" which were not necessarily API related.

Coding Protocol for Developer Needs and Relevant Information. To answer **RQ1** and **RQ2**, we analyzed the questions from the 266 threads. We preprocessed the questions from HTML format to clean text using BeautifulSoup [25]. Long code snippets that were wrapped by `<pre><code></code></pre>` were replaced with a placeholder "`-CODE-`" during parsing. Where necessary, a "." was added after `-CODE-` to ensure that the following sentence splitting is correct. For each question, we split the text in the title and question body into sentences and combined them together because we need to annotate questions at sentence-level. As shown in Fig. 2, for the question[5] "*How to split a string in Java*" is the first sentence and "*I have a string, "004-034556", that I want to split into two strings: `-CODE-`.*" is the second sentence.

Fig. 2 shows an example of how we coded the questions. First, three of the authors (two PhD and one MS student, each with more than five years Java and Android experience) coded the questions into *developer need types* through discussion and consensus. If the question expresses a developer need type but it does not meet the current definition of any type, we modify the definition of the existing type or create a new developer need type after discussion. If developer need types are changed, we re-annotate all questions again. As a result, one question could be classified into several developer need types at the same time, *e.g.*, the SO question shown in the box in the introduction. We classified the question in Fig. 2 into developer need type "*Functionality Implementation*" (Table 1).

Then if a question is an instance of a developer need type, we will check each sentence in the question for identifying the specific information that describes this developer need, which we call *relevant information*. The relevant information instances were classified and refined through card sorting. One sentence could be classified into several relevant information types of the same developer need types or different developer need types at the same time. If the sentence does not provide important information for any developer need type, we will annotate it as "*useless*" (*e.g.*, "*Thanks.*"). For example, the three colored sentences in the Fig. 2 are all annotated as relevant information "*Desired Functionality*" for "*Functionality Implementation*" (see Table 2) and the remaining sentences are annotated as " *useless*".

In summary, the coding for developer need types and relevant information was iterative. When we found that a question/sentence cannot be coded with an existing type, we created a new type or modified the definition of an existing type. Then, we re-annotated all the questions again, to account for the new type or the modified definition. We finished the coding process once we achieved saturation, *i.e.*, once no new developer need types and relevant information types were found.

To further verify that our coding for developer needs and relevant information is correct and complete, and to minimize any bias, we asked two MS students (not involved in previous annotation) familiar with Java and Android to annotate a subset of the 266 questions with our coding by following our coding protocol with the codes we derived. First we sampled five questions for each developer need type based on our annotation for questions, and we got 34 questions, after removing duplicate questions. The annotation was performed by both students independently. For each question, they annotated it with some developer need types, and they annotated each sentence with relevant information types corresponding to the annotated developer need types or *"useless"*. If none of the existing codes was suitable, they annotated the question or sentence as "*New Code*". The Cohen's Kappa agreement coefficients for developer need types are all above 0.6 (*i.e.*, substantial agreement), with a minimum of 0.62, a maximum of 1.00, and an average of 0.89. The Cohen's Kappa agreement coefficients for relevant information types are also all above 0.6, with a minimum of 0.66, a maximum of 1.00, and an average of 0.85. All developer need type codes and relevant information type codes were used in this round of annotation and no new codes were reported. We found the disagreements often occurred when the students overlooked some parts of very long questions.

To identify developer need instances we need to group the sentences that describe the same developer need in-

3. https://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html
4. https://developer.android.com/reference/packages
5. https://stackoverflow.com/questions/3481828

TABLE 1
Definitions and Examples of Developer Need Types (QC: Question Count, DNC: Developer Need Count)

Developer Need Type	Definition	SO ID	Example Summary	QC	DNC
Functionality Implementation	The developer wants to implement specific functionality	1816673	How do I check if a file exists in Java?	182	187
Non-Functional Improvement	The developer wants to improve the existing implementation for some non-functional requirements, *e.g.,* code quality	1306727	Is there a neater way for getting the length of an int than this method?	32	32
Functional Improvement	The developer wants to fix an implementation whose expected performance is not consistent with the actual performance without an obvious error message	869033	I want to copy the dum to dumtwo and change dum without affecting the dumtwo. But the code above is not doing that.	32	32
Error Handling	The developer wants to fix an implementation with an obvious error message	1393486	java.lang.OutOfMemoryError: GC overhead limit	28	28
Rationale Analysis	The developer wants to understand the internal implementation and design of an API.	7421004	Is FileInputStream using buffers already?	39	54
API Comparison	The developer wants to compare multiple APIs.	355089	Difference between StringBuilder and StringBuffer	27	27
Alternative Solution	The developer wants to find an alternative of an existing solution.	54516417	Backward alternative solution for ChronoUnit.Days.between()	9	9
API Usage Learning	The developer wants to learn how/when/where to use an API	2793150	How to use java.net.URLConnection to fire and handle HTTP requests	87	87

stances together. Note that the sentences in a question providing relevant information for the same developer need type are not always describing the same developer need instance. First, we filtered out sentences annotated as "*useless*" and grouped the remaining sentences. A sentence group contains several sentences from the same question and is annotated with relevant information types of the same developer need type. A sentence may appear in different groups because it may be annotated with relevant information types of different developer need types at the same time. For each sentence, two MS students were asked to group them independently, based on the developer need type they describe. The original question was provided as the context when grouping. If they disagreed on the grouping, one of the authors was assigned to resolve conflicts. The Cohen's Kappa agreement coefficient for sentence grouping is 0.95, (*i.e.,* nearly perfect agreement). The coding for developer needs and the relevant information are providing guidance for identifying developer need instances, which makes it easier for participants to reach consensus. As a result, we obtained 456 developer need instances described by one or more sentences in the questions from the 266 SO threads. For example, the three colored sentences in Fig. 2 are grouped into two developer need instances, colored with orange and green, respectively.

Coding Protocol for API Roles. To answer **RQ3**, we asked two MS students (same as above) to identify the APIs involved in answering the 456 developer needs and the roles of each involved API. Three of the authors analyzed 20 threads and defined an initial set of codes for different API roles. For example, "*suggested API*" is the role for the APIs suggested by answerers to satisfy the developer need.

The students were shown one developer need at a time with its original question and accepted answer of the question as context. The APIs could be identified from the question or its accepted answer. To make the annotation easier, we ignored the other answers of the question. An API could only be coded with one API role for the current developer need, but could be coded with different API roles for other developer needs. For example, "string" (*i.e.,* `java.lang.String`) is annotated as "*Context API*" (Table 3) for both developer needs in Fig. 2.

The students coded independently. If their role annotations for the same API in a developer need were different, one of the authors was assigned to resolve the conflict. During coding, if an API could not be coded with any existing API role, we changed the definition of existing API roles or created new API roles, after discussion and agreement. If the categories were changed, the students re-coded the APIs again. The two coders identified 2,049 APIs in total (same APIs for different developer needs were treated as different APIs) and among them 68.6% APIs (1,406 of 2,049) were identified by both coders. We checked the APIs that were not identified by both coders. The main cause was that the questions or the answers in those cases were very long, with large code snippets, and the coders missed some APIs. The Cohen's Kappa agreement coefficient for annotating API roles is 0.88 (*i.e.,* near perfect agreement). After resolving the conflicts, we obtained 1,932 APIs (note that the same API playing a different role is considered distinct) involved in the questions and answers of the 456 developer needs. Note that 117 APIs are removed after resolving the conflicts. Mainly, those removed APIs are APIs that were misidentified by participants, *e.g.,* java.lang.StringBufer, which should be java.lang.StringBuffer.

2.2 Results

2.2.1 RQ1 (Developer Need Types)

Table 1 shows the definitions and examples of the eight developer information need types we identified, with the numbers of questions where these developer need types appear. The last column indicates the number of distinct developer needs that belong to that type.

