Demystify Official API Usage Directives with Crowdsourced API Misuse Scenarios, Erroneous Code Examples and Patches

Xiaoxue Ren[*][†]
Zhejiang University
Hangzhou, Zhejiang, China
xxren@zju.edu.cn

Jiamou Sun[‡]
Australian National University
Canberra, ACT, Australia
Jiamou.Sun@anu.edu.au

Zhenchang Xing[§]
Australian National University
Canberra, ACT, Australia
zhenchang.Xing@anu.edu.au

Xin Xia[¶]
Monash University
Melbourne, VIC, Australia
xin.xia@monash.edu

Jianling Sun
Zhejiang University
Hangzhou, Zhejiang, China
sunjl@zju.edu.cn

ABSTRACT

API usage directives in official API documentation describe the contracts, constraints and guidelines for using APIs in natural language. Through the investigation of API misuse scenarios on Stack Overflow, we identify three barriers that hinder the understanding of the API usage directives, i.e., lack of specific usage context, indirect relationships to cooperative APIs, and confusing APIs with subtle differences. To overcome these barriers, we develop a text mining approach to discover the crowdsourced API misuse scenarios on Stack Overflow and extract from these scenarios erroneous code examples and patches, as well as related API and confusing APIs to construct demystification reports to help developers understand the official API usage directives described in natural language. We apply our approach to API usage directives in official Android API documentation and android-tagged discussion threads on Stack Overflow. We extract 159,116 API misuse scenarios for 23,969 API usage directives of 3138 classes and 7471 methods, from which we generate the demystification reports. Our manual examination confirms that the extracted information in the generated demystification reports are of high accuracy. By a user study of 14 developers on 8 API-misuse related error scenarios, we show that our demystification reports help developer understand and debug API-misuse related program errors faster and more accurately, compared with reading only plain API usage-directive sentences.

[*]Also with Ningbo Research Institute.
[†]Also with PengCheng Laboratory.
[‡]Contribute equally as co-first author.
[§]Also with Data61, CSIRO.
[¶]Corresponding author.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380430

CCS CONCEPTS

• **Software and its engineering** → *Software libraries and repositories.*

KEYWORDS

API usage directive, API misuse, Stack Overflow, Open information extraction

ACM Reference Format:
Xiaoxue Ren, Jiamou Sun, Zhenchang Xing, Xin Xia, and Jianling Sun. 2020. Demystify Official API Usage Directives with Crowdsourced API Misuse Scenarios, Erroneous Code Examples and Patches. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380430

1 INTRODUCTION

Application Programming Interfaces (APIs) provide the access to the features or data of an operating system, framework, or other service. Official API documentation, such as Java API Specification and Android API reference, describes the important API knowledge. Among different types of API knowledge, API usage directives are contracts, constraints, and guidelines that specify what developers are allowed/not allowed to do with the API. A recent work by Li et al. [9] calls such directives as API caveats because they are the knowledge that developers should be aware of to avoid API misuse. They extract a large number of API caveats from official API documentation and organize them into an API caveats knowledge graph to improve the accessibility of API caveats knowledge.

In this work, we are concerned with another important aspect of API usage directives in official API documentation, i.e., the understandability of API usage directives. Our investigation of the misuse scenarios of Android APIs in Stack Overflow questions reveal three correlated barriers that hinder the understanding of API usage directives. First, API usage directives are often abstract and lack specific usage context to illustrate when they are applicable. Second, an API usage directive often involve several cooperative APIs, but it may be described in just one or some APIs, and its relationships to other APIs, especially newly introduced APIs, may not be explicitly documented. Third, several APIs may support similar but different features, but their subtle differences are often hard to spot or understand from API usage directives. We identified the three barriers by observing 1000 randomly-sampled Stack Overflow questions that mention some Android APIs. As Stack Overflow has over 1 million Android questions, it is hard to collect accurate statistics. But the results of our approach could provide a rough estimation of the potential impact of the three barriers on correct API usage: we had extracted 159,116 API misuse scenarios for 23,969 API usage directives of 3138 classes and 7471 methods. Section 2 will illustrate these three barriers with concrete examples.

As our examples show, these barriers often result in the overlook or misunderstanding of certain API usage directives, which in turn results in API misuses. In face of unexpected program errors caused by API misuses, a common practice nowadays is to ask for help on community question and answering (Q&A) websites like Stack Overflow. Over time, such websites have become a repository of crowdsourced API misuse scenarios, erroneous code samples, and patches for fixing the misuses, which, once discovered and attached to the corresponding official API usage directives, will help to demystify convoluted official API usage directives.

In this work, we design a text mining approach to achieve this objective. We first use the API caveat extraction patterns developed in [9] to extract the sentences describing API usage directives in official API documentation. Considering each extracted API usage-directive sentence as a search query, we combine the BM25 ranking [20] and a word-embedding [13, 14] based sentence matching to measure the macro-level and micro-level relevance of the API usage-directive sentences and the Stack Overflow discussion threads (a discussion thread is a question plus all its answers). Next, we consider the relevant Stack Overflow discussion threads as the API misuse scenarios for a given API usage directive, and heuristically extract high-quality erroneous code examples from the questions, the patches from the answers, and related APIs and confusing APIs from the extracted erroneous code examples and patches. Finally, a demystification report is generated which summarizes relevant API misuse scenarios, erroneous code examples and patches, and related and confusing APIs for a given API usage directive.

We apply our approach to official Android API documentation and Android-tagged discussion threads on Stack Overflow. We extract 102,831 API usage directive sentences for 3,969 API classes and 36,627 API methods. We identify 159,116 API misuse scenarios for 23,969 API usage directive sentences of 3,139 API classes and 7,471 API methods. For 28,068 API misuse scenarios, we are able to extract erroneous code examples and patches. By the statistical sampling method [23], we confirm that the accuracies (with the error margin 0.05 at 95% confidence level) of the extracted API misuse scenarios, erroneous code examples and corresponding patches, related APIs, and confusing APIs are 81.34%, 72.88%, 96.23%, 77.01%, respectively. We design a user study to evaluate the usefulness of the generated demystification report for API usage directives in help developers understand API usage directives and debug relevant API misuses. Our user study involves 14 developers and 8 erroneous code snippets caused by API misuses. Our results show that participants, assisted by our demystification report, can identify and debug API misuses faster and more accurately, especially for complex API usage directives, compared with those who are given only plain API usage-directive sentences.

Table 1: Example of Lack of Specific Usage Context

Concerned API	`SynchronousQueue`
Concerned API Usage Directive	They are well suited for handoff designs, in which an object running in one thread must sync up with an object running in another thread in order to hand it some information, event, or task.
API Misuse Scenario	Synchronize handler creation and accessing
Related API(s)	`Loopper`, `Handler`, `Runnable`
Confusing API(s)	None

Table 2: Example of Indirect API Usage Directives

Concerned API	`Intent.ACTION_CHOOSER`
Concerned API Usage Directive	In this case the CHOOSER action should be used, to always present to the user a list of the things they can do, with a nice title given by the caller such as "Send this photo with:"
API Misuse Scenario	Android direct shared
Related API(s)	`Intent.createChooser()`, `SharedCompat.IntentBuilder.createChooserIntent()`
Confusing API(s)	`SharedCompat.IntentBuilder.getIntent()`

This paper makes the following contributions:

- By investigating the API misuse scenarios regarding API usage directives, we identify three correlated barriers that hinder the understanding of API usage directives.
- We design an open information extraction method to extract API misuse scenarios, erroneous code example and patches from Stack Overflow to demystify API usage directives.
- We apply our method to large-scale API usage directives and Stack Overflow discussions, and confirm the high accuracy of our information extraction method and the usefulness of the generated demystification reports for debugging API-misuse related program errors.

