Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph

Hongwei Li*, Sirui Li†, Jiamou Sun†, Zhenchang Xing†, Xin Peng‡, Mingwei Liu‡, Xuejiao Zhao§

*School of Computer and Information Engineering, *Jiangxi Normal University*, Nanchang, China

†Research School of Computer Science, CECS, *Australian National University*, Canberra, Australia

‡School of Computer Science, *Fudan University*, Shanghai, China

§School of Computer Engineering, *Nanyang Technological University*, Singapore, Singapore

lihongwei@jxnu.edu.cn, {u5831882, u5871153}@anu.edu.au, zhenchang.xing@anu.edu.au, {pengxin,17212010022}@fudan.edu.cn, xjzhao@ntu.edu.sg

***Abstract*—API documentation provides important knowledge about the functionality and usage of APIs. In this paper, we focus on API caveats that developers should be aware of in order to avoid unintended use of an API. Our formative study of Stack Overflow questions suggests that API caveats are often scattered in multiple API documents, and are buried in lengthy textual descriptions. These characteristics make the API caveats less discoverable. When developers fail to notice API caveats, it is very likely to cause some unexpected programming errors. In this paper, we propose natural language processing(NLP) techniques to extract ten subcategories of API caveat sentences from API documentation and link these sentences to API entities in an API caveats knowledge graph. The API caveats knowledge graph can support information retrieval based or entity-centric search of API caveats. As a proof-of-concept, we construct an API caveats knowledge graph for Android APIs from the API documentation on the Android Developers website. We study the abundance of different subcategories of API caveats and use a sampling method to manually evaluate the quality of the API caveats knowledge graph. We also conduct a user study to validate whether and how the API caveats knowledge graph may improve the accessibility of API caveats in API documentation.**

***Index Terms*—API caveats, Knowledge Graph, Coreference Resolution, Entity Linking**

I. Introduction

API documentation, such as *Java API Documentation and Android Developers websites,* is an important resource for developers to learn API usage. They provide important information about not only the declaration of APIs but also the purpose, functionality, quality attributes, and usage directives of APIs [1], [2]. A lot of research has been done on the quality of API documentation [3]–[7] and the traceability recovery of API documentation [8]–[13]. Techniques have been developed to recommend API documentation [8], [9], [14], generate API documentation [15], [16] or enhance API documentation with crowd knowledge [17], [18].

In this paper, we are concerned with a much less explored issue of API documentation, i.e., the accessibility of API documentation. Our point is that good and well-maintained API documentation does exist, like the website resources as Java API Documentation, Android Developers. However, *can developers effectively access the relevant API usage knowledge in these documents?* According to the taxonomy of knowledge types in API documentation defined by Maalej and Robillard [1], we focus on the accessibility of "directives" in this paper. As such API usage directives "specify what users are allowed/not allowed to do with the API element" [1], overlooking them would likely to incur unexpected program behaviors or errors. To emphasize the fact that such directives are contracts, constraints, and guidelines of API usage that developers should be aware of [2], we call them as *API caveats* in this paper.

To get a sense of the potential accessibility issue of API caveats in API documentation, we conduct a formative study of 20 randomly-sampled, most-viewed (view counts in top 1%), highly up-voted (votes in top 1%) Stack Overflow questions (see the appendix table of Statistics of the 20 examined StackOverflow Questions[1]). The questions we examine have been viewed at least 22,866 times (*Why onRestoreInstate() never gets called*) and up to 943,800 times (*How do I fix android.os.networkOnMainThreadException*), which indicate that they are common issues that developers frequently encounter. For 8 out of the 20 question we examine, their answers directly quote some API caveats in API documentation, and for 16 of the 20 questions, their answers paraphrase some API caveats in API documentation. Furthermore, for 12 of the 20 questions, their answers provide the URL to the relevant API document. Although by no means conclusive, this formative study suggests that many programming issues that affect millions of developers could be avoided if developers were aware of relevant API caveats in API documentation.

We cannot rule out the factor that developers fail to read API documentation carefully enough, but earlier studies reveal that documentation fragmentation does make the information less discoverable [7]. Our formative study shares the similar observation: we observe that 80% questions involve API caveats of more than one API and 50% questions involve API caveats from multiple documents. Furthermore, although it often looks very clear when an API caveat is quoted in an answer to a programming-issue question, the API caveat may

[1]The Table can be found at https://github.com/Text2KnowledgeGraph/data/blob/master/Statistics%20of%20the%2020%20examined%20StackOverflow%20Questions.pdf

not be noticeable in a lengthy description of various types of API knowledge in API documentation [1], [19], [20]. For the API caveats causing programming issues in the 20 examined questions, they are buried in documents from 1,412 to 73,269 words. That is, the post mortem discovery of an API caveat in a Q&A website after something wrong happened is much easier than bewaring of the API caveat beforehand to avoid the mistakes in the first place.

In this paper, we tackle the accessibility issue of API caveats revealed in our formative study by mining an API caveats knowledge graph from multiple sources of API documentation. We construct an API skeleton graph of APIs and regard an API (e.g., class, method, field) as an entity (node) and their declared relations (edges) by parsing semi-structured API reference documentation (e.g., Android API reference). We define three categories of API caveats (i.e., explicit, restricted, generic) based on the literature survey [2], [21]–[25] and our own observation of API caveats. We develop corresponding sentence syntactic patterns to extract API caveat sentences from the textual description of APIs. We apply co-reference resolution technique [26]–[29] and declaration-based heuristic to resolve the pronouns in these sentences to the corresponding APIs. We develop hyperlink-based, declaration-based and open linking methods to link API caveat sentences with API entities. As a result, we obtain an API caveats knowledge graph. Based on this knowledge graph, we can recommend a list of API caveats for the API(s) of interest (e.g., mentioned in a search query, discussed in a web page, or used in code).

We construct a proof-of-concept API caveats knowledge graph from the API documentation on the Android Developers website. The resulting knowledge graph contains 175,538 API entities and 160,112 unique API caveat sentences. Our abundance analysis of API caveats suggests that most of the API caveats (about 78%) are generic sentences without explicit caveat indicators. This could explain why API caveats are hard to notice in API documentation. Our manual examination of the resulting knowledge graph confirms the accuracy of extracting API caveat sentences, resolving co-references in API caveat sentences, and linking API caveat sentences to APIs. Our user study shows that searching API caveats in an API caveats knowledge graph can significantly improve the accessibility of API caveats, compared with directly searching API caveats in API documentation.

This paper makes the following contributions:

- We conduct a formative study of Stack Overflow questions to investigate the accessibility issue of API caveats in API documentation.
- We empirically develop a taxonomy of API caveats in API documentation, and develop an NLP approach to construct an API caveats knowledge graph from API documentation. The resulting knowledge graph can support entity-centric and Information Retrieval (IR) based search of API caveats.
- We construct a proof-of-concept API caveats knowledge graph from the API documentation on the Android Developers website and conduct a series of experiments to evaluate the quality of the resulting knowledge graph and its ability to improve the accessibility of API caveats.