Among the 456 specific developer needs we identified in the 266 questions, *functionality implementation* is the most frequent developer need type (41.0%). This implies that developers often ask for help to implement a specific functionality. Among the 266 questions, 128 questions contained

TABLE 2
Definitions and Examples of Relevant Information Types (SC: Sentence Count, DNC: Developer Need Count)

Developer Need Type	Relevant Information	Definition	Example	SC	DNC	Essen.
Functionality Implementation	Desired Functionality	Describes the functionality that developer wants to implement	How do I **check if a file exists in Java**?	379	187	Yes
Non-Functional Improvement	Implemented Functionality	Describes the functionality that has been implemented	Is there a neater way for **getting the length of an int** than this method: -code-	62	32	Yes
	Suboptimal Implementation	Describes the existing suboptimal implementation	Is there a neater way for getting the length of an int than **this method: -CODE-**	40	32	Yes
	Improvement	Describes the improvement developer wants to make	Is there **a neater way** for getting the length of an int than this method: -code-	37	27	No
Functional Improvement	Expected Result	Describes the result expected	I want to **copy the dum to dumtwo and change dum without affecting the dumtwo**.	66	32	Yes
	Actual Result	Describes the actual results obtained	However, I only **get the file name, not the file content.**	48	32	Yes
	Insufficient Implementation	Describes the insufficient implementation	I've tried doing this: **-CODE-** .	46	32	Yes
Error Handling	Error Type	Describes the error type of implementation	**java.lang.OutOfMemoryError: GC overhead limit exceeded**	49	28	Yes
	Error Occasion	Describes where the error occurred	I get exception **when using Thread.sleep(x) or wait()**	41	28	No
	Erroneous Implementation	Describes the implementation with error	This is my test case: **Parent.java -CODE-**.	18	15	No
Rationale Analysis	Rationale Question	Describes the question about the internal implementation or design of an API	**Is FileInputStream using buffers already?**	81	54	Yes
API Comparison	Comparison Subject	Describes the subjects being compared	Difference between **StringBuffer and StringBuilder**?	47	27	Yes
	Comparison Scenario	Describes the scenario for the comparison	What is the difference between using the Runnable and Callable interfaces **when designing a concurrent thread in Java**, why would you choose one over the other?	17	15	No
Alternative Solution	Current Solution	Describes the current solution that needs an alternative solution	Backward alternative solution for **ChronoUnit.Days.between()**	11	9	Yes
	Alternative Description	Describes the functionality and constraints that the alternative solution has	I need an alternative solution for ChronoUnit.Days.between() that **works for Android version prior to API 26**.	12	6	No
API Usage Learning	Used Subject	Describes which API the developer want to know the usage of	How to use **java.net.URLConnection** to fire and handle HTTP requests	149	87	Yes
	Usage Scenario	Describes the scenario to use the API	How to use java.net.URLConnection to **fire and handle HTTP requests**	140	80	No

1 developer need; 93 questions contained 2 developer needs; 38 questions contained 3 developer needs; 7 questions contained 4 developer needs. Among the questions with more than one developer need (51.9%), 92.8% (128 of 138) questions address developer needs of different types. That is, 48.1% (128 of 266) questions may be classified into several developer need types at the same time.

As shown in Table 1, the value of QC is the same as DNC's in six out of eight developer need types (except for functionality implementation and rationale analysis). This shows that different developer need types have different characteristics. For some developer need types (*e.g.,* functionality implementation and rationale analysis), the questioners may mention multiple developer needs of the same type but belonging to different instances in one question, *e.g.,* the question shown in Fig. 2. For other types, such as alternative solution, the questioner is unlikely to inquire about alternative solutions for two different implementations at the same time.

Some developer need types seem to overlap to some extent, *e.g.,* API comparison and API usage learning. However, each developer need type we define has a different way of describing questions from other types (see Section 2.2.2 and Section 2.2.3). These eight developer need types refine the taxonomy proposed by Beyer *et al.* [9]. We discuss in more detail at the end of this subsection how these two frameworks complement each other.

Among the eight developer need types, the top three (*i.e.,* functionality implementation, non-functional improvement, and functional improvement) are more general than the rest and they can apply to other non-API related questions. Some developer need types have commonalities. Non-functional improvement, functional improvement, error handling and alternative solution are for developers who

TABLE 3
Definitions of API Roles

API Role	Definition	Count
Context API	APIs as the context to describe the developer need, *e.g.,* APIs being compared, API in need of alternative	522
Suggested API	APIs suggested by the answerer to solve the developer need	1,188
Currently Used API	APIs used in current implementation of questioner.	138
Error API	APIs maybe the reason why this error happened.	53
Exception Type API	The exception type usually is Exception class or Error class.	31

already have a solution and want to improve/fix it. API comparison, API usage learning and rationale analysis are for developers who want to learn specific APIs.

In conclusion, questions on Stack Overflow could be quite complex and contain multiple developer needs from different developer need types. This is also consistent with our intuition. We need a way to more accurately analyze the developer needs in the questions.

2.2.2 RQ2 (Relevant Information Types)

Table 2 shows the definitions and examples of the 17 types of relevant information used for describing developer needs. We report how many sentences (SC) and how many developer needs (DNC) are described with the corresponding relevant information. We observed that not all relevant information types are used to describe all instances of a developer need type. For example, an API usage learning need described by the sentence "how to use FileInputStream" only provides relevant information for "Used Subject" without "Usage Scenario". The "Essen." (*i.e.,* Essential) column indicates whether the corresponding relevant information appears in the descriptions of all instances of the corresponding developer need type. We consider the other relevant information "non-essential" for a developer need type, meaning that it may be omitted from the description of a developer need of that type.

The 456 developer needs we identified are described in 1,027 sentences (764 unique ones) that provide relevant information. Each developer need is described by 2.3 sentences on average (min. 1, max. 9, median 2). These descriptive sentences constitute only 53.2% (764 of 1,436) of the sentences in the 266 questions providing relevant information. The implication is that, the developer needs from a question are described with only half of its sentences, on average. Further, we analyzed sentences providing relevant information related to developer needs. 75.7% of developer needs contain sentences providing duplicate types of relevant information, implying that we could summarize developer needs from questions in a concise way by using sentences without providing duplicate types of relevant information.

2.2.3 RQ3 (API Roles)

Table 3 shows the definitions of five API roles we identified and the last column is the number of APIs playing that role in at least one instance. *Suggested API* is a special API role and it is usually played by the APIs appearing in the answer as part of the solution for the developer need. APIs with other roles could appear in both questions and the answers, in the question as part of the developer need description and referenced in the answer. Table 4 shows the relations between API roles and developer need types and relevant information. Different developer need types could share API roles. Some developer need types include multiple API roles but not all corresponding API roles must exist in their developer need instances. We further analyzed the relations between API roles and relevant information, which are shown in the fourth column "Relevant Information". We found that APIs with a specific role tend to appear in sentences with specific relevant information. This suggests that we can design heuristic rules to determine the role of an API based on the relevant information type present in the sentence.

TABLE 4
Relationships Between Developer Need Types, API Roles, and Relevant Information Types

Developer Need Type	API Role	Count	Relevant Information
Functionality Implementation	Context API	188	Desired Functionality
	Suggested API	836	-
Non-Functional Improvement	Context API	39	Implemented Functionality
	Currently Used API	65	Suboptimal Implementation
	Suggested API	124	-
Functional Improvement	Context API	27	Expecting Result, Actual Result
	Currently Used API	73	Insufficient Implementation
	Suggested API	132	-
Error Handling	Error API	53	Erroneous Implementation, Error Occasion
	Exception Type API	31	Error Type
	Suggested API	77	-
Rational Analysis	Context API	80	Rationale Question
API Comparison	Context API	56	Comparison Subjects
Alternative Solution	Context API	8	Current Solution
	Suggested API	19	-
API Usage Learning	Context API	124	Used Subject

The 456 developer needs have 1,932 pertinent APIs in total (4.24 APIs on average, min. 1, max. 34, and median 3). 77.4% (353 of 456) of the developer needs have multiple pertinent APIs. Not all APIs appearing in the question/answer are pertinent for a developer need. The API could be mentioned only as an example or only related to one of the developer needs in the question (when multiple are present). The same API could be involved in different developer needs by playing different roles, which implies that those threads provide different information about the same API.

2.2.4 Comparison with prior taxonomies

We discuss here how the taxonomy we defined relates to the one proposed by Beyer *et al.* [9]. We contend that our taxonomy extends and complements the one proposed by

TABLE 5
Relationships Between Taxonomies

Our Taxonomy	Beyer *et al.* Taxonomy
Functionality Implementation	API Usage
Non-functional Improvement	Review
Functional Improvement	Discrepancy, API Change
Error Handling	Errors
Rationale Analysis	Conceptual, API Change, Review
API Comparison	API Change, Discrepancy
Alternative Solution	API Change
API Usage Learning	Learning

Beyer *et al.* Table 5 maps our **developer need types** to the categories from that taxonomy. There is no one-to-one correspondence between the categories. Since our taxonomy is finer-grained, one can consider that the categories from Beyer *et al.* that share our developer need types are related and the relationships are expressed by the common developer needs. Our other categories can be mapped to the ones derived by Beyer *et al.* , refining them into sub-categories. Moreover, our taxonomy includes *relevant information types* and *API roles*, which are not captured by Beyer *et al.* , hence extending that work.