2 MOTIVATING EXAMPLES

We now describe three examples[1] to illustrate the three barriers that hinder the understanding of API usage directives respectively, and how the API misuse scenarios on Stack Overflow can help developers overcome these barriers by learning from "somebody else's unfortunate mistakes" related to API usage directives. It is important to note that we organize the discussion in terms of the main barrier involved in each example, but the three barriers have compound effects on API misuses. Furthermore, a API usage directive may have several API misuse scenarios. Due to space limitation, we show only one scenario for each example for illustration purpose.

2.1 Lack of Specific Usage Context

According to the API reference of `SynchronousQueue`, synchronous queues support "*handoff designs, in which an object running*

[1]More examples can be found in our github repository.

Table 3: Example of Confusing APIs with Subtle Differences

Concerned API	`Intent.FLAG_ACTIVITY_NEW_DOCUMENT`
Concerned API Usage Directive	... whether the recent entry for it is kept after the activity is finished is different than the use of `FLAG_ACTIVITY_NEW_TASK` — if this flag is being used to create a new recents entry, then by default that entry will be removed once the activity is finished
API Misuse Scenario	How to create the same Activity Multiple times to have an effect like Google Chrome Tabs?
Related API(s)	`FLAG_ACTIVITY_MULTIPLE_TASK`, `setFlags`
Confusing API(s)	`FLAG_ACTIVITY_NEW_TASK`

in one thread must sync up with an object running in another thread in order to hand it some information, event, or task." Reading this abstract usage guideline, developers may not realize in which part of Android applications the handoff design can be useful. The Stack Overflow question "Synchronizing handler creation and accessing" illustrates a specific erroneous scenario that demands the use of the handoff design. In this scenario, the main UI thread uses the handler created in another thread to post some processing task to the message loop of that thread. As the handler is created asynchronously to the main UI thread, the handler will sometimes be NULL at the time the main UI thread uses it. The accepted answer suggests that `SynchronousQueue` is a perfect solution to solve this problem. Inspecting this erroneous scenario, developers can learn not only specific usage contexts of `SynchronousQueue`, but also observe several APIs (`Looper`, `Handler`, `Runnable`) that are related to the use of `SynchronousQueue`.

2.2 Indirect API Usage Directives

According to the API reference of `Intent.ACTION_CHOOSER`, it "*should be used, to always present to the user a list of things they can do*". The API reference also points out a static helper function `Intent.createChooser()` - "*As a convenience, an Intent of this form can be created with the* `createChooser()` *function*". However, the API reference of `Intent.createChooser()` does not explicitly mention when this helper function should be used. That is, developers may not realize the API usage directive associated with `Intent.createChooser()`, which is only explained for `Intent.ACTION_CHOOSER`. Similarly, another helper function `ShareCompat.IntentBuilder.createChooserIntent()` can also be used to create the intent of `ACTION_CHOOSER`. It does not explicitly mention the relevant API usage directive either. Even worse, it was added in version 22.1.0. The `Intent.ACTION_CHOOSER` description does not mention this new helper function.

The Stack Overflow question Android direct shared illustrates an API misuse scenario in which the developer uses the wrong API `ShareCompat.IntentBuilder.getIntent()`, but what he should use is `ShareCompat.IntentBuilder.createChooserIntent()`, as suggested by the accepted answer. As a result, the developer cannot achieve the goal "The share dialog must show the most used contacts from messaging apps, like WhatsApp contacts". As the accepted answer only suggests the correct API to use, the other answer complements the accepted answer by quoting the API usage directive of `Intent.ACTION_CHOOSER` to explain why createChooserIntent() should be used instead of `getIntent()`. In addition, there is another answer which solves the problem using `Intent.createChooser()`. Inspecting this API misuse scenario, developers can observe two helper APIs (`Intent.createChooser()`, `ShareCompat.IntentBuilder.createChooserIntent()`) that are related to the API usage directive of `Intent.ACTION_CHOOSER`, as well as a confusing API `IntentBuilder.getIntent()`.

2.3 Confusing APIs with Subtle Differences

Then Intent class declares many flags, some of which have very similar names, such as FLAG_ACTIVITY_NEW_DOCUMENT and FLAG_ACTI-VITY_NEW_TASK. Although they look very similar, the API reference of `FLAG_ACTIVITY_NEW_DOCUMENT` warns that "*... whether the recent entry for it is kept after the activity is finished is different than the use of* `FLAG_ACTIVITY_NEW_TA- SK` *— — if this flag is being used to create a new recents entry, then by default that entry will be removed once the activity is finished*". This warning indicates some subtle difference between the two flags. First, this warning is quite convoluted, Second, `FLAG_ACTIVITY_NEW_TASK` does not have this warning.

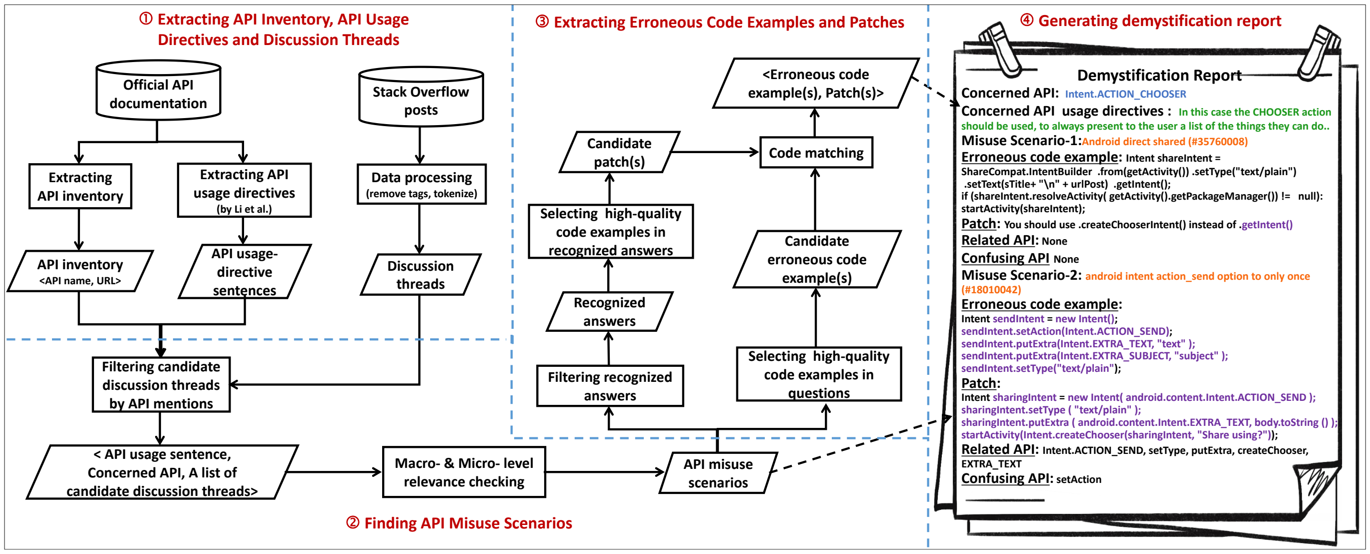
The question "How to create the same Activity Multiple times to have an effect like Google Chrome Tabs?" illustrates an exact scenario in which the developer misuses `FLAG_ACTIVITY_NEW_TASK`, but should use `FLAG_ACTIVITY _NEW_DOCUMENT` instead. This type of misuse also occurs in other questions such as "How to make tasks in Overview Screen looks like together like Chrome?". In the second scenario, the developer made another error. He did not realize that he not only needs to use the right `NEW_DOCUMENT` or `NEW_TASK` flag, but also needs to use that flag together with `FLAG_ACTIVITY_MULTIPLE_TASK`. As a result, his program did not work as expected, although he tried each flag individually. This cooperative usage need is described in the API reference of the three flags, but the descriptions lack specific usage context. Inspecting these two API misuse scenarios, developer can understand better the subtle difference between the two confusing flags `NEW_DOCUMENT` or `NEW_TASK` from the errors caused by their misuses. Furthermore, they can understand better the cooperative need of using them with the flag MULTIPLE_TASK.