II. Formative Study

In this formative study, the first author reads a set of Stack Overflow questions, identifies the programming issues in the questions, summarizes the solutions in the answers, and determines whether the solutions are covered by some API caveats in API documentation. The second author validates the analysis results by the first author, and the two authors discuss and come to an agreement on the final results.

As we are concerned with programming issues and their root causes, we particularly search for questions whose title is a negative sentence (e.g., not work, not being called) and/or contains error-indicating terms such as "error", "exception", "fail", "fix". We limit our search to questions tagged with *Android* as our current proof-of-concept implementation targets at Android APIs. We sort the questions by their view counts and votes, and randomly select 20 questions that have view counts in top 1% and votes in top 1%. We select high-view-count and high-vote questions because this study is about API caveats that may affect a large number of developers, rather than some unique programming issues.

For each selected question, we read all its answers with positive votes, as well as the comments on the question and the answers, to summarize the solutions for the programming issue(s) in the question. We then try to determine whether the solutions are due to some overlooked API caveats in API documentation. This can be easily determined if the answer directly quotes some API documents and/or references to relevant API documents. If such explicit indicators are not present, we identify the APIs mentioned in the Q&A discussion and read relevant API documents to make a decision.

Table I and Table II show the two examples of our analysis results. In Table I, the question we examine is "SharedPreferences.onSharedPreference-ChangeListener not being called". The programming issue is "the listener is not always called". The answer suggests to "keep a strong reference to the listener" and this solution is actually an API caveat explicitly explained in the reference document of SharedPreferences. Table II shows a more complex problem about "FileUriExposedException: ... exposed beyond app through Intent.getData()". This causes "The app crashes when opening a file from the SD card" and "The app cannot open files in root directories". The answers suggest several solutions such as "Replace file:// URI with content:// URI" or using FileProvider API. Again, the programming issues are caused by overlooking some API caveats and the solutions are just paraphrases of these API caveats.

As summarized in the appendix table of Statistics of the 20 examined StackOverflow Questions[1], the 20 examined questions cover a wide range of programming tasks and issues. These questions have been viewed in total 4,170,612 times and have received in total 6,296 votes. This reveals the developers' common interests in the API knowledge in these questions. We

TABLE I
EXAMPLE QUESTION 1

Question	**SharedPreferences.onSharedPreferenceChangeListener not being called consistently**
Key Issues	The listener is not always called.
Solutions	Keep a strong reference to the listener.
API Caveats	Caution: ... You must store a strong reference to the listener, or it will be susceptible to garbage collection.

TABLE II
EXAMPLE QUESTION 2

Question	**android.os.FileUriExposedException: file:///storage/emulated/0/test.txt exposed beyond app through Intent.getData()**
Key Issues	1. The app crashes when opening a file from the SD card. 2. The app cannot open files in root directories.
Solutions	1. Replace file:// URI with content:// URI. 2. Use FileProvider class to make files accessible. 3. Create class inheriting FileProvider to avoid conflict.
API Caveats	1. The exception that is thrown when an application exposes a file:// Uri to another app. 2. Instead, apps should use content:// Uris so the platform can extend temporary permission for the receiving app to access the resource. 3. FileProvider ... facilitates secure sharing of files associated with an app by creating a content:// Uri for a file instead of a file:// Uri. 4. If you want to override any of the default behavior of FileProvider methods, extend the FileProvider class and use the fully-qualified class name ...

identify 1-5 solutions for the programming issue in each question. According to our analysis, the solutions for 11 questions are fully covered by the API caveats in the documentation of APIs discussed in the questions and answers, and the solutions of the other 5 questions are partially covered by the relevant API caveats. Only for 4 questions (Id: 14, 15, 18, 19), we do not identify any API caveats in relevant documentation. Among the 20 examined questions, 40% contain direct quote (DQ) of some API caveats from API documentation, 60% provide the URLs of relevant API documents (ER), and 80% contains the paraphrases of some API caveats (CP). These statistics suggest that many programming issues that affect millions of developers could be avoided if developers were aware of relevant API caveats in API documentation.

To understand the challenges for developers to notice API caveats in API documentation, we further examine the number of APIs involved (#API) in the programming issue(s) of each question, the number of documents (#Doc) that contain API caveats relevant to the programming issue(s), and the number of words (#Words) in these documents. We find that 80% questions involve API caveats of more than one API and 50% involves API caveats from two or more documents. Furthermore, for 30%, 30% and 40% questions, relevant API caveats are in documents with less than 5,000 words, 5,000 to 20,000 words, and over 20,000 words, respectively. Documentation fragmentation and the lengthy description in API documentation could become the barrier for developers to notice important caveats beforehand to avoid the programming issues. Our observation is consistent with some API learning obstacles identified in the survey of developers [7].

Summary: Our formative study shows that many programming issues could actually be avoided if developers were aware of relevant API caveats in API documentation. Unfortunately, developers mostly discover API caveats post mortem after something wrong happened, rather than bewaring of the API caveats beforehand to avoid the mistakes in the first place.

III. RELATED WORK

There has been much research on the quality of API documentation [4]–[6], [9], [30], [31], which investigates the issues like the absence, incompleteness, and staleness of API documents. Many techniques have been proposed to address these issues, for example, by automatically generating API documents or keeping the documents up-to-date [8], [15], [32]. As the online Q&A websites become popular, they provide an alternative way of documentation, i.e., crowd documentation [17], [18], [33]. Studies [34]–[36] show that crowd documentation is a good complement to the traditional API documentation.

In addition to the quality of API documentation, another well-studied aspect is the traceability of API documents. For example, Bacchelli et al. [37], [38] develop an API extraction and linking infrastructure, called Miler. Dagenais and Robillard [10] develop RecoDoc to extract Java APIs from several learning resources (formal API documentation, tutorial, forum posts, code snippets) and then perform traceability link recovery across different sources. Subramanian et al. [8] use code context information to link an API mention in a partial code fragment to APIs in a knowledge base. Ye et al. [39] propose mention-mention and mention-entity similarity metrics for linking API mentions in natural language sentences to API entities. Their work inspires our open linking method to link API caveat sentences to APIs. These existing works only recover traceability links, but our approach organizes API entities, their declared relations, and associated API caveats in a knowledge graph.

Compared with the studies on the quality and traceability of API documentation, the accessibility of API documentation is much less explored. Some recent works [19], [40] aim at fine-grained information retrieval at passage or sentence-level, which could improve the accessibility of API knowledge in lengthy documents. Some NLP techniques used in these works, such as sentence type identification, pronoun resolution, API mention discovery, provide inspirations for the design of our method. But different from these fine-grained text retrieval methods, our method supports the search of API caveats based on the API caveats knowledge graph.