2.2.5 Summary

Through our systematic annotation of 266 SO threads, we have identified 8 types of API-related developer needs and 17 types of information relevant to these developer needs. In addition, for the APIs related to these developer needs, we have identified the roles they play, leading to a set of 5 API roles.

2.3 Threats to Validity

The internal validity of our findings is dependent on whether our codes for developer need types, relevant information, and API roles are correct and complete. To alleviate this threat we had more than one coder participating in each coding activity and have reported the agreement. Our data is available in the replication package [26] and the study may be replicated in the future, confirming the validity of the codes. Another threat is that the categorization of developers needs has been done by students, not developers with industrial experience, cf. existing studies on potential differences [27], [28], [29]. To alleviate this threat we reported their Java development experience. The types identified by the students are not esoteric, so it is unlikely that more-informed coders would disagree with them.

The external validity of our findings is dependent on the number of threads we used in the study. We only analyzed 266 API-related threads, which affects generalizability. The findings may not generalize to other API-related questions. To alleviate this threat, as much as possible, we chose questions related to two popular libraries (JDK and Android) with high scores and we believe they are representative of typical API related questions. Scaling up our analysis to more SO threads may lead to the identification of additional developer need or relevant information types. The extension of our taxonomy is expected and desirable, but it will not invalidate the results we obtained. It would lead to better analysis tools in the future.

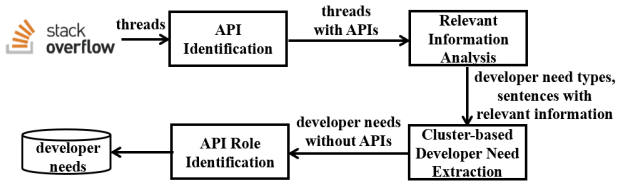

Fig. 3. Overview for Automated Identification of Developer Needs

TABLE 6
Regular Expressions for API Identification

Regular Expressions	Matching Examples
(\w+\.)+(\w)+	java.lang.String
([A-Z](\w)+)([A-Z](\w)+)+	StringBuffer
(\w+\.)*(\w)+\(\)	String.split()
(\w+\.)*(\w)+\([\w,]+\)	StringBuffer.insert(int,int)

3 Automated Identification of Developer Needs

We develop an approach for the automated identification of developer needs from threads. An overview of the approach is presented in Fig. 3. For a given thread from SO, we identify the APIs mentioned in the thread (Section 3.1). Then we use pretrained classifiers to identify developer need types in the question and describing sentences providing the relevant information (Section 3.2). After that, we extract developer needs from the question by a cluster-based approach (Section 3.3). Finally, we identify the APIs pertinent to each developer need and their roles (Section 3.4).

3.1 API Identification

For an API-related thread, we consider its question and its accepted answer for API identification. We identify APIs in three different places (*i.e.*, code snippets, stack traces, text) in different ways from the question and its accepted answer, based on our observations from the empirical study. APIs identified from different places may play different roles in developer needs. For example, the API identified from a code snippet may be a currently used API, the API identified from a stack trace may be an error API, and the API identified from text may be a context API.

API Identification from Code Snippets/Stack Traces. In Stack Overflow, code snippets and stack traces are both wrapped in `<pre><code></code></pre>`[6]. They can both contain APIs and their formats are quite different. Thus, we extract APIs from them in different ways. First, we extract each text wrapped in `<pre><code></code></pre>` and classify it into three categories: code, stack trace, and other. The classification is done by a list of regular expressions designed based on empirical study data, available in our replication package [26]. We extract APIs from code snippets with our own implementation of Baker [30], an approach to link APIs in incomplete code snippets to their qualified names. Because the official implementation is not available, we implemented a version of Baker by ourselves and built the oracle for Java APIs. To build the oracle for Baker, we used JavaPaser[7] to analyze the third-party libraries from

6. An example in https://stackoverflow.com/questions/2965747
7. https://github.com/javaparser/javaparser

Maven together with JDK 1.8 and Android 27. As a result, the oracle contains 946,325 classes, 9,711,745 methods and 3,448,472 fields for 32,238 libraries. We tested the Baker implementation on a test set and the results were comparable with those reported for the original implementation.

For a stack trace, we identify all APIs based on the structure of the stack trace by using regular expressions as well. The regular expressions are shown in Table 6, which take into account common API name conventions (similar to [31], [32]). For example, from the stack trace: `06-03 15:05:29.614: ERROR/AndroidRuntime(7737): java.lang.UnsupportedOperationException 06-03 15:05:29.614: ERROR/AndroidRuntime(7737): at java.util.AbstractList.remove(AbstractList.java:645)`, we extract two APIs *java.lang.UnsupportedOperationException* and *java.util.AbstractList.remove*.

API Identification from Text. We identify APIs in text in the following way. (1) Using regular expressions shown in Table 6 for common API name conventions, *e.g.,* camel-case (*e.g.,* `StringBuffer`), qualified name (*e.g.,* `java.lang.String`) and method name (*e.g.,* `String.split()`); (2) We consider the text wrapped in `<code>` as an API and the text shorter than two characters or more than one word will be filtered out. (3) We match the text with all APIs in the given API list, including the qualified names and the aliases of APIs. This part is optional and with the help of a given API list, we could identify APIs in text which could also be common words. For example, "string" in "How to split a string in Java" could match with `java.lang.String` if we provide JDK APIs for matching.

After we collect all APIs identified from the question and the accepted answer, we remove the duplicates. Qualified names and their aliases are considered duplicates, *e.g.,* `java.lang.String` and `String`. We ignore parameters for API methods because API methods mentioned in SO questions are often lacking parameters. We further filter out APIs that match SO tag names. SO tags could represent some common words. For example, "BeautifulSoup" will be identified as an API from text because of its camel case name and it will be filtered out by matching with the SO tag "beautifulsoup". If no APIs can be identified, then we consider this thread as not being API related and will not perform the next steps for developer needs extraction.

3.2 Relevant Information Analysis

We use text classifiers to classify the questions into developer need types and each describing sentence from the question into relevant information types.

Text Preprocessing. We preprocess the text of the question as follows. (1) We extract the text from HTML format using BeautifulSoup [25]. The text in the title and the question body are combined together as the question text. (2) Text wrapped by `<pre><code></code></pre>` is replaced with one of following placeholders: "`-CODE-`" for code snippet, "`-STACKTRACE-`" for stack trace, "`-XML/JSON-`" for XML or Json format data, "`-NUMBER-`" for number or list of numbers, and "`-TEXT-`" for plain text. The placeholder type is determined using regular expressions. We shorten the text as the text classifier underperforms for long text. Further, in the specific content replaced by placeholders, there are many unique words (*e.g.,* custom variable names), which would introduce noise to the classifiers. Moreover, the placeholders allow the classifiers to focus on the type and the context of the content replaced by placeholders, rather than the specific content. For example, "`-STACKTRACE-`" usually implies an "Error Handling" developer need type in the question and "An exception was thrown while I run the following code: -CODE-." usually implies an "Erroneous Implementation". Note that we record the original content replaced by the placeholder for later restoration. The relationships between code snippets/stack traces and the API identified from them in Section 3.1 are also recorded. (3) We add a placeholder "`-API-`" before each API mention in the text (identified in Section 3.1). If the API mention is ending with "Exception" or "Error", the added placeholder is "`-EXCEPTION-`". The presence of APIs is an important feature for some developer need types (*e.g., API Usage Learning*, *Error Handling*) and relevant information (*e.g., Used Subject*, *Error Type*).

Developer Need Type and Relevant Information Classification. We define this task as a sentence classification and we design a two-phase classifier. In the first phase, we classify the question text into developer need types. We train binary classifiers for each developer need type, which classify an input question into two classes "Yes" or "No", representing whether the input question contains this type of developer needs or not.

In the second phase, we classify describing sentences into relevant information types. For each type of relevant information, we train a binary classifier that classifies a sentence into two classes: "Yes" and "No", representing whether this sentence provides this type of relevant information or not. If the question is classified into one developer need type, we use all relevant information classifiers that belong to this developer need type. Based on the results of RQ2 (see Table 2), some types of relevant information are *essential* for the related developer need types (*e.g.,* "*Used Subject*" is essential for "*API Usage Learning*"). We use this finding to improve the sentence classification. We combine the results of the sentence classifiers from the same question together to fix classification errors. For example, if a sentence is classified as "*Usage Scenario*" and none of the sentences in the same question is classified as "*Used Subject*", then we will adjust the classification result of "*Usage Scenario*" from "Yes" to "No". If a sentence can not be classified as "Yes" by any classifier, it will be annotated as "*useless*".