3 APPROACH

Figure 1 shows an overview of our approach, which includes four main steps: extract API inventory, API usage directives and discussion threads from official and crowdsourced documentation, finding API misuse scenarios, extracting erroneous code examples and patches, and generating demystification report.

3.1 Extracting API Inventory, API Usage Directives and Discussion Threads

Our approach aims to finding API misuse scenarios in crowdsourced documentation which can be used to demystify API usage directives in official API documentation. Therefore, the raw input to our approach include both official API documentation and crowdsourced documentation. In this work, we exemplify and evaluate our approach with official Android API documentation and Stack


① Extracting API Inventory, API Usage Directives and Discussion Threads
Official API documentation
Stack Overflow posts
Extracting API inventory
Extracting API usage directives (by Li et al.)
Data processing (remove tags, tokenize)
API inventory <API name, URL>
API usage-directive sentences
Discussion threads
Filtering candidate discussion threads by API mentions
< API usage sentence, Concerned API, A list of candidate discussion threads>
Macro- & Micro- level relevance checking
API misuse scenarios
② Finding API Misuse Scenarios
③ Extracting Erroneous Code Examples and Patches
<Erroneous code example(s), Patch(s)>
Candidate patch(s)
Code matching
Selecting high-quality code examples in recognized answers
Candidate erroneous code example(s)
Recognized answers
Filtering recognized answers
Selecting high-quality code examples in questions
④ Generating demystification report

Demystification Report
Concerned API: Intent.ACTION_CHOOSER
Concerned API usage directives : In this case the CHOOSER action should be used, to always present to the user a list of the things they can do..
Misuse Scenario-1:Android direct shared (#35760008)
Erroneous code example: Intent shareIntent =

```
ShareCompat.IntentBuilder  .from(getActivity()) .setType("text/plain")
 .setText(sTitle+ "\n" + urlPost)  .getIntent();
if (shareIntent.resolveActivity( getActivity().getPackageManager()) !=   null):
startActivity(shareIntent);
```

Patch: You should use .createChooserIntent() instead of .getIntent()
Related API: None
Confusing API None
Misuse Scenario-2: android intent action_send option to only once (#18010042)
Erroneous code example:

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "text" );
sendIntent.putExtra(Intent.EXTRA_SUBJECT, "subject" );
sendIntent.setType("text/plain");
```

Patch:

```
Intent sharingIntent = new Intent( android.content.Intent.ACTION_SEND );
sharingIntent.setType ( "text/plain" );
sharingIntent.putExtra ( android.content.Intent.EXTRA_TEXT, body.toString () );
startActivity(Intent.createChooser(sharingIntent, "Share using?"));
```

Related API: Intent.ACTION_SEND, setType, putExtra, createChooser, EXTRA_TEXT
Confusing API: setAction

Figure 1: Main Steps of Our Approach

Overflow questions and answers. Android system has complex APIs, and Android API documentation contains a large number of API usage directives [9]. Stack Overflow is the most popular community question and answering site for computer programming, and it contains over 1.2 millions android-tagged questions that provides a rich source for Android API misuse scenarios.

From the official Android Developers website, we crawl both API Reference and Developer Guides web pages (as of 31 March 2019). We discard web page navigation content and retain the main API reference and tutorial content. By parsing the semi-structured API reference content, we build an inventory of Android APIs. The API inventory stores the fully-qualified name of each API and its URL in the Android API reference website. We use the API usage-directive extraction patterns developed by Li et al. [9] to extract API usage directives from the crawled Android API reference and develop-guide documentation. These API usage-directive extraction patterns essentially look for uage-directive-indicating sentence patterns. Table 1, 2 and 3 shows some examples of the extracted API usage-directive sentences in which the directive-indicators are highlighted in red. Readers are referred to [9] for the complete list of directive-indicating sentence patterns. We use the entity linking method in [9] to link the extracted API usage-directive sentences to the corresponding APIs in the API inventory.

From the latest Stack Overflow data dump of 4th March, 2019, we collect the android-tagged questions and their answers as a corpus of discussion threads. The content of a question and all its answers is merged into a discussion thread. The questions with no answers are discarded as they has little value for problem solving. We retain the textual content and extract the long code snippets in <pre><code>, but remove the content in <image>. Long code snippets are stored separately from the textual content and are linked to the questions and answers from which they are extracted. Note that short code elements in <code> in text remain in text to keep the sentence integrity. All HTML tags are removed from the text. We split the text into sentences by punctuation, and use domain-specific tokenizer [30] to tokenize the sentences. This tokenizer preserves the integrity of code-like tokens and the sentence structure. For example, it treats the API `ConnectedTread.cancel()` mentioned in text as a single token, rather than a sequence of 5 tokens: ConnectedTread . cancel ( ). This supports accurate detection of API mentions in Stack Overflow text and accurate identification of API misuse scenarios and corresponding erroneous code examples and patches. For the extracted long code snippets, we use the API linking method developed in [24] to determine the API(s) used in the code snippets against the official API inventory. We use a modified version of an ANTLR parser to tokenize code snippets.

3.2 Finding API Misuse Scenarios

Given a API usage directive, we try to find relevant API misuse scenarios in the corpus of Stack Overflow discussion threads. The API of this given API usage directive is referred to as the concerned API. First, we detect the mentions of the concerned API in the discussion threads to narrow down the candidates. Then, we combine the BM25 ranking [20] and the word-embedding [13, 14] based sentence matching to estimate the macro- and micro-level relevance of candidate discussion threads to the given API usage-directive sentence, respectively. Each relevant discussion thread is considered as a API misuse scenario for the given API usage directive (see Table 1, Table 2 and Table 3 for examples). A API usage directive may have multiple API misuse scenarios as illustrative in Figure 1.

3.2.1 Filtering Candidate Discussion Threads by API Mentions. We detect the mentions of the concerned API in discussion threads in two ways. First, if a token in a discussion thread matches the name of the concerned API, this token is considered as a mention of the concerned API. In the informal discussions on Stack Overflow, it is common that a class is mentioned without package name, and a method is mentioned without package/class name or parameters [30]. Therefore, we perform the partial name matching between the tokens and the API names. Second, a discussion thread may not explicitly mention the API name, but it may reference the API URL in the API reference website. Therefore, we also examine each hyperlink in the discussion thread. If a hyperlink matches the URL of the concerned API in the API inventory, this hyperlink is considered as a mention of the concerned API. The discussion threads that mention at least once the concerned API are the candidates for finding API misuse scenarios.

3.2.2 Determining the Relevance of Discussion Threads to API Usage-Directive Sentence. We consider the given API usage-directive sentence as a search query and each of the candidate discussion threads as a document in the corpus of all android-tagged discussion threads. Our relevance function combines the macro-level relevance by the BM25 ranking and the micro-level relevance by the word-embedding based sentence matching (SM_{W2V}). The relevance estimation considers only the textual content of the discussion threads, but not the large code snippets in the discussion threads.

Macro-level relevance by Okapi BM25: The Okapi BM25 [20], a ranking function based on the Term-Frequency/ Inverse-Document-Frequency (TF/IDF) metric. It is simple to compute and has been widely used by search engines to rank the documents according to their relevance to the search query. In our application of BM25, except removing regular stop words (e.g., is, the), we retain all other tokens in text for indexing.

Let q_i be a term in the search query Q and D be a document. The correlation score of a term q_i in Q and D is calculated by $tf(q_i, D)$, i.e., the term frequency of q_i in the document D. The higher the term frequency is, the more relevant the document is to the query term. The correlation score of Q and D is the weighted sum of the correlation score of each query term and the document as follows: $Score_{BM25}(Q, D) = \sum_{i=1}^{n} idf(q_i) \cdot \frac{tf(q_i,D)\cdot(k_1+1)}{tf(q_i,D)+k_1\cdot(1-b+b\cdot|D|/avgdl)}$, where $|D|$ is the length of the document D in tokens, and $avgdl$ is the average document length in the corpus; k_1 and b are free parameters, usually $k_1 \in [1.2, 2.0]$ and $b = 0.75$. $idf(q_i)$ is the Inverse Document Frequency weight of the query term q_i, which is computed as $idf(q_i) = log\frac{N-n(q_i)+0.5}{n(q_i)+0.5}$, where N is the total number of documents in the corpus, and $n(q_i)$ is the number of documents containing q_i. That is, the more documents contain the query term, the less important the query term is towards the overall query-document correlation. We use $Score_{BM25}(Q, D)$ to estimate the macro-level relevance of a discussion thread D to a given API usage-directive sentence Q.

Micro-level relevance by word-embedding based sentence matching: We learn domain-specific word embeddings using the continuous skip-gram model [13, 14] on the corpus of all android-tagged discussion threads. The continuous skip-gram model learns the word representation of each word that is good at predicting the surrounding words in the sentences in a corpus. The learned word embeddings can capture important syntactic and semantic features of words and improve the performance of many natural language processing tasks, including text retrieval [6, 7, 31] We learn domain-specific word embeddings because recent studies [3, 29] show that domain-specific word embedding outperforms general word embedding for domains-specific text retrieval. We set the word embedding dimension at 200, as this setting has the best performance on similar corpora in existing studies [4].

Given a sentence S, we obtain the sentence embedding V_S by average pooling [2], i.e., average the word embeddings of the words in the sentence. We measure the similarity of the two sentences by the cosine similarity of the two sentence embeddings. Using this method, we compute the similarity of the given API usage-directive sentence Q and each of the sentences S in a discussion thread D, and use the highest similarity score as the micro-level relevance $Score_{W2V}$ of the discussion thread to the given API usage-directive sentence. That is, $Score_{W2V}(Q, D) = max_{S \in D} cos(V_S, V_Q)$.

Combined relevance: Given a API usage directive Q and a discussion thread D, the macro-level relevance $Score_{BM25}(Q, D)$ indicates the relevance of the whole discussion thread to the API usage directive, while the micro-level relevance $Score_{W2V}(Q, D)$ identifies a sentence in the discussion thread that is the most relevant to the API usage directive. Finally, we average the macro-level relevance $Score_{BM25}(Q, D)$ and micro-level relevance $Score_{W2V}(Q, D)$ as the overall relevance. We observe that when a discussion thread is relevant to a API usage directive, the discussion thread likely discusses some misuses of the concerned API. Therefore, if the overall relevance of the API usage directive and the discussion thread is above the user-specified threshold, we consider the discussion thread as a API misuse scenario for the API usage directive. We empirically set the threshold at 0.5 in our current implementation, which achieves a good balance of accuracy and diversity in finding API misuse scenarios on a small validation dataset of 100 randomly sampled API usage directives.