The works that are most close to ours are the two empirical studies on the knowledge types in API documentation [1], [2]. Our definition of API caveats is inspired by these two studies. The development of syntactic patterns for extracting API caveats mainly absorbs and extends the API-directives patterns in [2]. However, these two studies focus on empirical observations of API knowledge in API documentation, with no specific application objectives. In contrast, our work proposes

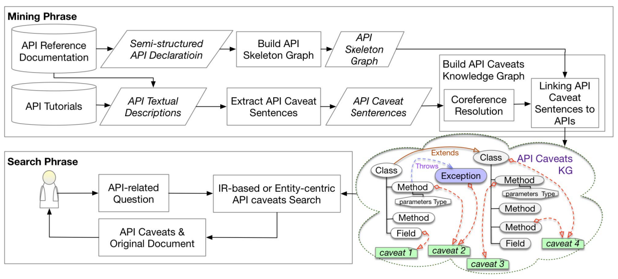


Fig. 1. An Overview of Our Approach

practical methods to extract API caveats from API documentation and organize the extracted knowledge in a knowledge graph for improving the accessibility of API caveats.

IV. The Approach

Fig. 1 presents the key steps in our approach, which contains two main parts: mining API-caveats knowledge graph from API documentation and searching API caveats based on the mined knowledge graph. Given a set of API documentation, the mining process consists of four steps: preprocess input documentation (Section IV-A), build an API skeleton graph from semi-structured API reference documentation (Section IV-B), extract API caveat sentences from API textual descriptions (Section IV-C), and construct the API-caveats knowledge graph by linking API caveat sentences to relevant APIs (Section IV-D). Based on the mined API-caveats knowledge graph, the search engine can recommend API caveats for the API(s) of interests (Section IV-E).

A. Input and Preprocessing

In this work, we consider two types of API documentation: *API reference documentation* and *API tutorials*. API reference documentation, such as Android API reference and Java SDK API specification provide a semi-structured declaration of APIs and explain the purpose, functionality, and caveats of APIs. API tutorials, such as Android Developer Guides , Java Tutorials explain and demonstrate how to use an API in different tasks.

We crawled online API documentation using the web crawling tool such as BeautifulSoup [2]. We consider each crawled web page as an API document. As we are interested in the semi-structured API declarations and the API textual descriptions, we remove other document contents from the crawled web pages, for example, code snippets, program execution outputs, images. We observe that such document contents in official API documentation are usually contained. Such document contents are contained in HTML tags, such as <code>, <image> and <script>, which can be easily identified and removed.

API textual descriptions have to be tokenized for further natural language processing(NLP). API tokens are usually out of natural language vocabulary and contain special characters such as ".", "()", "[]", "_". For example, the method declaration sentence "void setOnBufferAvailableListener (Allocation.OnBufferAvailableListener callback) Set a notification handler for USAGE_IO_INPUT" contains the method signature "void setOnBufferAvailableListener (Allocation.OnBufferAvailableListener callback)" and the field name "USAGE_IO_INPUT". A general English tokenizer will break API tokens into several tokens, such as "USAGE", "_", "TO", "_", and "INPUT". This breaks normal sentence integrity and will negatively affect the subsequent NLP steps. Therefore, we expand the software-specific tokenizer developed by Ye et al. [41] for extracting API mentions in natural language sentences. Our software-specific tokenizer will retain the integrity of API tokens during text tokenization. After tokenization, we use Stanford CoreNLP[3] to split texts into sentences.

B. Building API Skeleton Graph

We first build an API skeleton graph from API reference documentation. API reference documentation is semi-structured, and they organize APIs into different sections. They also provide the full-qualified name of APIs. Relations between APIs, such as inheritance, data type reference, thrown exception are hyperlinked by the relevant API's URL. Our approach exploits such semi-structured information in API reference documentation to extract API entities and declared relations between APIs. In this work, we consider classes, interfaces, fields, methods, and parameters. Relations include containment, inheritance/implementation, field data type, method return type, method parameter type, and method-thrown-exception. We build an API skeleton graph using the extracted API entities and their relations. Each API entity in the graph is identified by its fully qualified name. Each API entity can also be identified by its unique URL in API reference documentation. This helps to identify API entities that are hyperlinked in textual descriptions.

C. Extracting API Caveat Sentences

An API caveat is a sentence that specifies some constraint or guideline of API usage [2]. We define a taxonomy of API caveats and develop corresponding sentence syntactic patterns to extract API caveat sentences from API textual descriptions. The taxonomy and syntactic patterns have been developed based on the literature survey [2], [21]–[25] and our own observation of API caveats in API documentation.

As summarized in Table III, our taxonomy contains three general categories, each of which has some subcategories: **explicit** - error/exception, recommendation, alternative, imperative, note; **restricted** - conditional, temporal; and **generic** - affirmative, negative, emphasis. Each subcategory has some distinctive syntactic patterns. A syntactic pattern is defined as a regular expression of some tokens. When a sentence matches a pattern, the sentence will be categorized as the corresponding subcategory. A matching is case-insensitive. Following [2], we use word stem to match different infected variants of a word, for example "assum*" for "assume", "assumes", "assuming".

[2]https://www.crummy.com/software/BeautifulSoup/

[3]https://stanfordnlp.github.io/CoreNLP

TABLE III
API-CAVEATS CATEGORIES AND SYNTACTICS PATTERNS

Category	Subcategory	Syntactic Pattern Examples
Explicit	Error/Exception	"insecure", "susceptible", "error", "null", "exception", "susceptible", "unavailable", "not thread safe", "illegal", "inappropriate", "insecure"
	Recommendation	"deprecate", "better/best to", "recommended", "less desirable" "discourage"
	Alternative	"instead of","rather than","otherwise"
	Imperative	"do not"
	Note	"note that", "notably", "caution"
Restricted	Conditional	"under the condition", "whether ...", "if ...", "when ...", "assume that ..."
	Temporal	"before", "after"
Generic	Affirmative	"must", "should", "have to", "need to"
	Negative	"do/be not ...", "never"
	Emphasis	"none", "only", "always"

As a sentence may match several patterns at the same time, the taxonomy is not exclusive. That is, an API caveat sentence may belong to several subcategories.

Next, we explain each subcategory as follows: name, description, syntactic patterns, and typical examples. In the examples, we highlight in bold font the matching syntactic pattern for the subcategory under discussion and underline the syntactic patterns for other subcategories.

1) Explicit API Caveats: **Error/exception** caveats explicitly mention programming errors or unwanted behaviors. Syntactic patterns match error-indicating terms that are commonly used to describe programming errors, such as "error", "exception", "null", "susceptible", "unavailable", "not thread safe", "illegal", "insecure". Typical examples include: "**FileUriExposedException** is thrown when an application exposes a file://Uri to another app", "Camera class is **not thread safe** ...", "You must store a strong reference to the listener, otherwise it will be **susceptible** to garbage collection".

Recommendation caveats explicitly recommend what to do or what not to do. Syntactic patterns match terms/phrases such as "deprecate", "better/best to", "highly recommended", "discourage", "less desirable". Typical examples include: "you are **better off** using JobIntentService, which uses jobs instead of services ...", "If ..., it is **highly recommended** you use the various APIs provided by the java.util.concurrent package ...", "This exposure is **discouraged** since the receiving app may not have access to the shared path".