We train multiple binary classifiers for two reasons: (1) A previous study for knowledge pattern classification for sentences of API reference documentation [33] has shown that when there are many classes for sentence classification, training a binary-classifier for each class is better than directly training a single complex classifier; and (2) Our classification is a multi-label task, *i.e.,* each question could have several developer need types and each description sentence could provide multiple types of relevant information. It is easier to collect training data for training multiple binary classifiers.

We used FastText [34] to implement the two classifiers. FastText is a fast approximation of the softmax classifier designed by Facebook, which is based on n-gram features and dimensionality reduction. It is fast enough that we can

run iterative tests quickly.

We asked five MS students (with more than two years of Java development experience each) to annotate sentences by relevant information type, in addition to those we annotated in the empirical study (Section 2.1). To make the annotation easier, we designed the annotation task as a binary classifier as well. For each sentence, we show a candidate relevant information type and the annotators only need to annotate "Yes" or "No". If a sentence of the question is annotated as the relevant information type, we will consider the question annotated with the developer need type that the relevant information describes. We used doccano [35], an online annotation tool to support the annotation. We sampled the sentences for each relevant information type in the following ways: (1) random sampling; (2) sentences containing keywords related to the corresponding relevant information (*e.g.,* "difference between" for "Compared Subject"); (3) sentences classified as positive by the classifier trained on the already annotated data; (4) sentences with high text similarity (based on TF-IDF) with already annotated data; (5) sentences in a question that were already annotated with the developer need type for this type of relevant information. There is no priority between different sampling ways. We took the union set of the sentences selected by the five ways. The same sentences may be sampled for different relevant information types. Each sampling sentence was annotated by two persons and a third person was assigned to resolve conflicts. The different sampling approaches were used for reducing annotation costs and getting more annotated data. The annotated data is provided in the replication package [26].

3.3 Cluster-based Developer Need Extraction

Since a question may contain several developer need instances, as shown in Fig. 2, we extract the developer need instances in the question via clustering.

First, we filter out sentences annotated as "*useless*" and cluster the remaining ones by relevant information. A sentence cluster contains sentences from the same question with information types relevant to the same developer need type, *e.g.,* sentences annotated with "*Used Subject*" and "*Usage Scenario*" are in the same cluster, after clustering based on relevant information.

Then we refine the clusters to separate different developer need instances. We represent each sentence into a fixed-length vector by averaging the vectors of all words in the sentence. The vector for a word is obtained from a 100-dimensional Word2Vec [36] model pre-trained on the Wikipedia corpus[8]. The model is tuned based on the corpus of all SO threads tagged with "java" by gensim [37]. The similarity between two sentences S_1 and S_2 is computed by Eq. (1). Eq. is short for Equation.

$$Sim(S_1, S_2) = (cos(V_{S_1}, V_{S_2}) + 1)/2 \quad (1)$$

For each sentence cluster C_s, we refine the cluster in two phases. In the first phase, we cluster the sentences that provide the same relevant information types. We use DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [38] as the clustering algorithm. We also add all other sentences from the question to act as noise in the clustering. After obtaining the clusters, we remove all the noise sentences and obtain several non-empty clusters.

In the second phase, we use the list of clusters $Set(Cluster)$ and we merge them. In each merging, we remove the two most similar clusters $Cluster_1$ and $Cluster_2$ from $Set(Cluster)$ that do not contain the same relevant information type and merge $Cluster_1$ and $Cluster_2$ as $Cluster_{new}$ and then add the $Cluster_{new}$ back to the $Set(Cluster)$. We stop when $Set(Cluster)$ has only one $Cluster$ or we can not merge any cluster pair. The similarity of two clusters $Cluster_1$ and $Cluster_2$ is the the highest sentence similarity between the sentences in the two clusters.

Finally, we remove $Clusters$ in $Set(Cluster)$ that do not contain all essential relevant information types for the corresponding developer need types. Each $cluster$ left in the $Set(Cluster)$ corresponds to a developer need instance.

3.4 API Role Identification

Given a developer need, we first select APIs pertinent to the developer need as candidates. Then, we classify each candidate API into an API role using rule-based classifiers.

Candidate APIs Selection. Based on our experience, an API is relevant to a developer need DN in two cases: (1) it explicitly appears in the sentence of this developer need, *e.g., java.lang.String* in "How to split a string in Java"; (2) it is not explicit in the sentence of this developer need, but its name or the context of the API implies its relevance, *e.g.,* the sentence "Just use the appropriate method: String#split()." from the answer of "How to split a string in Java" mentions the *java.lang.String.split* API and its name already shows the relevance.

Then, for a developer need DN and each API E in the same thread, identified in Section 3.1, we calculate their relevance score based on the location of E or the context similarity by Eq. (2). If DN contains the API E in any of its sentences, then $Contain(DN, E) = 1$; otherwise, $Contain(DN, E) = 0$. The context similarity between E and DN is calculated by Eq. (3) based on a Word2Vec [36] model. We represent E and DN by averaging the vectors of all words in their description text. For E, the description text is the combination of all sentences in the thread that mentioned the API E and all aliases of E (same as Section 2.1); For DN, the description text is the combination of all sentences in developer need DN. The Word2Vec model is the same as we used for sentence clustering (see Section 3.3). We filter out APIs with relevance score less than a threshold T.

$$Rel(DN, E) = max(Contain(DN, E), Sim(DN, E)) \quad (2)$$

$$Sim(DN, E) = (cos(V_{DN}, V_E) + 1)/2 \quad (3)$$

API Role Classification. Based on the results of RQ3 (see Section 2.2.3), we know the relations between API roles and the relevant information (see Table 4). We designed rule-based binary classifiers for each type of API role and use each classifier to classify each API E for developer need DN as: (1) *Context API*, if E appears in a sentence classified as one of: *functionality implementation*, *implemented functionality*, *expected result*, *actual result*, *rationale analysis*, *comparison subject*, *used subject*, *current solution*; (2) *Currently*

8. https://github.com/3Top/word2vec-api

Used API, if E appears in a sentence classified as one of: *suboptimal implementation*, *insufficient implementation*; (3) *Error API*, if E appears in a sentence classified as one of: *erroneous implementation*, *error occasion*; (4) *Exception Type API*, if E appears in a sentence classified as *error type* and containing "Error" or "Exception" in its name; (5) *Suggested API*, if E only appears in the answer.

It is worth noting that all API role classifiers assume that the developer need type must have corresponding API roles. If an API is classified as having multiple API roles, then we select the final role, following the priorities: *Exception Type API* > *Error API* > *Context API* > *Currently Used API* > *Suggested API*.

3.5 Evaluation

As shown in Fig. 3, our approach contains four steps: API identification, relevant information analysis, cluster-based developer need extraction, and API role identification. To verify the effectiveness of our approach, we evaluated the main parts of our approach: developer need type classification, relevant information classification, cluster-based developer need extraction, and API role identification. API identification mainly consists of existing methods and heuristic rules, which are not the focus of our approach, and we did not perform a separate evaluation for that part. The relevant information analysis includes developer need type classification and relevant information classification, which we evaluated separately.

Developer Need Type and Relevant Information Classification. We obtained annotation data for developer need types and relevant information from our empirical study in Section 2.1 and additional annotation from Section 3.2. As a result, we have 6,985 annotated questions for developer need classification and 14,718 annotated sentences for relevant information classification. For each developer need type and relevant information, we conducted 10-fold cross validation on the annotated data. That is, we randomly divided the annotated data into 10 folds and each time used 9 folds as the training set and the remaining one fold as test set. We trained the 8 developer need type classifiers and 17 relevant information classifiers based on the official implementation of FastText on GitHub[9]. We did not compare FastText with other baselines because the purpose of this evaluation is to show that FastText is an acceptable choice. FastText can be replaced with a more advanced model, based on progress in the NLP domain, in the future. The average precision, recall and F1 for all classifiers across the 10 folds are shown in Table 7. "P" means precision and "R" means recall, while "F1" is the harmonic mean of the two. For the developer need type classification, precision, recall and F1 for FastText are all above 0.8, while for relevant information classification, they are all above 0.7 (with one exception).

We attribute the lower classification accuracy on some relevant information to the size of the training data, *e.g.*, the precision of "*Comparison Scenario*" is only 0.64 because it is not essential for API comparison (*i.e.*, it may be missing), hence we only had 67 positive samples in the annotation data. At the same time, we also observed that our method does not perform well when the question body is very lengthy with vague descriptions.