3.3 Extracting Erroneous Code Examples and Patches

Given a API misuse scenario on Stack Overflow for a API usage directive, we attempt to extract the erroneous code example(s) from the question and the patch(es) from the answer(s). We observe that the question in a API misuse scenario usually contains erroneous code example(s), which is a good practice advocated by the Stack Overflow community, but the answers may provide code patch(es) or just explain the patch(es) in natural language (e.g., "You should use createChooserIntent() instead of getIntent()"). The demystification report in Figure 1 illustrates some examples of the erroneous code examples and patches (code or natural language) extracted from the API misuse scenarios for `Intent.ACTION_CHOOSER`.

As not all code examples in Stack Overflow questions and answers are of good quality [16], we adopt the heuristics developed by Nasehi et al. [16] to extract high-quality erroneous code examples and patches in API misuse scenarios. First, we check if the API misuse scenario has recognized answers. According to [16], a recognized answer is an accepted answer, an answer with the vote greater than 10, or an answer with the normalized vote greater than 0.4. The normalized vote is calculated by $X' = \frac{X-X_{min}}{X_{max}-X_{min}}$ where X is the vote of an answer in a discussion thread.

If the API misuse scenarios has one or more recognized answers, then we use the code filters developed in [16] to extract high-quality code examples in both the question and the recognized answers. The filters filter out too-long code (> 120 lines, roughly about 2 A4 pages) which has too much distractive information and code fragments showing error information like stack traces. The retained code snippets in the questions are candidate erroneous code examples, and the retained code snippets in the recognized answers are candidate code patches. If a recognized answer does not contain code snippets, but its length is less than 120 lines and it mentions certain APIs (e.g., the accepted answer in the API misuse scenario Android direct shared), we use the answer text (e.g., "You should use createChooserIntent() instead of getIntent()") as a candidate natural language patch. However, if the answer already contains a candidate code patch, its text is omitted.

Let Can_e be a candidate erroneous code example and Can_p be a candidate patch (code or natural language). We establish the correspondence between Can_e and Can_p based on their similarity by bag-of-token matching. Bag-of-token matching is essentially the same as the bag-of-words model [32] for Information Retrieval. We call it bag-of-token in our work, because our tokenizer preserves the integrity of code-like tokens, rather than split them into separate English words (see Section 3.1). We use bag-of-token matching rather than code differencing commonly used for comparing programs for two reasons. First, a candidate patch can be in natural language, rather than code snippet. Second, code snippets on Stack Overflow are rather informal, and often incomplete. For example, a code patch may copy-modify only a small portion of erroneous code. Thus, code differencing techniques designed for comparing different versions of a complete program will not work for our erroneous code examples and patches.

In this work, if the similarity of Can_e and Can_p by bag-of-token matching is $(0.5, 1]$, we consider Can_e and Can_p as a pair of erroneous code example and patch (see Figure 1 for illustrative examples.) Some program errors are caused by wrong API call sequence, and the answers sometimes copy the entire code snippet in the question and change just the statement sequence. In such cases, bag-of-token matching gives the similarity 1. However, we observe that the answerers sometimes copy the entire code snippet in the question, but they just discuss the copied code without changing anything in code. In such cases, the code in the answer is not really a patch to the erroneous code in the question. So when the similarity of Can_e and Can_p is 1, we further compute their sequence similarity by the difflib tool. If the sequence similar is also 1, we will discard Can_p but consider the corresponding answer text as a candidate patch.

3.4 Generating Demystification Report

Finally, we compose the found API misuse scenario(s) and the extracted pairs of erroneous code example(s) and code patch(es) (if any) into a demystification report for the given API usage directive and the concerned API. Figure 1 shows an example of the demystification report. To assist developers in understanding the demystification report, we detect and highlight the code differences between the erroneous code example and the corresponding code patch if their similarity is > 0.7. If the patch is in natural language, we highlight the API(s) in the erroneous code that are mentioned in the natural language patch.

Furthermore, we extract related APIs and confusing APIs from the code differences between the erroneous code example and the corresponding code patch, and list these APIs in the demystification report. We obtain three sets of APIs: $Set1$ contains all APIs extracted from the code differences between the erroneous code and the corresponding patch. $Set2$ contains all APIs extracted from the erroneous code. $Set3$ contains all APIs mentioned in the API reference document for the concerned API (e.g., `FLAG_ACTIVITY_MULTIPLE_TASK` for `FLAG_ACTIVITY_NEW_DOCUMENT`, `createChooser()` for `Intent.ACTION_CHOOSER`). We consider the APIs that the question askers misuse in their erroneous code but the answerers do not use in their patch as likely confusing APIs. So we identify confusing API(s) by $Set1 \cap Set2$, i.e., the APIs appear in the code differences at the erroneous code side. We further identify related APIs by $Set1 \cap Set2$, excluding the concerned API and the confusing API(s). See Section 2 and Figure 1 for examples of related APIs and confusing APIs extracted from API misuse scenarios.

4 QUALITY OF EXTRACTED INFORMATION FOR DEMYSTIFICATION REPORT

Our approach forms a demystification report to assist the understanding of a API usage directive. The content of the demystification report is extracted from the community Q&A discussions on Stack Overflow, which is informal, noisy software text. We carefully examine the quality of three types of extracted information for the demystification reports: API misuse scenarios, erroneous code examples and patches, and related API and confusing APIs.

4.1 Experiment Setup

4.1.1 Experimental Datasets. We build an API inventory from Android API reference and Developer Guides. We carefully verify our web crawler implementation to ensure its correctness in extracting API information from the web documentation. Using the method in [9], we extract 102,831 API usage-directive sentences for 3,969 API classes and 36,627 API methods. The accuracy of this API-usage-directive-extraction method has been evaluated in the two previous studies [9?], which extract the API usage-directive sentences from the same Android documentation as our study. The reported accuracies are 98.85% and 99.2% in the two studies respectively.

We use the Stack Overflow data dump of 4th March, 2019 in this study, which has 1,176,198 android-tagged questions and 1,692,686 answers. From these discussion threads, we identify 159,116 API misuse scenarios for 23,969 API usage directive sentences of 3,139 API classes and 7,471 API methods. 11,823 API usage directives have one API misuse scenario, 5,239 have two, and 6,907 have three or more. For 28,068 API misuse scenarios, we are able to extract erroneous code examples and patches. For 13,307 API misuse scenarios, we extract some related APIs and/or confusing APIs.

4.1.2 Evaluation Method. As our approach extracts large numbers of data instances (i.e., API misuse scenarios, erroneous code examples and patches, and related APIs and confusing APIs for API usage directives), we adopt a statistical sampling method [23] to examine the minimum number MIN of API usage directives for each type of the extracted information in the demystification report. This sampling method ensures that the estimated accuracy is in a certain error margin at a certain confidence level. This MIN can be determined by the formula: $MIN = n_0/(1 + (n_0 - 1)/populationsize)$. In the equation, n_0 depends on the selected confidence level and the desired error margin: $n0 = (Z^2 * 0.25)/e^2$, where Z is a confidence level's z-score and e is the error margin. We use the error margin 0.05 at 95% confidence level in our evaluation. Given large numbers of API usage directives and API misuse scenarios, MIN is approximately 384 at this statistical setting.

The two authors independently evaluate the accuracy of the sampled data instances. They judge whether or not a data instance (i.e., a API misuse scenario, a combination of an erroneous code example and a patch, a related API, or a confusing API) is relevant to the given API usage directive. We compute Cohen's Kappa [8] to evaluate the inter-rater agreement. For the data instances that the two authors disagree, they have to discuss and come to a consensus. Based on the consensus annotations, we evaluate the quality of each type of the extracted information.