Alternative caveats explicitly mention an alternative or substitute. Syntactic patterns match phrases such as "instead of", "rather than", "otherwise". Typical examples include: "You must call release() when you are done using the camera, **otherwise** it will remain locked and be unavailable to applications".

Imperative caveats explicitly tell developers not to do something. Syntactic patterns match imperative expressions "do not". Typical examples include: "**Do not** pass a resource ID", "**Do not** confuse this method with activity lifecycle callbacks such as onPause()...".

Note caveats explicitly point out some information that developers should pay attention to. Syntactic patterns match terms like "note that", "notably", "caution". Typical examples include: "**Note**: For all activities, you must declare your intent filters in the manifest file", "**Caution**: On some devices, this method may take a long time to complete. It is best to ...".

2) Restricted API Caveats: **Conditional** caveats identify some specific conditions or circumstances for using an API. Syntactic patterns match conditional clause such as "if ...", "when ...". Typical examples include: "**if** everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI", "**When** using a subclass of AsyncTask to run network operations, you must be cautious ...".

Temporal: API caveats in this subcategory identify the order of some operations. Syntactic patterns match temporal words such as "before", "after". Typical examples include: "This may be null if the service is being restarted **after** its process has gone away", "... you don't create a memory leak ... **before** the AsyncTask finishes its background work".

3) Generic API Caveats: **Affirmative** caveats indicate something that developers must or should do. Syntactic patterns match terms/phrases like "must", "should", "have to", "need to". Typical examples include: "Note that not all Typeface families actually have bold and italic variants, so you may **need to** use setTypeface(Typeface, int) ...", "The identifier does not have to be unique in View, but it **should** be positive".

Negative caveats indicate something that developers should avoid or that an API does not do. Syntactic patterns match negative expressions such as "do/be not ...", "never". We detect negative expressions by sentence dependency parsing and then identifying a syntactic role of negation. Typical examples include: "StrictMode **is not** a security mechanism and **is not guaranteed** to find all disk or network accesses." "Any activities that **are not** declared there **will not be** seen by the system and will **never be** run".

Emphasis caveats emphasize some particular conditions or operations. Syntactic patterns match qualifier words such as "none", "only", "always". Typical examples include: "**Only** objects running on the UI thread have access to other objects on that thread", "if you do not declare any intent filter for an activity, then it can be started **only** with an explicit intent".

D. Building API Caveats Knowledge Graph

Given the API skeleton graph and the API caveat sentences, the last step of the mining process is to link the API caveat sentences to relevant APIs in the API skeleton graph. As a result, we obtain an API caveats knowledge graph.

1) Co-reference Resolution: APIs are not always mentioned by their API names in API caveat sentences, because developers commonly use pronouns to represent APIs in a paragraph of explanations. For example, the description of the "android.app.Activity.startActivity" states that "*This startActivity* *method implementation overrides the base version, ... It throws ActivityNotFoundException if there was no Activity found to run the given Intent*." According to our API-caveat-sentence syntactic patterns, the second sentence will be extracted as an API caveat sentence that belongs to conditional and explicit-exception categories. The pronoun "it" of the second sentence refers to the "startActivity" method in the first sentence.

Due to the wide presence of such co-references in API caveat sentences, linking only explicit API mentions in API caveat sentences to API entities would miss many important links between API caveat sentences and APIs. For example, we can link the above caveat sentence to "ActivityNotFoundException" and "Activity" based on the explicit API mentions in the sentence, but we cannot link this sentence to the method "startActivity" that throws the exception. To recover as many links between API caveat sentences and APIs as possible, we use co-reference resolution technique (as implemented by Stanford CoreNLP[4]) to resolve the pronouns in API caveat sentences to the APIs that the pronouns represent in the paragraphs from which the caveat sentences are extracted.

Furthermore, when explaining an API in API reference documentation, it is a common practice to refer to the API being explained as "this class", "this method", etc. For example, under the declaration section of "Activity.onActionModeStarted", the description states that "*Activity subclasses overriding this method should call the superclass implementation. If you override this method you must call through to the superclass implementation*." Co-reference resolution tools cannot resolve this type of co-reference because the corresponding API does not appear in the surrounding texts. However, we can replace the co-references like "this method" with the name of the API declared in the corresponding API section. We refer to this method as declaration-based co-reference resolution.

2) Linking API Caveat Sentences to API Entities: We distinguish three linking scenarios: hyperlink based, declaration based, and open linking.

Hyperlink based: If an API caveat sentence contains a hyperlink to the URL of an API reference document, our approach will identify the API at the URL and link the sentence to the corresponding API. For example, as the "ActivityNotFoundException" in the caveat sentence "It throws ActivityNotFoundException if ..." is hyperlinked to the reference page of "ActivityNotFoundException", we can then link this sentence to the "ActivityNotFoundException".

Declaration based: If an API caveat sentence is from API reference documentation and it mentions the name (could be simple or qualified) of the API declared in the API section from which the sentence is extracted, our approach will link the caveat sentence to the corresponding API. Sometimes, an API caveat sentence from the API section may not explicitly mention any APIs. For example, from the declaration section of the method "View.setId(int)", our approach extracts two API caveat sentences "Do not pass a resource ID" and "The identifier should be a positive number". Although these sentences do not explicitly mention "View.setId(int)", it is intuitive to link such caveat sentences to the corresponding APIs.

Open linking: First, our method uses Open Information Extraction (OpenIE) software[5] to extract *Subject-Verb-Object* (SVO) triples from the API caveat sentences. For example, given the API caveat sentence "You must call release() ...", OpenIE extracts the subject "You", verb phrase "must call", and object "release()". We use OpenIE because it is the best performing tool to extract SVOs for a similar task to ours, i.e., building a task knowledge graph [42].

Next, our method attempts to link the Subject and Object of the SVO triples to some APIs, and this essentially links the corresponding API caveat sentences to relevant APIs. First, if a Subject (or Object) matches an API name in the knowledge graph (e.g., "release()" in the above SVO example), we consider the Subject (or Object) as a candidate API mention and the name-matching API as a candidate API. Then, following the API linking method by Ye et al. [39], we adopt mention-mention similarity and mention-API similarity to link a candidate API mention to an API.

For a candidate API mention, mention-mention similarity examines the API document from which the API caveat sentence is extracted and checks if there are some same API mentions that can be linked to an API using the hyperlink- or declaration-based method. If so, the candidate API mention will be linked the same API. The underlying intuition is that the same API mentions in an API document should refer to the same API. For example, the hyperlinked mention of "ActivityNotFoundException" will help to link other non-hyperlinked mentions of "ActivityNotFoundException" in the same API document to the class "ActivityNotFoundException".