9. https://github.com/facebookresearch/fastText

Cluster-based Developer Need Extraction. We grouped a list of sentences providing relevant information from 266 questions into 456 developer needs in Section 2.2.1. We used this data as the ground truth for evaluating our developer need extraction, using as input the sentences with human annotated relevant information.

For each question, we obtained a list of sentence pairs from its ground truth developer needs. A sentence pair contains two sentences that share a developer need. We compared the extracted developer needs and ground truth developer needs on sentence pairs to compute the precision and recall for this question. The average precision and recall for all questions are 0.91 and 0.92 respectively, which indicates that if the relevant information classification is accurate enough, we can extract developer needs correctly.

API Role Identification. We randomly selected 5 developer need instances for each developer need type from the 456 developer needs identified before, in total 40 developer need instances with 145 APIs (44 for *context API*, 21 for *currently used API*, 64 for *suggested API*, 11 for *error API*, 5 for *error type API*). We used the approach to identify the APIs with roles for these 145 APIs. Comparing with the human annotated ground truth, the average precision and recall for identification of APIs roles are 77.5% and 69.0%. The precision and recall for *context API* are 73.1% and 86.4%; for *suggested API*, 89.7% and 54.7%; for *currently used API* 61.1% and 52.5%; for error API *73.3% and 100%*; for *error type API* 100% and 100%. The lower accuracy for *suggested API* and *currently used API* roles is caused by the fact that APIs with these two roles often appear in code snippets (a limitation of Baker). Another problem is that the mention of the API in the text is often a common word and we did not recognize it or link it correctly to its qualified name. Identifying the API mentions in SO posts more accurately is not the focus here, but subject of future work.

Summary. Our approach can accurately (83.6% precision and 85.4% recall, in average) identify developer needs, relevant information types, and API roles, in SO threads.

4 Large-Scale SO Question Analysis

The motivation for our research is that many SO posts contain multiple developer needs, hence the need for our detectors. We used our tool to extract developer needs from 213,959 SO threads, as follows: (1) tagged with "java"; (2) created time is before March 2016; (3) has at least one accepted answer or one answer with at least one vote. This particular SO data was also used in previous research on SO question retrieval [19].

Our approach extracted developer needs from 83.6% of the questions (178,868 of 213,959). It identified 337,267 developer needs, 66,074 for *functionality implementation*, 37,441 for *non-functionality improvement*, 70,881 for *functional improvement*, 68,419 for *error handling*, 47,233 for *rationale analysis*, 24,402 for *API comparison*, 11,570 for *alternative solution* and 11,247 for *API usage learning*. *Functionality implementation* is the most common need, followed by *functional improvement* and *error handling*, which is consistent with

TABLE 7
Evaluation of Developer Need Type and Relevant Information Classification

Developer Need Type							
Category	P	R	F1	**Category**	P	R	F1
Functionality Implementation	0.90	0.95	0.92	Non-Functional Improvement	0.91	0.97	0.94
Functional Improvement	0.89	0.94	0.91	Error Handling	0.92	0.97	0.95
Rationale Analysis	0.91	0.96	0.93	API Comparison	0.93	0.99	0.96
Alternative Solution	0.91	0.98	0.94	API Usage Learning	0.84	0.98	0.91
Relevant Information							
Desired Functionality	0.78	0.94	0.85	Implemented Functionality	0.80	0.95	0.87
Suboptimal Implementation	0.74	0.95	0.83	Improvement	0.84	0.95	0.89
Expecting Result	0.75	0.90	0.81	Actual Result	0.82	0.95	0.88
Insufficient Implementation	0.78	0.94	0.85	Error Type	0.82	0.97	0.89
Error Occasion	0.74	0.95	0.83	Erroneous Implementation	0.80	0.99	0.88
Rationale Question	0.81	0.90	0.84	Comparison Subject	0.93	0.99	0.96
Comparison Scenario	0.64	0.88	0.73	Current Solution	0.90	0.98	0.94
Alternative Description	0.78	0.98	0.87	Used Subject	0.74	0.92	0.82
Usage Scenario	0.72	0.90	0.79				

our intuition. Developers often ask for help on SO for implementing specific functionality or debugging an existing implementation. As opposed to *functionality implementation*, *API usage learning* is the least common. The reason may be that most of the questions on SO are task-oriented. The problem for developers is that they do not know which API to use, not how to use a specific API.

For 106,026 questions with multiple developer needs, we found that in 66.4% of the questions (70,387 of 106,026), at least one sentence provides relevant information for different developer needs at the same time. We observed two patterns for describing multiple developer needs in a question. One is sequential, where developers describe different developer needs in turn, and the other is interrelated where multiple needs are mentioned in parallel.

Further, 66.3% of the developer needs are not included in the sentences from the title. For questions with one developer need, the developer needs from 63.5% questions do not contain the sentences from title. Because sometimes the title is too vague and low quality and does not reflect the developer needs of the developer in detail, *e.g.,* in the question with title "Simple data thread question - java" [10]. For questions with multiple developer needs, only 9.5% of the questions contain sentences in the title for describing all needs. The other 90.5% questions contain at least one developer need that is not described in the title. Sometimes it is hard to express multiple developer needs in a short title, which leads to some developer needs being described in the body of a question only, *e.g.,* the developer need " I also want to check if the string has '-' in it." is not reflected by the title "How to split a string in Java" of the question[11].

We conclude that a large proportion of SO threads refer to multiple developer needs and in 90% of these cases, titles do not describe all the needs. This observation has implications on using question titles when retrieving related questions, as they may have insufficient information.

5 Relevant Question Retrieval

To show the usefulness of our taxonomy and automated identification of developer needs, we design an approach for the retrieval of API-related questions for a given query, based on developer needs.

5.1 Retrieval Approach

The main idea of our retrieval approach is that we match the user's query with the question at the granularity of developer needs, not just the title of the question or the whole question body. The retrieval approach has two parts, an offline part for developer needs extraction, and an online part to retrieve a set of questions related the given query.

For the offline part, we first collect questions to be retrieved as a question corpus and extract developer needs from questions in the corpus. At the same time, we train a TF-IDF metric using gensim [37] based on the question corpus. Each question is treated as a document and preprocessed in the same way as described in Section 3.2. The TF-IDF metric measures the importance of a word in the corpus.

For the online part, we define the relevance between a given query q in natural language and an API-related question Q, using the extracted developer needs, as a linear combination of two similarity measures, (see Eq. 4). First, we calculate the relevance between the query q and each developer need DN (*i.e.,* $Sim_{text}(q, DN)$) in the question Q, and select the DN most relevant to q. Then we use the relevance between q and DN as the relevance between q and Q. When calculating the relevance between q and DN, both text similarity and API similarity are considered and weighted by two weights W_1 and W_2 respectively (See Eq. 4).

We convert q and DN into bags of words W_q and W_{DN} respectively after stop word removal and stemming. An asymmetric text similarity $Sim_{text}(q \rightarrow DN)$ between the query q and the developer need DN is computed by Eq. 5. $Sim(w_q, W_{DN})$ is computed as $\max_{w_{DN} \in W_{DN}} sim(w_q, w_{DN})$ where $sim(w_q, w_{DN})$ is the cosine similarity of two vectors of w_q and w_{DN} normalized to the range between 0 and 1. That is, for a word w_q in the query q, we select a word from DN that has the closest semantics and use the similarity score between w_q and the selected word as the similarity between w_q and DN. The importance of each word w_q in the query q is different. We use the TF-IDF value to measure the importance of w_q.

10. https://stackoverflow.com/questions/4719146
11. https://stackoverflow.com/questions/3481828

Intuitively, the most important word in the query should carry more weight when calculating relevance.

The symmetric text similarity $Sim_{text}(q, DN)$ between the query q and the developer need DN is computed as Eq. 6, which is the average of the two asymmetric text similarities between q and the developer need DN.

The API similarity is calculated with Eq. 2 using the number of APIs that appear in the query q and the developer need DN at the same time. API_q is the set of APIs identified from the query q and API_{DN} is the set of APIs involved in DN.

$$Rel(q, Q) = \max_{DN \in Q} W_1 * Sim_{text}(q, DN) + W_2 * Sim_{API}(q, DN) \quad (4)$$

$$Sim_{text}(q \rightarrow DN) = \frac{\sum_{w_q \in W_q} TFIDF(w_q) * Sim(w_q, W_{DN})}{\sum_{w_q \in W_q} TFIDF(w_q)} \quad (5)$$

$$Sim_{text}(q, DN) = (Sim_{text}(q \rightarrow DN) + Sim_{text}(DN \rightarrow q))/2 \quad (6)$$

$$Sim_{API}(q, DN) = |API_q \cap API_{DN}|/|API_q| \quad (7)$$

Based on the relevance between the query and each question in the question corpus, the questions in the corpus are ranked and the top-N ranked questions are returned as the relevant questions for the query.