4.1.3 Evaluation Metrics. As the identification of API misuse scenarios and the identification of erroneous code examples and patches are a ranking task, we use Mean Average Precision@k (MAP@k) and Mean Reciprocal Rank (MRR) to evaluate the effectiveness of our information extraction method for finding candidate items. Although many candidate items could be found, we assume that developers would be interested in only the top relevant ones. Therefore, we compute MAP@k (k={1,3,5,10}). MAP@k can be computed by $MAP@k = \sum_{i=1}^{|Q|} AveP@k(Q_i)/|Q|$, where Q is the set of all API usage directives, and $AveP@k(Q_i)$ is average precision@k for the i-th API usage directive. $AveP@k(Q_i)$ is $\sum_{j=1}^{k} P@j \times rel(j)/rel@k$ where $P@j$ is the precision@j, $rel(j)$ is 1 if the item at rank j is relevant, 0 otherwise, and $rel@j$ is the number of all relevant items at and before rank k. If all relevant items are ranked at the top of the list, the MAP@k is 1. Note that irrelevant items contribute 0 to the $AveP@k(Q_i)$. MRR is defined as $MRR = \sum_{i=1}^{|Q|} \frac{1}{rank_i}/|Q|$, where $rank_i$ is the rank position of the first relevant item. If the first relevant item is ranked at the top 1 position for all Q, the MRR is 1. In addition MAP@k and MRR, we compute the accuracy of each type of the extraction information, which is the percentage of the actually relevant data instances over all the extracted data instances. Because our task is an open information extraction problem where the set of all relevant items is unknown, we cannot compute recall.

4.2 Quality of Finding API Misuse Scenarios

4.2.1 Baseline Methods. To find relevant API misuse scenarios for a given API usage directive, our approach combines the macro-level relevance by the BM25 [20] ranking and the micro-level relevance by the word-embedding based sentence matching (SM_{W2V}) (see Section 3.2.2). We consider using only the BM25 or the SM_{W2V} as a baseline. These two baselines finds API misuse scenarios for a API usage-directive sentence in the same database of Stack Overflow discussion threads (filtered by API mentions) as our method. If the similarity of a discussion thread and a API usage-directive sentence is > 0.5 by the BM25 (or the SM_{W2V}), we consider the discussion thread as a API misuse scenario.

Furthermore, we use Google search and the search engine of the Stack Overflow website (SO search) as the other two baselines, which simulate the situation when developers search API misuse scenarios for a API usage-directive sentence on the Web. We limit Google search to search only the Stack Overflow website. As this study involves only android-tagged questions, we add the keyword "android" to the query for the Google search, and add android tag to the query for the SO search. Google search and SO search search all android-tagged discussion threads, without filtering discussion threads by API mentions. We consider the top-10 returned discussion threads as the API misuse scenarios that developers would find using these two public search engines.

4.2.2 Results. Table 4 reports the performance of different methods for finding candidate API misuse scenarios for the sampled 384 API usage directives. For these 384 API usage directives, our method finds at least one API misuse scenario. The MAP@k and MRR metrics are computed based on the consensus annotations by the two annotators. The column Acc. (Improv.) reports the accuracy of the API misuse scenarios extracted by different methods and the improvement by our method over the baselines. The column Kappa is the inter-rater agreement of the two annotator's independent annotations. As shown in Table 4, the Cohen's kappa metrics for all methods are > 0.600, which indicate at least substantial agreement between the two annotators.

Stack Overflow search engine performs the worst (accuracy 25.5%) in finding API misuse scenarios, because it uses very primitive keyword based search [5]. The two annotators mostly agree (Kappa metric achieves 0.794) on the irrelevance of the SO search results as the API misuse scenarios for API usage directives.

Google and the SM_{W2V} (i.e, micro-level sentence matching) has similar performance in ranking candidate API misuse scenarios, but SM_{W2V} has a better accuracy due to its more strict selection threshold (SM_{W2V} 62.06% versus Google Search 52.68%). The high MRR of these two baselines suggest that the first return results are very likely to be relevant API misuse scenario. However, the decreasing MAP@k as k increases indicates that some relevant API misuse scenarios are ranked low in the search results. Inspecting the search results by Google search and the SM_{W2V} reveals that these two baselines tend to return a diverse set of Stack Overflow questions which may be related to the concerned API at the sentence level, but these questions may not provide relevant API misuse scenarios for the concerned API usage directive. For example, Google returns at rank 1 the question "get CORRECT referrer application on deep linking" for the API usage directive "*Note that this is not a security feature – you can not trust the referrer information, applications can spoof it.*" of the method `Activity.getReferrer()`". Unfortunately, only some part of this discussion thread explains how to use the method in the code and references the API usage-directive sentence as a caution. But this discussion thread itself is not a API misuse scenario for the usage directive. Furthermore, due to the diversity of Google search results, the two annotators have the least agreement on whether the top ranked questions are API misuse scenarios.

In contrast, the BM25 and our Method perform much better (BM25 73.88% and our method 81.34%), and the two annotators have almost-perfect agreement on the relevance of API misuse scenarios these two methods find. This suggests that the macro-level relevance ranking by the BM25 is an effective method to find API misuse scenarios for API usage-directive sentences. Furthermore, although the micro-level sentence matching by SM_{W2V} is not very effective in finding API misuse scenarios, the much better MRR by our method (0.957) than the BM25 (0.825) shows that combining the

BM25 and the SM_{W2V} (i.e., both macro- and micro-level relevance) can rank significantly more correct API misuse scenarios to the rank 1 position, compared with using the BM25 only. For example, the BM25 finds a correct API misuse scenario for the API usage directive "*If you do not require all camera features or can properly operate if a camera is not available, then you must modify your manifest as appropriate in order to install on devices that don't support all camera features*". However, it ranks this scenario at the 4th position. With the help of sentence-level relevance, this API misuse scenario is ranked at the first position by our method.

By combining macro- and micro-level relevance, our method outperforms the baselines using either macro-level or micro-level relevance or using the public search engines for finding API misuse scenarios for API usage directives.

4.3 Quality of Extracting Erroneous Code Examples and Patches

4.3.1 Baseline Method. Our approach finds pairs of erroneous code examples and patches by bag-of-tokens matching (Section 3.3). Here we use bag-of-words as a baseline to compare with our method. Bag-of-words treats candidate erroneous code examples and patches as plain text. We apply common text preprocessing steps, including splitting sentences, tokenization, and removing punctuation and stop words. After getting a bag of words, we compute the similarity score of a pair of erroneous code and patch and return the matching pairs in the same way as our bag-of-tokens matching does.

4.3.2 Results. Table 5 shows the performance of the two methods for ranking candidate pairs of erroneous code examples and patches for the 384 sampled API misuse scenarios. For these 384 API misuse scenarios, our method finds at least one pair of erroneous code example and patch. The Kappa metrics are > 0.7 for both methods, which indicates the substantial agreement between the two annotators regarding the relevance of candidate pairs of erroneous code examples and patches to the API usage directives.

The MRR and accuracy of the two methods are very close (MRR 0.774 versus 0.782, Accuracy 70.19% versus 72.88%), with our method being slightly better. The close MRR indicates that bag-of-tokens matching and bag-of-words matching do not make a big difference in ranking the first relevant pair of erroneous code example and patch. However, when looking into MAP@k, we find that bag-of-tokens matching can better rank all relevant pairs to the top of the list, compared with bag-of-words matching. Furthermore, we find that the similarity score by our bag-of-tokens matching is usually much higher than that by bag-of-words matching for the same candidate pair of erroneous code and patch, because our method preserve the integrity of code-like tokens. Unfortunately, this higher similarity score makes many candidate pairs pass the threshold, some of which are incorrect matchings that lowers the accuracy of the recommended pairs of erroneous code examples and patches. In contrast, the same threshold can actually filter out many of these incorrect pairs for the bag-of-words matching, but at the same time many correct pairs are filtered out too. That is, the bag-of-words matching finds much fewer correct pairs of erroneous code examples and patches (Bag-of-words 114 pairs versus our method 537 pairs), even though it has similar accuracy to our method.