If mention-mention similarity cannot link the candidate API mention to an API, we then use mention-API similarity, which measures the textual relevance between the paragraph from which the API caveat sentence is extracted and the description of a candidate API whose name matches the Subject (or Object). If multiple candidate APIs exist, the one with the highest relevance score is selected. If the mention-API similarity of the selected API is above the user-defined threshold, the candidate API mention is linked to this API.

Finally, we link the API caveat sentences that are not linked to any APIs by the three linking methods to an artificial API entity as a general corpus of API caveats. That is, all extracted caveat sentences are accessible in the knowledge graph, even they may not be linked to specific APIs.

E. Searching for API Caveats in the Knowledge Graph

Traditionally, developers need to first find relevant API documents and then read through them to find the API caveats. Our API caveats knowledge graph enables a different information seeking paradigm, in which developers can quickly find the API caveats relevant to the programming issue they have, and then learn more about these caveats in the documents.

Our approach supports two modes of searching for API caveats. First, we consider all API caveat sentences as a

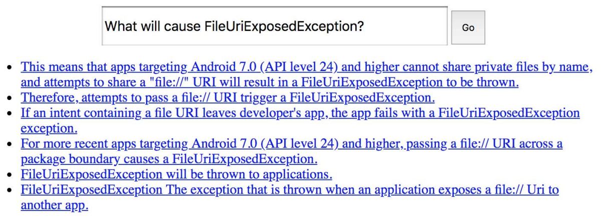
[4]https://stanfordnlp.github.io/CoreNLP/coref.html

[5]https://nlp.stanford.edu/software/openie.html

text corpus, and use traditional IR techniques to search this corpus of API caveats given an input query. Second, we perform entity-centric search of API caveats for the API(s) that developers are interested in, based on the API caveats knowledge graph. The API(s) of interest can come from a search query of some programming issues, a webpage that a developer is reading, or some code that the developer is writing. For example, given a search query "onActivityResult is not being called in Fragment", entity-centric search first finds the API(s) mentioned in the query ("onActivityResult" and "Fragment" in this example) by matching query terms with the API names in the knowledge graph. It may also collect the neighboring APIs of the mentioned API through the declared relations between APIs in the knowledge graph, for example, the declaring class "Activity" of the method "onActivityResult". It then measures the relevance of the search query (or the webpage, the code) and the API caveats of the collected APIs. Finally, it ranks and returns the top-N relevant API caveats to the developer.

V. Proof-of-Concept Implementation

We implement a proof-of-concept tool of our approach using the API documentation from the Android Developers website. We crawled 11,352 web pages from this website, including not only 6,042 API reference documentation, but also 5,310 other types of API tutorials such as "Training", "API Guide", "Samples", "Topics". The crawled web pages contain a huge volume of API textual descriptions (about 600,000 sentences and over 7.2 million words after text preprocessing). Using this API documentation, we build an API caveats knowledge graph for Android APIs. The resulting knowledge graph contains 175,538 API entities, 160,112 unique API caveat sentences, and about 1.1 million links between API caveat sentences and APIs (see Section VI-A for details).


What will cause FileUriExposedException?

Go

- This means that apps targeting Android 7.0 (API level 24) and higher cannot share private files by name, and attempts to share a "file://" URI will result in a FileUriExposedException to be thrown.
- Therefore, attempts to pass a file:// URI trigger a FileUriExposedException.
- If an intent containing a file URI leaves developer's app, the app fails with a FileUriExposedException exception.
- For more recent apps targeting Android 7.0 (API level 24) and higher, passing a file:// URI across a package boundary causes a FileUriExposedException.
- FileUriExposedException will be thrown to applications.
- FileUriExposedException The exception that is thrown when an application exposes a file:// Uri to another app.

Fig. 2. The Search UI of the API Caveats Knowledge Graph

We implement a web interface for searching API caveats in our knowledge graph (see Fig. 2). Developers can enter a search query of some API-related question, for example, "what will cause FileURLExposedException". The application searches the backend knowledge graph and returns a ranked list of relevant API caveats. For example, the returned API caveats in Fig. 2 reveal the root causes for FileURLExposedException. Developers can click an API caveat to view it in its original API document. Note that the six returned API caveats roughly describe the same thing, and they are scattered in 5 documents with over 50,000 words. Being able to directly finding API caveats can improve the accessibility of API caveats in API documentation, without the need to read through the fragmented and lengthy API documentation.

VI. Evaluation

Our approach extracts and organizes various categories of API caveats that are scattered in a large set of API documentation into an API caveats knowledge graph. We report our experiments to answer the three research questions about the effectiveness and usefulness of our approach:

- RQ1: What is the abundance of different subcategories of API caveats in API documentation?
- RQ2: Can our approach accurately extract API caveat sentences, resolve co-references in these sentences, and link API caveat sentences to API entities?
- RQ3: Can our API caveats knowledge graph and API caveats search improve the accessibility of API caveats, compared with traditional documentation search?

A. Android API Caveats Knowledge Graph

In our experiments, we use our proof-of-concept Android API caveats knowledge graph. This knowledge graph contains 175,538 API entities, including 10,916 classes / Interfaces / Exceptions / Enums / Annotations, 28,167 fields, 46,672 methods and 89,783 parameters. Our approach extracts 160,122 unique API caveat sentences. As one API caveat sentence may belong to two or more subcategories, we have in total 267,891 times of sentences by the 10 subcategories of API caveats (see Table III). 91,613 (57.2%) out of the 160,122 unique API caveat sentences have been linked to some APIs in the knowledge graph. In total, the knowledge graph has about 1.1 million links between API caveat sentences and APIs, including 505,725 hyperlink-based, 282,419 declaration based, and 306,500 open-linking links.

B. The Abundance of API Caveats (RQ1)

Motivation: our approach extracts a large number of API caveat sentences of ten subcategories. We first would like to get a good understanding of the abundance of different categories of API caveats. This will shed the light on the potential accessibility issue of API caveats in API documentation.

Approach: We count the number of each subcategory of API caveats. Inspired by the study of different kinds of API directives [2], we adopt the ACFOR scale (Abundant, Common, Frequent, Occasional, Rare)[6]. This scale is originated from ecology for measuring species abundance within a given area [43]. The ACFOR scale is determined as follows. First, we compute the average of abundance frequency A. We say that a subcategory of API caveats is abundant if it appears more than $2A$, common if its frequency $F \geq 1.5A$, frequent if $F \geq A$, occasional if $F \geq 0.5A$ and rare otherwise.

Results: Table IV summarizes the number of API caveat sentences for each subcategory, the abundance frequency of each subcategory, and the ACFOR analysis results. The average abundance frequency is 0.1. As such, restricted-conditional and generic-affirmative caveats are abundance in

[6]http://en.wikipedia.org/wiki/Abundance_(ecology).

Android API documentation, generic-emphasis are common, and generic-negative are frequent. These four subcategories account for 78.2% of all API caveats. Explicit-recommendation is occasional and explicit-error/exception are close to occasional. These two subcategories account for 11.7% of all API caveats. The rest four subcategories (explicit-alternative, explicit-imperative, explicit-note and restricted-temporal) are rare, and they account for 10.1% of all API caveats.