The similarity measures in our approach (Eq. 5 and Eq. 6) are related to the ones used by AnswerBot [19], a tool for the retrieval and summarization of SO posts. The measures used by AnswerBot directly compute the text similarity between the query and the question represented by the title as the score for ranking. In contrast, our approach uses the developer needs, not the title of the question. Further, AnswerBot uses IDF as the weight in Eq. 6 and we use TF-IDF as the weight, because the developer needs are usually longer than the title and important word may have higher frequency. In addition, we also use the API-similarity, compared to AnswerBot, that only uses text similarity, because we focus on API related question.

5.2 Evaluation

We compare our retrieval approach with AnswerBot [19] and we refer to it as $baseline_{title}$.

We used the implementation of $baseline_{title}$ from the replication package[12] of AnswerBot. Since we want to compare the retrieval using the complete question (title and body), not just the title, we have modified $baseline_{title}$ and obtained its variant $baseline_{full}$, which uses the title and the body of a question together, when calculating similarity to a query.

Data. We obtained the 100 SO questions that were used to evaluate AnswerBot from the authors' replication package. Then we manually removed questions that were not API-related and obtained 64 for our evaluation. AnswerBot handles any type of question, as it relies on textual similarity only, whereas our approach focuses on API-related questions only. We used the same 213,959 questions from Section 4 as the retrieval corpus. This corpus does not include the 64 questions used as retrieval tasks. We further used the developer needs extracted from these questions (see Section 4) for retrieval with our approach.

TABLE 8
Top@K Accuracy And MRR of Our Approach And the Baseline Approaches In Relevant Question Retrieval

Approach	Top@1	Top@5	Top@10	MRR
$Baseline_{title}$	0.484	0.797	0.828	0.617
$Baseline_{full}$	0.438	0.734	0.797	0.568
Our Approach	**0.625**	**0.828**	**0.859**	**0.698**

Protocol. For each of these 64 retrieval tasks, we used the title of the question as the query and retrieved the top 10 results using $baseline_{title}$, $baseline_{full}$, and our approach. We merged the search results for the same task and removed duplicate questions. Then we invited 8 participants (MS students with more than three years of Java development experience each) to assess the relevance of the results. All results retrieved for a task were assessed by the same two participants independently. When assessing the retrieved results for a task, participants were asked to read the SO threads for the task carefully to ensure they understood the task. They judged the relevance of each retrieved question to the task, *i.e.*, whether the retrieved question is a hit for the task. Note that a related question does not need to exactly match the task. All retrieval results for the same task were shuffled before assessment and participants did not know which approach retrieved the result. If the assessment of two annotators for the same retrieved result was inconsistent, a third annotator was assigned to produce an additional judgment, and the final annotation was determined based on majority. The agreement between the judgments was substantial (*i.e.*, Cohen's Kappa coefficient [24] of 0.684).

Results. Based on the judgments we compute the Top@1, Top@5, and Top@10 accuracy measures, which gives the average of how many results returned in the top 1, 5, or 10 (respectively) are relevant. We also use MRR (Mean Reciprocal Rank) [39] to compare the approaches, as it reflects the ranking of the first relevant questions in the returned results. These measures are used by previous work that evaluated AnswerBot and are commonly used measures in information retrieval. In each case, higher values represent a better performance. The measures are shown in Table 8. Our approach achieves the best results for all metrics, while $baseline_{full}$ the worst. We argue that the main reason explaining the results is that, although the question bodies contain useful information, they also contain a lot of noise, which hampers the retrieval. Using the content of the question body did not bring improvement, especially when the query is a short title. Our approach uses the developer needs extracted from questions, which arguably are less noisy than other information in the question bodies.

5.3 User Study

In order to further show how our approach can help developers, we conduct a user study by asking participants to complete programming tasks with the help of our approach and the help of a baseline.

Data. We selected 6 Java programming exercises shown in Table 9 from Practice-it[13] as the tasks for participants

12. https://github.com/XBWer/AnswerBot.git

13. https://practiceit.cs.washington.edu/

TABLE 9
Tasks for User Study

ID	Task Name	Summary	Group
T1	plusScores	Reads students' score records and calculate the proportion of plus scores for each student	TA
T2	readEntireFile	Reads a file and returns the entire text contents of that file as a String	TA
T3	mostCommonNames	Reads a file that contains several names in each line and find names that occurs the most frequently in each line of that file	TA
T4	inputStats	Reads a file and report various statistics about the file's text, e.g., the number of lines in the file, the longest line	TB
T5	removeDuplicates	Reads an ArrayList of Strings and eliminates any duplicates from this list	TB
T6	countUnique	Reads List of integers and returns the number of unique integer values in the list by using Set	TB

to complete. Those tasks cover different domains, such as string operations, file reading and writing, lists, and sets. We ensure that some JDK APIs are involved in each programming exercise solutions. We randomly divided the 6 tasks into two roughly equivalent groups (*TA* and *TB*) according to difficulty.

Our Approach and Baseline. We only use $baseline_{title}$ from Section 5.2 as the baseline, because the experiment showed that it outperforms $baseline_{full}$. We developed web pages for both our approach and the baseline. According to the query provided by the user, the web pages will display the titles of the top-100 most relevant questions obtained by our approach or the baseline. We show a summary for each question in the search results. For our approach, the summary of a question is the developer need most relevant to the query in each question. For the baseline, the summary of a question is the first 200 characters of the question body.

Protocol. We asked 10 Master students with 1–3 years of Java programming experience to participate in the study. We conducted a pre-experiment survey on their Java programming experience and divided them into two roughly equivalent groups (*GA* and *GB*) based on the survey. For *GA*, participants complete *TA* with the baseline and *TB* with our approach. For *GB*, participants complete *TB* with the baseline and *TA* with our approach. When completing a task, participants must submit the complete code for each task and the code is reviewed by the authors to confirm its correctness. Participants can write their own queries and search multiple times with our approach or the baseline. We will record the time they need to complete the task. If a participant does not complete the task within 15 minutes, the participant will stop and the completion time will be recorded as 15 minutes.

After the completion of their tasks, the participants were interviewed and were asked to describe how they used the SO retrieval tools and what issues they encountered, if any.

Results. We received 54 solutions that were completed on time. We checked the participants' submitted code for each task and evaluated their correctness by comparing with the ground truth solution. From a total of 54 submitted solutions, we received 6 incorrect solutions (3 using our approach and 3 using the baseline), mainly because participants did not understand the task correctly. We removed those incorrect solutions from the following analysis.

Fig. 4 shows the participants' completion time using our approach and the baseline. Using our approach, participants completed the tasks 27.0% faster (378s vs. 518s on average) compared to the baseline. In the six tasks, except the *T4*, using our approach is faster than using the baseline. *T4* is an exception is mainly because some useful questions for

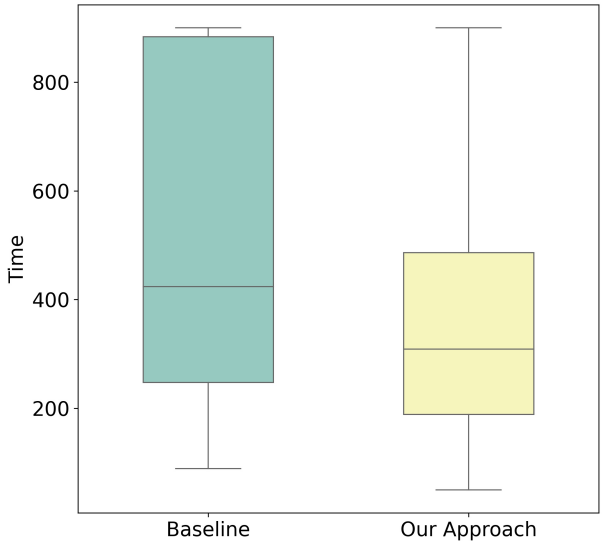

Fig. 4. Completion Time for Our Approach and Baseline

T4 rank higher when using the baseline. We used Welch's t-test [40] for verifying the statistical significance of the difference between our approach and the baseline on completion time. The difference is not statistically significant ($p = 0.06$).

The feedback we received from participants shows that when using the baseline they usually spent a lot of time to check whether the questions shown in the search results are relevant to their query. They often find that it does not meet their needs after reading the description of a question in detail. Conversely, the summary of the developer needs displayed by our approach can help them judge whether the question is relevant faster without reading the question in detail.