Our bag-of-tokens matching can find much more correct pairs of erroneous code examples and patches than bag-of-words matching, with slightly better accuracy. Considering the superior ranking capability of our method, filtering candidate pairs by both similarity score and the top-k ranking could further improve the accuracy.

4.4 Quality of Identifying Related APIs and Confusing APIs

To enrich the content of our demystification report, we identify related API and confusing API from the error code examples and corresponding patches for the API misuse scenarios (see Section 3.4). As a API misuse scenario may not have both related APIs and confusing APIs, we select two sets of 384 API misuse scenarios: those in the Set-1 have at least one related API, and those in Set-2 have at least have one confusing API. Table 6 shows the evaluation results. The Kapps metric is 0.862 (i.e., almost perfect) for the relevance of related APIs, but is lower (0.656, i.e, substantial agreement) for confusing APIs. The accuracy (77.01%) for confusing APIs is very good, but it is much lower than that (96.23%) for related APIs. We determine related APIs by contrasting APIs used in the code difference of erroneous code examples and patches against the APIs mentioned in the document of the concerned APIs. It is not surprising that we have excellent accuracy and annotation agreement for related APIs. However, our heuristic to identify confusing APIs seems a bit simplistic, in face of the informal, noisy code snippets in Stack Overflow discussions. Furthermore, the low Kappa for confusing APIs indicates that human annotators often do not agree on what are confusing APIs or not, because the definition of confusing is rather context sensitive.

Our method can almost perfectly identify related APIs. The accuracy of identify confusing APIs is also acceptable, but it is more challenging partially due to the code informality on Stack Overflow and partially because the vague definition of confusing.

5 USEFULNESS OF DEMYSTIFICATION REPORT

We conduct a user study to evaluate the usefulness of our generated demystification report for understanding API usage directives and debugging API-misuse related program errors. Our user study assumes that developers already identify the concerned API and the concerned API usage directive related to a program error by some means (beyond the scope of this work), and then want to propose some plausible solution to fix the error scenario based on the understanding of the concerned API usage directive (the focus of this study), in a similar way to the Q&A on Stack Overflow.

5.1 User Study Design

5.1.1 Experimental Tasks. Table 7 lists the eight experimental tasks in our study. We create these tasks from the demystification reports that have two or more API misuse scenarios. To ensure the diversity of our experimental tasks, we select the demystification reports that concern different APIs and different API usage directives. Furthermore, the concerned API usage directive in the report should not be too straightforward or too convoluted, which will make the experimental tasks very easy to answer even without any analysis

Table 4: Performance of Finding API Misuse Scenarios

	MAP@1	MAP@3	MAP@5	MAP@10	MRR	Acc. (Improv.)	Kappa
Google	0.718	0.655	0.606	0.557	0.739	52.68% (+54.40%)	0.658
SO Search	0.313	0.302	0.297	0.280	0.316	25.50% (+218.98%)	0.794
BM25	0.918	0.887	0.863	0.842	0.825	73.88% (+10.10%)	0.825
SM$_{W2V}$	0.763	0.670	0.648	0.585	0.769	62.06% (+31.07%)	0.713
Our Method	0.934	0.903	0.899	0.882	0.957	81.34% (-)	0.846

Table 5: Performance of Erroneous Code Examples and Patches

	MAP@1	MAP@3	MAP@5	MAP@10	MRR	Acc. (Improv.)	Kappa
Bag-of-words	0.650	0.639	0.595	0.557	0.774	70.19% (+3.83%)	0.724
Our Bag-of-tokens	0.902	0.897	0.884	0.865	0.782	72.88% (-)	0.758

Table 6: Accuracy of Related API(s) and Confusing API(s)

	Collected API	Correct API	Accuracy	Kappa
Related	637	613	96.23%	0.862
Confusing	548	422	77.01%	0.656

or investigation, or make them too hard to answer in a reasonable time. We select the concerned API usage directives roughly at the similar level of complexity as those discussed in Section 2. Based on our own analysis of these error scenarios, we estimate that Q1/2 are easy, Q3/4/Q5 are medium, and Q6/Q7/Q8 are difficult.

We randomly select one of the API misuse scenarios in the report and use the question of this scenario as the error scenario to be answered by the participants. The question title and body is used as the error scenario description. We generate the experimental tasks for the selected error scenarios as follows. We discard the selected error scenario from the demystification report, and modify the report to keep only one of the other API misuse scenarios. We combine the error scenario question and the modified demystification report as a task for the experimental group that debug the error scenario with our generated demystification report. We attach the concerned API and the concerned API usage-directive sentence to the error scenario question as a task for the control group that debug the error scenario with only API usage-directive sentence. To make the comparison fair, we ensure that the API misuse scenario in the modified demystification report is different from the error scenario, in the sense that they would not be marked as duplicated questions on Stack Overflow.

5.1.2 Experiment Procedure. In our study, we want to simulate an online Q&A forum where some developers (the authors in this study) post questions and others (study participants) answer these questions. We design a simple Q&A system[2], which is similar to Stack Overflow. In the system, participants can view the error scenario description of an experimental task one at a time, and then identify the root cause of the error and propose a solution to fix the error scenario. An error scenario we gave the participants include: an API usage directive sentence, concerned API, the URL of the API doc containing the API usage directive sentence, the question title and body from Stack Overflow as the task to be answered, and a demystification report generated by our approach (only for experimental group). The control group are provided with only the API usage-directive sentences, while the experimental group are provided with the modified version of our generated demystification report.

We ask the participants to give answers in the way recommended by Stack Overflow [17]. When attempting an error scenario, participants can search and read online materials (e.g., API documentation, online blogs), but they are instructed explicitly to ignore the Stack Overflow questions from which the error scenarios are derived. As most of Stack Overflow questions provide only partial code [1] or involve specific development environments, we do not ask the participants to actually replicate the error scenarios and test their suggested solutions. For the effectiveness of our user study, each question can only be answered once and participants cannot go back to revise their answers once submitted. We record the task completion time for each error scenario, from the time participants start a scenario to the time they submit the answer.

5.1.3 Participants. We recruit 14 developers from an IT company that have over 2000 developers. These 14 developers have 1 to 5 years (on average 3.2 years) of Java and Android development experience on either commercial or open-source projects. Based on the development experience of these developers, we divide them into two "comparable"groups: G-1 and G-2. Each group has 7 developers. G-1 is the control group that are given only the concerned API usage directives, while G-2 is the experimental group that are given our demystification report.

5.1.4 Evaluation Metrics. We evaluate the participants' performance based on their task completion time and answer correctness. Task completion time is automatically recorded during the study and it evaluates how fast a participant can solve these error scenarios. Answer correctness evaluates whether the answer submitted by a participant is actually an appropriate solution to the question. The two authors collaboratively determine the correctness of each submitted answer by examining it against the answers to the original Stack Overflow questions and relevant API documentation. If the submitted answer to an error scenario can solve the problem, the participant get 1 mark, otherwise 0 mark. To avoid annotation

[2]http://49.233.184.42:8100

Table 7: Error Scenarios in Our User Study Tasks

NO.	Error Scenario	Concerned API	Concerned API Usage Directive	Difficulty
Q1	Android: Tint one of three icons	`AnimatedVectorDrawable.mutate()`	...especially useful when you need to modify properties of drawables loaded from resources...	Easy
Q2	observable pattern not notify in android	`observable.notifyobservers`	...then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed	Easy
Q3	Clear intent action default	`Intent.ACTION_CHOOSER`	... CHOOSER action should be used... present to the user a list of the things...	medium
Q4	Android/Java sequence of serally executed tasks	`SynchronousQueue`	...suited for handoff designs, in which an object running in one thread must sync up ...	Medium