Restricted-conditional and the three generic API caveats are dominant in our knowledge graph. The fact that most of API caveats do not have explicit caveat indicators could help to explain why API caveats are hard to notice in the API descriptions, especially the lengthy ones.

C. The Quality of API Caveats Knowledge Graph (RQ2)

Motivation: Three steps in the mining process of our approach affect the quality of the resulting API caveats knowledge graph. They are: extracting API caveat sentences by syntactic patterns, co-reference resolution for API caveat sentences, and linking API caveat sentences to APIs. We want to confirm the accuracy of these three steps to build the confidence in the quality of the resulting API caveats knowledge graph.

Approach: As we have a large number of API caveat sentences to examine, we adopt a sampling method [44]. According to [44], we examine the minimum number MIN of data instances in order to ensure that the estimated population is in a certain confidence interval at a certain confidence level. This MIN can be determined by the formula: $MIN = n_0/(1 + (n_0 - 1)/populationsize)$. n_0 depends on the selected confidence level and the desired error margin: $n_0 = (Z^2 * 0.25)/e^2$, where Z is a confidence level's z-score and e is the error margin. For each mining step, we examine MIN instances of relevant data for the error margin $e = 0.05$ at 95% confidence level. For each sampled API caveat sentence, the two authors first independently evaluate its accuracy (binary decision) for a respective mining step. Then, we compute Cohen's Kappa [45] to evaluate the inter-rater agreement. For the API caveat sentences that the two authors disagree, they have to discuss and come to a final decision. Based on the final decisions, we compute the accuracy of each respective mining step.

Results: Next, we report the data sampling and accuracy of analysis results for the three mining steps respectively.

1) Accuracy of Extracting API Caveat Sentences: We perform accuracy analysis for each subcategory of API caveats. Table V summarizes our analysis results. The column $\#MIN$ is the number of API caveat sentences we randomly sample and examine for each subcategory. This number is determined based on the number of API caveat sentences in each subcategory and the above sampling formula. The annotator determines if a sampled sentence is actually an API caveat or not. The columns $AA1$ and $AA2$ show the accuracy results determined by the two annotators independently, and the column AF is the final accuracy for each subcategory after resolving the disagreement. The column AC is the average accuracy of the subcategories of a general category (i.e., explicit, restricted, or generic).

TABLE IV
THE ABUNDANCE OF DIFFERENT SUBCATEGORIES OF API CAVEATS

Category	Subcategory	#N	#F	AAF	ACFOR
Explicit	Error/Exception	11,973	0.045	0.100	R
	Recommendation	19,209	0.072		O
	Alternative	4,032	0.015		R
	Imperative	6,410	0.024		R
	Note	8,183	0.031		R
Restricted	Conditional	70,404	0.263		A
	Temporal	8,479	0.032		R
Generic	Affirmative	61,952	0.231		A
	Negative	36,136	0.135		F
	Emphasis	41,113	0.153		C
Total:		267,891	1.000		

#N: Numbers; #F: Frequency; AAF: Average Abundance Frequency; ACFOR: Abundant, Common, Frequent, Occasional, Rare

TABLE V
ACCURACY RESULTS OF EXTRACTING API CAVEAT SENTENCES

Category	Subcategory	MIN	AA1	AA2	AF	AC
Explicit	Error/Exception	373	99.73	100	99.73	99.89
	Recommendation	377	100	100	100	
	Alternative	351	100	99.61	100	
	Imperative	363	99.72	97.52	99.72	
	Note	367	100	100	100	
Restricted	Condition	383	100	100	100	100
	Temporal	368	100	97.83	100	
Generic	Affirmative	382	99.48	99.74	98.95	99.65
	Emphasis	381	100	100	100	
	Negative	381	100	93.96	100	

The Cohen's kappa metric for each subcategory of API caveats between the two annotators is all > 0.88, which indicate almost perfect agreement between the accuracy decisions of the two annotators. The lowest final accuracy after resolving the disagreement is 98.95 for alternative caveats. The final accuracy of six subcategories is 100. This high accuracy is not surprising as the API caveat sentences are extracted using carefully designed caveat-indicating syntactic patterns.

2) Accuracy of Co-reference Resolution: Based on the number of API caveat sentences that require co-reference resolution and the above sampling formula (see VI-C), we randomly sample and examine 384 API caveat sentences. The annotator determines if the co-reference in the sentence has been correctly resolved to the corresponding API. The Cohen's kappa metric between the two annotators' decisions is 0.97 which indicates almost perfect agreement. After resolving the disagreements, the two annotators determine that the co-references in 285 sentences have been correctly resolved, i.e., co-reference resolution accuracy is 285/384=74.22%. Among these 285 sentences, 215 (75.44%) has been resolved by the CoreNLP tool and the rest 70 (24.56%) has been resolved by our declaration-based heuristic.

3) Accuracy of Caveat-Sentence-API Linking: As the hyperlink-based linking is always accurate, we exclude them from this analysis. Based on the number of declaration-based linking and open linking instances and the above sampling formula, we randomly sample and examine 384 and 384 caveat-

TABLE VI
TEN ANDROID API RELATED QUESTIONS IN OUR USER STUDY

Questions	Relevant API(s)	#API caveats
1. What should I do to prevent registering the receiver multiple times when I register a receiver in onResume?	2	2
2. What will cause FileUriExposedException?	1	4
3. getColor method was deprecated. What API should I use to replace this method?	1	2
4. Is StrictMode a secure mechanism? Why?	1	1
5. To set the dialog cancellable, should I use Dialog.setCancelable or DialogFragment.setCancelable? Why?	2	1
6. When should onPause and onStop be called?	4	3
7. Why the Intent in startService is null?	1	1
8. Is there any alternative background thread management tools I can use besides AsyncTask?	3	2
9. What will happen if I don't call release after using the Camera? And How can I release it?	2	3
10. What can getParentFragment method do from within a nested fragment?	1	1

sentence-API links for the two linking scenarios respectively. The annotator determines if the link correctly associates an API caveat sentence to the corresponding API. The Cohen's kappa metric between the two annotators' decisions is 99.74% for the declaration-based linking and 98.70% for open linking, respectively, which indicates almost perfect agreement. For declaration-based linking, the final accuracy is 99.48%, and for open linking, the final accuracy is 98.44%.

All the three knowledge graph construction steps are accurate. This ensures the high quality of the resulting API caveats knowledge graph.

D. The Improvement of API Caveats Accessibility (RQ3)

Motivation: The goal of our approach is to improve the accessibility of API caveats by mining API caveat sentences and constructing an API caveats knowledge graph. As such, developers can directly search for API caveats, without the need to read through the lengthy API documentation. We want to evaluate how well we achieve this goal.

Approach: We conduct a user study to compare the effectiveness of searching for API caveats using a document-based search method and our knowledge-graph based search method.