We conclude that using our approach, which retrieves questions based on developer needs, decreases the amount of time developers need for completing programming tasks.

5.4 Threats to Validity

A threat to the internal validity is related to the subjective judgment of the annotators during data annotation. To alleviate this threat, we follow commonly used data analysis principles, such as, multiple annotators, conflict resolution, and reporting agreement coefficients, where appropriate. Another threat to the internal validity of the user study is

related to individual differences between the 10 students. To alleviate this threat, we conducted a pre-experiment survey about participants' Java programming experience and divided them into two roughly equivalent groups based on the survey. The 6 tasks were divided into two roughly equivalent groups, according to difficulty as well. We conducted a crossover study and adopted a balanced treatment distribution for the groups. Another threat is related to the prior knowledge of students. Depending on the knowledge that students have, they may be able to solve tasks without further searching with a tool. To alleviate this threat, we asked participants to search at least once when completing a task with our approach or the baseline.

A threat to the external validity of our evaluation and user study is related to the limited number of subjects (*e.g.,* questions, tasks, and participants) considered in the evaluation and user study. The evaluation and user study may not generalize to broader development scenarios.

6 Discussion

Discussions on SO are often rich in information and can be quite complex. Existing research has treated the entire thread/question as plain text and generated coarse-grained classifications. We posit that by analyzing the developer needs with relevant information and API roles involved in SO discussions, we can achieve a deeper understanding of the semantics of SO discussions. This is a crucial step towards converting SO discussions from unstructured natural language into structured knowledge enabling us to make better use of the information contained in these discussions.

Multiple applications are possible using such structured information. Section 5 has already shown one of the possible applications, *i.e.,* retrieving API-related questions based on developer needs. Other possible applications are as follows.

(1) **Supplementing API reference documentation with real developer needs and corresponding solutions.** The scenario-oriented knowledge about the same API is gathered according to the developer needs. Developers can explore scenario-oriented knowledge about an API based on different aspects, such as the developer need types, API roles.

(2) **Relevant SO question recommendation for a given question, with relevance established based on shared developer needs and API roles.** Developers can investigate relevant SO questions according to their interests. For example, for a functionality implementation question, developers may be concerned about API comparison questions comparing the suggested APIs with other alternative APIs or error handling questions involving the suggested APIs as error APIs.

(3) **Explaining API recommendation results based on a query, highlighting the developer need related to the query and the recommended APIs.** Developers can understand the relationship between a recommended API and the query based on the developer need provided, so as to select the appropriate API more accurately and quickly.

7 Related Work

Previous research has categorized Stack Overflow content. Treude *et al.* [4] identified ten categories of SO questions: *i.e., how-to*, *discrepancy*, *environment*, *error*, *decision help*, *conceptual*, *review*, *non-functional*, *novice*, *noise*. Nasehi *et al.* [41] described SO question types along two dimensions: (1) the main technology or construct that the question revolves around and usually can be inferred from the question tags; (2) the main concerns of the questioners and what they wanted to solve (*i.e., Debug/Corrective*, *Need-To-Know*, *How-To-Do-It*, and *Seeking-Different-Solution*). Allamanis *et al.* [16] used topic modeling to uncover question categories and identified five question categories: *Does not work*, *How/Why something works*, *Implement something*, *Way of using*, and *Learning*. Beyer *et al.* [18] identified eight SO question types: *How to...?*, *What is the Problem...?*, *Error...?*, *Is it possible...?*, *Why...?*, *Better Solution...?*, *Version...?*, and *Device...?*. In a similar study, Rosen *et al.* [15] manually investigated 384 mobile-related posts and categorized them into three main categories: *How*, *What*, and *Why*.

These studies tend to gloss over the complexity of SO questions, in particular the fact that they may express multiple concerns of a developer. To address this gap, we propose a fine-grained taxonomy of developer needs in SO posts, together with the information needed to express them, and the roles of the APIs in pertinent to the needs.

In addition, several studies have analyzed the discussions around domain-specific topics on Stack Overflow, such as security [14], mobile development [15], Android testing [10], requirements engineering [11], and configuration-as-code [13]. In contrast, our focus is on API-related SO questions, orthogonal to the application domain.

Many researchers have proposed applications based on Stack Overflow data to help developers, *e.g.,* by building a question-answering system based on question-and-answer pairs [42], by integrating Stack Overflow post recommendations into the IDE [43] and through API recommendations [44].

In addition to AnswerBot [19], several approaches were previously proposed for retrieving information from Stack Overflow [45], [46], [47], [48]. These approaches focus on retrieving entire posts (as opposed to related questions) and ignore the explicit developer needs.

Other studies have targeted mining knowledge from Stack Overflow. For example, Wong *et al.* [49] mined code snippets and their descriptions from Stack Overflow to support comment generation for similar code snippets. Zhang *et al.* [50] developed BDA (Bing Developer Assistant) to recommend sample code mined from GitHub and Stack Overflow. Treude *et al.* [51] used a machine-learning model to identify insight sentences in Stack Overflow posts that could be used to augment API reference documentation. Xu *et al.* [19] proposed AnswerBot to summarize the answers to a question. Uddin *et al.* [52] developed a tool, Opiner, to present summaries of opinions about an API from Stack Overflow. They focus on extracting API-related opinion sentences from Stack Overflow posts and summarizing them into several aspects (*e.g.,* performance, usability). Unlike them, we focus on analyzing the developer needs in the questions and the role that the API plays in different developer needs.

8 Conclusion and Future Work

Our new taxonomy, focuses on API-related questions in Stack Overflow and defines fine-grained developer needs

and pertinent information, complementing existing frameworks, while addressing their gaps. API-related SO questions describe such developer needs with specific types of information, while the pertinent APIs play various roles in the answers. These aspects are also captured by our new taxonomy.

We found that the fine-grained developer needs and relevant information, together with API roles in SO threads, can be identified automatically with high accuracy, using a combination of heuristic-based and supervised learning approaches. We argue that these elements capture the essential information of SO questions. A practical application of our taxonomy is the automated retrieval of SO questions, based on the automatically extracted developer needs and API roles. A comparison with state-of-the-art baseline approaches, which use textual similarities, supports our argument.

In the future we will focus on improving our tools and the retrieval approaches, as well providing additional API knowledge services based on the identification of API-related developer needs in Stack Overflow.

9 Data Availability

All the data used in the empirical studies is included in the replication package [26].

References

[1] G. Uddin, B. Dagenais, and M. P. Robillard, "Temporal analysis of API usage concepts," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. IEEE Computer Society, 2012, pp. 804–814.

[2] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," *Empir. Softw. Eng.*, vol. 16, no. 6, pp. 703–732, 2011.

[3] W. Maalej and M. P. Robillard, "Patterns of knowledge in API reference documentation," *IEEE Trans. Software Eng.*, vol. 39, no. 9, pp. 1264–1282, 2013.

[4] C. Treude, O. Barzilay, and M. D. Storey, "How do programmers ask and answer questions on the web?" in *33rd International Conference on Software Engineering, ICSE 2011, May 21-28, 2011, Waikiki, Honolulu, HI, USA*. ACM, 2011, pp. 804–807.

[5] M. Nassif, C. Treude, and M. P. Robillard, "Automatically categorizing software technologies," *IEEE Trans. Software Eng.*, vol. 46, no. 1, pp. 20–32, 2020.

[6] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Trans. Software Eng.*, vol. 41, no. 6, pp. 565–581, 2015.

[7] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *29th International Conference on Software Engineering, ICSE 2007, May 20-26, 2007, Minneapolis, MN, USA*. IEEE Computer Society, 2007, pp. 344–353.

[8] R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. IEEE Computer Society, 2012, pp. 987–996.

[9] S. Beyer, C. Macho, M. Pinzger, and M. D. Penta, "Automatically classifying posts into question categories on stack overflow," in *26th Conference on Program Comprehension, ICPC 2018, May 27-28, 2018, Gothenburg, Sweden*, 2018, pp. 211–221.

[10] I. K. Villanes, S. M. Ascate, J. Gomes, and A. C. Dias-Neto, "What are software engineers asking about android testing on stack overflow?" in *31st Brazilian Symposium on Software Engineering, SBES 2017, September 20-22, 2017, Fortaleza, CE, Brazil*, 2017, pp. 104–113.

[11] Z. S. H. Abad, A. Shymka, S. Pant, A. Currie, and G. Ruhe, "What are practitioners asking about requirements engineering? an exploratory analysis of social q&a sites," in *24th IEEE International Requirements Engineering Conference, RE 2016, September 12-16, 2016, Beijing, China*, 2016, pp. 334–343.

[12] S. Beyer and M. Pinzger, "Grouping android tag synonyms on stack overflow," in *13th International Conference on Mining Software Repositories, MSR 2016, May 14-22, 2016, Austin, TX, USA*, 2016, pp. 430–440.

[13] A. Rahman, A. Partho, P. Morrison, and L. Williams, "What questions do programmers ask about configuration as code?" in *4th International Workshop on Rapid Continuous Software Engineering, RCoSE@ICSE 2018, May 29, 2018, Gothenburg, Sweden*, 2018, pp. 16–22.

[14] X. Yang, D. Lo, X. Xia, Z. Wan, and J. Sun, "What security questions do developers ask? A large-scale study of stack overflow posts," *J. Comput. Sci. Technol.*, vol. 31, no. 5, pp. 910–924, 2016.

[15] C. Rosen and E. Shihab, "What are mobile developers asking about? A large scale study using stack overflow," *Empir. Softw. Eng.*, vol. 21, no. 3, pp. 1192–1223, 2016.

[16] M. Allamanis and C. A. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *10th Working Conference on Mining Software Repositories, MSR 2013, May 18-19, 2013, San Francisco, CA, USA*, 2013, pp. 53–56.

[17] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger, "Analyzing the relationships between android API Classes and Their References on Stack Overflow," Technical Report. University of Klagenfurt, University of Sannio, Tech. Rep., 2017.

[18] S. Beyer and M. Pinzger, "A manual categorization of android app development issues on stack overflow," in *30th IEEE International Conference on Software Maintenance and Evolution, ICSME 2014, September 29 - October 3, 2014, Victoria, BC, Canada*, 2014, pp. 531–535.

[19] B. Xu, Z. Xing, X. Xia, and D. Lo, "Answerbot: automated generation of answer summary to developersź technical questions," in *32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, October 30 - November 03, 2017, Urbana, IL, USA*, 2017, pp. 706–716.

[20] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, pp. 101 – 77, 2006.

[21] M. P. Robillard and C. Treude, "Understanding wikipedia as a resource for opportunistic learning of computing concepts," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education, SIGCSE 2020, March 11-14, 2020, Portland, OR, USA*, J. Zhang, M. Sherriff, S. Heckman, P. A. Cutter, and A. E. Monge, Eds. ACM, 2020, pp. 72–78.

[22] C. Lima and A. C. Hora, "What are the characteristics of popular APIs? A large-scale study on java, android, and 165 libraries," *Softw. Qual. J.*, vol. 28, no. 2, pp. 425–458, 2020.

[23] StackOverflow, "Stack overflow data dump version from march 3, 2019," https://archive.org/download/stackexchange/.

[24] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica: Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[25] (2020) Beautifulsoup. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[26] (2020) Replication package. [Online]. Available: https://developneed.github.io/

[27] J. C. Carver, L. Jaccheri, S. Morasca, and F. Shull, "Issues in using students in empirical studies in software engineering education," in *9th IEEE International Software Metrics Symposium, METRICS 2003, September 3-5, 2003, Sydney, Australia*. IEEE Computer Society, 2003, p. 239.

[28] R. Feldt, T. Zimmermann, G. R. Bergersen, D. Falessi, A. Jedlitschka, N. Juristo, J. Münch, M. Oivo, P. Runeson, M. J. Shepperd, D. I. K. Sjøberg, and B. Turhan, "Four commentaries on the use of students and professionals in empirical software engineering experiments," *Empir. Softw. Eng.*, vol. 23, no. 6, pp. 3801–3820, 2018.

[29] M. Svahnberg, A. Aurum, and C. Wohlin, "Using students as subjects - an empirical evaluation," in *2nd International Symposium on Empirical Software Engineering and Measurement, ESEM 2008, October 9-10, 2008, Kaiserslautern, Germany*. ACM, 2008, pp. 288–290.

[30] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *36th International Conference on Software Engineering, ICSE 2014, May 31 - June 07, 2014, Hyderabad, India*, 2014, pp. 643–652.

[31] D. Ye, Z. Xing, C. Y. Foo, J. Li, and N. Kapre, "Learning to extract API mentions from informal natural language discussions," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, October 2-7, 2016, Raleigh, NC, USA*. IEEE Computer Society, 2016, pp. 389–399.

[32] J. Sun, Z. Xing, R. Chu, H. Bai, J. Wang, and X. Peng, "Know-how in programming tasks: From textual tutorials to task-oriented knowledge graph," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, September 29 - October 4, 2019, Cleveland, OH, USA*. IEEE, 2019, pp. 257–268.

[33] D. Fucci, A. Mollaalizadehbahnemiri, and W. Maalej, "On using machine learning to identify knowledge in API reference documentation," in *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, August 26-30, 2019, Tallinn, Estonia*, 2019, pp. 109–119.

[34] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *CoRR*, vol. abs/1612.03651, 2016.

[35] H. Nakayama, T. Kubo, J. Kamura, Y. Taniguchi, and X. Liang, "doccano: Text annotation tool for human," 2018, software available from https://github.com/doccano/doccano. [Online]. Available: https://github.com/doccano/doccano

[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *27th Annual Conference on Neural Information Processing Systems 2013, NIPS 2013, December 5-8, 2013, Lake Tahoe, Nevada, USA*, 2013, pp. 3111–3119.

[37] (2020) gensim. [Online]. Available: https://radimrehurek.com/gensim/

[38] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *2nd International Conference on Knowledge Discovery and Data Mining, KDD 1996, Portland, Oregon, USA*, 1996, pp. 226–231.

[39] C. D. Manning, H. Schütze, and P. Raghavan, *Introduction to information retrieval*. Cambridge university press, 2008.

[40] B. L. Welch, "The generalization of student's problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947.

[41] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *28th IEEE International Conference on Software Maintenance, ICSM 2012, September 23-28, 2012, Trento, Italy*. IEEE Computer Society, 2012, pp. 25–34.

[42] D. Wu, X. Jing, H. Chen, X. Zhu, H. Zhang, M. Zuo, L. Zi, and C. Zhu, "Automatically answering api-related questions," in *40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, May 27 - June 03, 2018, Gothenburg, Sweden*, 2018, pp. 270–271.

[43] C. Greco, T. Haden, and K. Damevski, "Stackintheflow: behavior-driven recommendation system for stack overflow posts," in *40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, May 27 - June 03, 2018, Gothenburg, Sweden*, 2018, pp. 5–8.

[44] Q. Huang, X. Xia, Z. Xing, D. Lo, and X. Wang, "API method recommendation without worrying about the task-api knowledge gap," in *33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, September 3-7, 2018, Montpellier, France*, 2018, pp. 293–304.

[45] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *26th IEEE/ACM International Conference on Automated Software Engineering ,ASE 2011, November 6-10, 2011, Lawrence, KS, USA*, 2011, pp. 323–332.

[46] T. Ye, B. Xie, Y. Zou, and X. Chen, "Interrogative-guided re-ranking for question-oriented software text retrieval," in *29th ACM/IEEE International Conference on Automated Software Engineering, ASE 2014, September 15-19, 2014, Vasteras, Sweden*, 2014, pp. 115–120.

[47] L. B. L. de Souza, E. C. Campos, and M. de Almeida Maia, "Ranking crowd knowledge to assist software development," in *22nd International Conference on Program Comprehension, ICPC 2014, June 2-3, 2014, Hyderabad, India*, 2014, pp. 72–82.

[48] E. C. Campos, L. B. L. de Souza, and M. de Almeida Maia, "Searching crowd knowledge to recommend solutions for API usage tasks," *J. Softw. Evol. Process.*, vol. 28, no. 10, pp. 863–892, 2016.

[49] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, November 11-15, 2013, Silicon Valley, CA, USA*, 2013, pp. 562–567.

[50] H. Zhang, A. Jain, G. Khandelwal, C. Kaushik, S. Ge, and W. Hu, "Bing developer assistant: improving developer productivity by recommending sample code," in *24th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2016, November 13-18, 2016, Seattle, WA, USA*. ACM, 2016, pp. 956–961.

[51] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from stack overflow," in *38th International Conference on Software Engineering, ICSE 2016, May 14-22, 2016, Austin, TX, USA*. ACM, 2016, pp. 392–403.

[52] G. Uddin and F. Khomh, "Automatic summarization of API reviews," in *32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, October 30 - November 03, 2017, Urbana, IL, USA*. IEEE Computer Society, 2017, pp. 159–170.