Subject questions: We use Android APIs and their caveats as the subject to search. Based on the 20 Stack Overflow questions examined in the formative study, the two authors collaboratively formulate 10 Android API related questions. For each question, the two authors also identify relevant API(s) and API caveats as the ground-truth answer to the question, based on the upvoted answers to the corresponding Stack Overflow questions and the relevant API documentation. The two authors have to discuss and come to an agreement for the answers. Table VI lists the formulated questions, the relevant API(s) for each question, and the number of API caveats in the ground-truth answer to each question[7].

[7]The ground-truth answers can be found at https://github.com/Text2KnowledgeGraph/data/blob/master/Ten%20Questions.pdf

TABLE VII
PERFORMANCE OF CONTROL GROUP VERSUS EXPERIMENTAL GROUP

	Participants	AveQCT(second)	AveCP(%)
Control Group	P1	170.8	55.83
	P2	132.3	32.5
	P3	143.9	37.5
	P4	63.4	15.83
	P5	214.4	49.83
	P6	129.2	54.16
	Ave±stddev	142.33±50.02	40.94±15.42
Experimental Group	P7	94.3	57.5
	P8	70.9	54.16
	P9	89	73.33
	P10	76.2	65.82
	P11	124.6	65
	P12	61.8	61.66
	Ave±stddev	86.13±22.26	62.91±6.76

AveQCT: Average Question-Completion-Time
AveCP: Average Correct Percentage

Participants: We recruit 12 third- and fourth-year undergraduate students from our school. These students have the similar course-taking history and academic records. None of them have Android development experience. We randomly split these 12 students into two groups: a control group and an experimental group. Each group has 6 students. The control group uses Google search engine to search for API caveats on the Android Developers website. The experimental group uses our proof-of-concept search tool to search for API caveats in the Android API caveats knowledge graph we construct.

Experiment Procedure: We develop a simple application for the participants to read the experiment questions and enter their answers. The application displays one question at a time and it records the time when a participant starts a question and the time when he/she submits the answer. Each question is given up-to 5 minutes. The application will automatically save the answer and move to the next question when the time is up. After each question, the application asks the participants to rate the difficulty of the question and their confidence in the submitted answer using the 5-point Likert scale. We conduct post-experiment interviews with the participants to understand the reasons of their ratings.

Data analysis: After the experiments, we compile the question-completion-time statistics and the question-difficulty and answer-confidence ratings of the two groups. The two authors mark the correct API caveats in the submitted answers against the ground-truth answers. We use Wilcoxon Rank Sum Test [46] to measure whether the difference of question-completion-time statistics and the difference of correct-API-caveats percentages between the control and experiment group is statistically significant at p-value < 0.05.

Results: Table VII shows the average question-completion time (AveQCT) and the average correct-API-caveats percentage (AveCP) of each participant in the two groups. Overall, the participants of the experimental group complete the questions faster than those of the control group (86.13±22.26 seconds versus 142.33±50.02 seconds), and the API caveats that the experimental group finds are more accurate than those found by the control group (62.91±6.76% versus 40.94±15.42%). The Wilcoxon Rank Sum Test shows that both the difference between question-completion time and the difference of correct-API-caveats percentage between the two groups are statistically significant at p-value < 0.05.

Fig. 3 compares the question-difficulty ratings and the answer-confidence ratings of the two groups. The ratings results are rather surprising to us. We expect that the experimental group would rate the question-difficulty lower than the control group and rate the answer-confidence higher than the control group. However, the actual ratings are opposite to our expectation. The ratings results also contradict to the performance results of the two groups. That is, although the control group has lower question-difficulty ratings and higher answer-confidence ratings than the experimental group, the objective question-completion-time and the correct-API-caveats percentage of the control group are both worse than the performance of the experimental group.

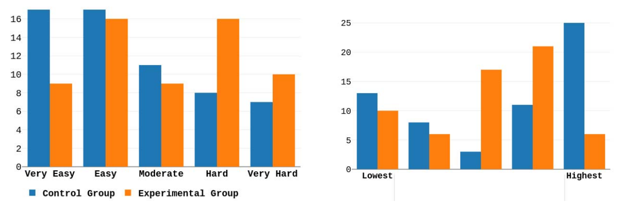
We interview the participants about this contradiction between the objective performance results and the subject ratings. We find that the control group participants have to read long documents to find the API caveats. As the time is limited, they usually use the most seemingly correct information they have read as the answer. Within the limited information they have read, they thought that they found the right API caveat easily. Therefore, they tend to rate the question-difficulty lower and the answer-confidence higher. Unfortunately, they do not realize that the questions are not as easy as they thought and they actually miss other important API caveats.

In contrast, the experimental group participants directly see a list of highly relevant API caveats. On the one hand, this helps them include more relevant API caveats in their answers in a shorter time. On the other hand, this leaves them many seemingly relevant API caveats to compare and judge. As a result, they tend to rate the question more difficult and their confidence in answers lower.

The objective performance results and the "surprising" subject ratings reveal the accessibility issue of API caveats in API documentation which could create an illusion of already knowing the right API usage. As our knowledge graph makes the API caveats more easily accessible, bewaring of the API caveats would make the developers realize that using an API properly may not be as easy as it looks. This improved awareness of API caveats could make the developers more cautious when using an API, and thus potentially avoiding some mistakes in the first place.

E. Threats to Validity

A threat to internal validity is the annotation errors when examining the output of the knowledge graph construction steps. To decrease human errors, we have two annotators annotate the data independently, and their annotation results have an almost perfect agreement. Another internal threat is the equivalence of the control and experimental group. To mitigate this threat, we recruit the students with the similar course-taking history and academic records. Furthermore, student participants may



(a) Question Difficulty (b) Answer Confidence

Fig. 3. Question-Difficulty Ratings and Answer-Confidence Ratings

lack practical programming experiences which may limit the generalization of our results to professional developers. The major threat to external validity is the generalization of our results. Our proof-of-concept tool involves only Android API documentation and our user study is small scale. In the future, we will reduce this threat by applying our approach to more API documentation and release our knowledge graph and tool for public evaluation.

VII. Conclusions and Future Work

In this paper, we first show that API caveats in API documentation have often been overlooked which consequently cause unexpected program errors that affect millions of developers. To tackle this API-caveats accessibility issue, we present an NLP approach to extract API entities, declared relations between APIs and API caveat sentences from API documentation, and organize the extract API information into an API caveats knowledge graph. Different from traditional document search, this knowledge graph enables an entity-centric paradigm for searching API caveats in a more structured way. We validate our approach by constructing a large Android API caveats knowledge graph. The core mining steps of our approach demonstrate the high accuracies in extracting and linking API caveat sentences. A small-scale user study provides the initial evidence that API caveats knowledge graph can make API caveats more easily accessible and potentially improve the awareness of API caveats. In the future, we will investigate more flexible caveat sentence extraction methods and improve the accuracy of co-reference resolution. We will extend our approach to API documentation with different document characteristics, for example, using different markup styles or adopting different caveat description.

Acknowledgment

Hongwei Li gratefully acknowledge financial support by China Scholarship Council, Jiangxi Provincial Department of Education Science and Technology Research Project under Grant No.GJJ160278, and National Natural Science Foundation of China under Grant No.61562042.

References

[1] W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.

[2] M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini, "What should developers be aware of? an empirical study on the directives of api documentation," *Empirical Software Engineering*, vol. 17, no. 6, pp. 703–737, 2012.

[3] M. P. Robillard, "What makes apis hard to learn? answers from developers," *IEEE Software*, vol. 26, no. 6, pp. 27–34, 2009.

[4] G. Uddin and M. P. Robillard, "How api documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, 2015.

[5] B. A. Myers and J. Stylos, "Improving api usability," *Communications of the ACM*, vol. 59, no. 6, pp. 62–69, 2016.

[6] D. Ko, K. Ma, S. Park, S. Kim, D. Kim, and Y. Le Traon, "Api document quality for resolving deprecated apis," in *Proceedings of the 21st Asia-Pacific Software Engineering Conference*, vol. 2. IEEE, 2014, pp. 27–30.

[7] M. P. Robillard and R. Deline, "A field study of api learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[8] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 643–652.

[9] M. P. Robillard and Y. B. Chhetri, "Recommending reference api documentation," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1558–1586, 2015.

[10] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE, 2012, pp. 47–57.

[11] P. C. Rigby and M. P. Robillard, "Discovering essential code elements in informal documentation," in *Proceedings of the 35th International Conference on Software Engineeringon*. IEEE, 2013, pp. 832–841.

[12] X. Chen and J. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques," in *Proceedings of the 26th International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 223–232.

[13] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 404–415.

[14] H. Li, Z. Xing, X. Peng, and W. Zhao, "What help do developers seek, when and how?" in *Proceedings of the 20th Working Conference on Reverse Engineering*. IEEE, 2013, pp. 142–151.

[15] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 279–290.

[16] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcml: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration," in *Proceedings of the 29th International Conference on Software Maintenance*. IEEE, 2013, pp. 516–519.

[17] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced faqs into api documentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 456–459.

[18] E. C. Campos, L. B. Souza, and M. d. A. Maia, "Searching crowd knowledge to recommend solutions for api usage tasks," *Journal of Software: Evolution and Process*, vol. 28, no. 10, pp. 863–892, 2016.

[19] H. Jiang, J. Zhang, Z. Ren, and T. Zhang, "An unsupervised approach for discovering relevant tutorial fragments for apis," in *Proceedings of the 39th International Conference on Software Engineering*. ACM, 2017, pp. 38–48.

[20] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza, "Too long; didn't watch!: extracting relevant fragments from software development video tutorials," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016, pp. 261–272.

[21] S. Bradner, "Key words for use in rfcs to indicate requirement levels," Harvard University, Tech. Rep., 1997. [Online]. Available: https://tools.ietf.org/html/rfc2119

[22] M. Bruch, M. Mezini, and M. Monperrus, "Mining subclassing directives to improve framework reuse," in *Proceedings of the 7th Working Conference on Mining Software Repositories*. IEEE, 2010, pp. 141–150.

[23] U. Dekel and J. D. Herbsleb, "Improving api documentation usability with knowledge pushing," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE, 2009, pp. 320–330.

[24] B. Bokowski, J. Arthorne, and J. des Rivires, "Designing eclipse apis," in *Tutorial at the EclipseCon conference*, 2006.

[25] B. Bokowsk, "Java api design," in *Tutorial at the EclipseCon conference*, 2008.

[26] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[27] M. Recasens, M.-C. de Marneffe, and C. Potts, "The life and death of discourse entities: Identifying singleton mentions," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 627–633.

[28] H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky, "Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task," in *Proceedings of the 15th conference on computational natural language learning: Shared task*. Association for Computational Linguistics, 2011, pp. 28–34.

[29] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning, "A multi-pass sieve for coreference resolution," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2010, pp. 492–501.

[30] C. Scaffidi, "Why are apis difficult to learn and use?" *Crossroads*, vol. 12, no. 4, pp. 4–4, 8 2006.

[31] Y. Zhou, R. Gu, T. Chen, Z. Huang, S. Panichella, and H. Gall, "Analyzing apis documentation and code to detect directive defects," in *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 2017, pp. 27–37.

[32] H. Jiang, L. Nie, Z. Sun, Z. Ren, W. Kong, T. Zhang, and X. Luo, "Rosf: Leveraging information retrieval and supervised learning for recommending code snippets," *IEEE Transactions on Services Computing*, vol. -, no. -, pp. 1–1, Doi:10.1109/TSC.2016.2592909 2016.

[33] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social q&a sites are changing knowledge sharing in open source software communities," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 342–354.

[34] R. Suzuki, "Interactive and collaborative source code annotation," in *Proceedings of the 37th International Conference on Software Engineering-Volume 2*. IEEE Press, 2015, pp. 799–800.

[35] M. Squire, ""should we move to stack overflow?" measuring the utility of social media for developer support," in *Proceedings of the 37th International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 219–228.

[36] R. Abdalkareem, E. Shihab, and J. Rilling, "What do developers use the crowd for? a study using stack overflow," *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017.

[37] A. Bacchelli, M. Lanza, and V. Humpa, "Towards integrating e-mail communication in the ide," in *Proceedings of 2010 Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*. ACM, 2010, pp. 1–4.

[38] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes, "Benchmarking lightweight techniques to link e-mails and source code," in *Proceedings of the 16th Working Conference on Reverse Engineering*. IEEE, 2009, pp. 205–214.

[39] D. Ye, L. Bao, Z. Xing, and S.-W. Lin, "Apireal: an api recognition and linking approach for online developer forums," *Empirical Software Engineering*, pp. 1–32, 2018.

[40] C. Treude, M. Sicard, M. Klocke, and M. Robillard, "Tasknav: Task-based navigation of software documentation," in *Proceedings of the 37th International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 649–652.

[41] D. Ye, Z. Xing, C. Y. Foo, J. Li, and N. Kapre, "Learning to extract api mentions from informal natural language discussions," in *Proceedings of the 32nd International Conference on Software Maintenance and Evolution*. IEEE, 2016, pp. 389–399.

[42] C. X. Chu, N. Tandon, and G. Weikum, "Distilling task knowledge from how-to communities," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 805–814.

[43] G. A. Bartelt, R. E. Rolley, and L. E. Vine, "Evaluation of abundance indices for striped skunks, common raccoons and virginia opossums in southern wisconsin," Tech. Rep., 2001.

[44] R. Singh and N. S. Mangat, *Elements of Survey Sampling*. Dordrecht ; Boston: Kluwer Academic Publishers, 01 1996, vol. 15.

[45] J. R. Landis and G. G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, pp. 363–374, 1977.

[46] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.