Q5	android, matcher. appendReplacement(sb, '$8') through ArrayIndexOutOfBoundsException	`String.replaceAll`	replaceAll (repl) Note that backslashes (\ ) and dollar signs ($) in the replacement string...	Medium
Q6	How can I animate a view in Android and have it stay in the new position/size?	`R.attr.fillAfter`	When set to true, the animation transformation is applied after the animation is over	Difficult
Q7	Accessbility service description	`Manifest.permission.BIND_ACCESSIBILITY_SERVICE`	Must be required by an AccessibilityService...	Difficult
Q8	Facebook URl scheme to page post	`r.attr.scheme`	...schemes here should always use lower case...	Difficult

Table 8: Results on Task Completion Time and Answer Correctness

Metric	Group	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Average	*p*-value
Answer Correctness	G-1	0.714	0.714	0.429	0.429	0.714	**0.286**	**0.286**	**0.286**	0.482	0.014
	G-2	0.857	1.000	0.714	0.714	0.857	1.000	0.571	0.429	0.768	
	Improv.	20.03%	40.06%	66.43%	66.43%	20.03%	249.65%	99.65%	50.00%	59.34%	-
Time Consumption (min)	G-1	11.68	13.67	17.88	16.81	14.25	**18.70**	18.42	14.93	15.79	0.030
	G-2	12.35	8.74	12.48	11.59	12.27	10.47	17.59	11.12	12.26	
	Improv.	5.73%	-36.06%	-30.20%	-31.05%	-13.89%	-44.01%	-4.51%	-25.52%	-22.36%	-

biases, we mix together the answers of the two groups so the authors do not know the ground id of the answers. We use Wilcoxon signed-rank test [28] with Bonferroni correction [27] to determine if the performance difference between the control and experimental group is statistically significant at the confidence level of 95%.

5.2 Results

Table 8 shows average answer correctness and task completion time consumption for each error scenario over all 7 participants. The last two columns list the average results of each group over all 8 error scenarios and the p-value of the Wilcoxon signed-rank test on the correctness difference and the completion time difference. The results show that the answer correctness of the experimental group is statistically significantly better than that of the control group, and the experimental group participants complete the tasks statistically significantly faster.

The control group answer reasonably well only for two easy (Q1/Q2) and one medium (Q5) error scenarios. For these three error scenarios, 5 participants in the control group submitted the correct answers. The control group have the smallest correctness gap with the experimental group on these three scenarios. But the control group have poor answer correctness for the other 5 error scenarios, especially for the different error scenarios (Q6/Q7/Q8).

In contrast, the experimental group can answer the three difficult questions much better, especially for Q6. Although the error scenario in Q6 is different from the API misuse scenario in the demystification report, the objective of the two questions are very similar at the conceptual level. Thus, looking at the demystification report, the experimental group can infer the appropriate solution to the Q6 in a shorter period of time (about 10 minutes on average). The concerned API usage-directive sentence for Q6 is rather abstract "when set to true, the animation transformation is applied after the animation is over". Looking at just this sentence, it is not straightforward to derive a solution for the problem in Q6. Therefore, even though the control group spent on average about 18 minutes, only two participants submitted the correct answers.

For Q7 and Q8, the experimental group have the worse answer correctness (although still better than the control group). The reason is that the solution to a specific misuse scenario is often context sensitive. For Q7 and Q8, although the solution in the API misuse scenarios in the demystification report is related to the error scenario, the solution in the report is not directly applicable to the error scenario. Some participants in the experimental group did not realize this context sensitivity and just reuse the solution in the report as the answer to the error scenarios. Unfortunately, we do not regard this solution as the correct answer.

In conclusion, participants suggest that demystification reports help them in two aspects: summarize meaningful information scents to find a solution, and enhance their confidence on the answers. Although they had to spend time on reading the report, nobody complained about this effort. It seems that they considered the help offered by the report was worth the time spent.

6 THREATS TO VALIDITY

Threats to internal validity relate to errors in our experimental data, tool implementation and personal bias in user studies. To avoid errors in experimental data, we carefully checked our tool implementation, and manually examine a large number of data instances outputted by each step of our tool. To reduce the personal bias in the manual examination of the extracted information in the demystification reports, the two authors annotate the data instances independently and the Cohen's Kappas indicates the substantial or almost perfect agreement between the two annotators.
Threats to external validity relate to the generalizability of our demystification report. In this project, we use the data from Stack Overflow community and Android API documentation. However, there are many other Q&A sites with different data characteristics and a variety of official documentations for different languages. Further studies on other official and crowdsourced documentations are required to generalize our results. We will also release our dataset and tool for the public validation.

7 RELATED WORK

By sampling and manual analysis, Maalej and Robillard [12] conclude 12 patterns of knowledge in API documentation, including usage directives we focus on in this study. They define the directive knowledge as the key information that developers should execute or avoid when using the specific APIs. They also discover that the directives have high frequency in the API documentation, and confirm that the directive knowledge is important. Monperrus et al. [15] perform an empirical study on directive knowledge of API documents. It divide API directives into 23 types, and analyze the good and bad practices of describing API directives. It conclude that a good practice of documenting API directives is to provide clear examples, on the other hand, the bad documentation practice often involve ambiguous descriptions that generate more questions than they answer. The studies by Robillard and his colleagues [21, 22] echo this observation. However, they find that most of API documents lack the high-quality examples. Our work aims to extract high-quality API misuse scenarios from Stack Overflow to demystify official API usage directives.

Parnin et al. [18] studied the complementary nature of crowd documentation for official API documentation. Techniques [10, 11, 26] have been proposed to extract useful information such as undocumented API usage directives from Stack Overflow to augment official documentation. Li et al. [9] focuses on extracting API caveats in official documentation into a knowledge graph to improve the accessibility of API caveats. Zhou et al. [33] check the inconsistencies between the API usage-directive sentences and the corresponding API implemetation. Our goal is different from these works: we extract API misuse scenarios to augment API usage directives, but we reuse the API caveat extraction patterns in [9] to API usage directives from official documentation.

Many other studies also tap on the knowledge repository on Stack Overflow. BIKER [6] recommend code snippets on Stack Overflow for API usage queries. Subramanian et al. [25] develop a linking method to link the code element on Stack Overflow to official APIs. We adopt this technique to detect API mention in Stack Overflow text and code snippets. But these two techniques focus on general traceability recovery between code and text. In contrast, our work focuses on API-misuse related code in crowdsourced documentation and API usage-directive text in official documentation. Ren et al. [19] discover and summarize the controversial discussions on Stack Overflow. They use official API caveats to explain the discovered controversies. In contrast, our work finds API misuse scenarios to explain the API usage directives.

8 CONCLUSION AND FUTURE WORK

This paper present an approach for enhancing the understandability of API usage directives with the API misuse scenarios, erroneous code example and patches, as well as related APIs and confusing APIs mined from the community Q&A site like Stack Overflow. Our approach bridges the information gulf between the large number of API usage directives described in official API documentation, which often lack concrete examples, and the large number of API misuse scenarios discussed on Stack Overflow, which contain concrete errors caused by overlooking or misunderstanding the API usage directives. Our demystification report links the two information sources, which turns "somebody else's unfortunate mistakes" into a fortune to help developers learn the API usage directives. Our evaluation confirms the accuracy of our open information extraction method, and our user study demonstrates the usefulness of our demystification reports for debugging API-misuse related program errors with respect to the concerned API usage directives.

9 ACKNOWLEDGEMENTS

This research was partially supported by the National Key Research and Development Program of China (2018YFB1003904), NSFC Program (No. 61972339), and the Australian Research Council¿s Discovery Early Career Researcher Award (DECRA) funding scheme (DE200100021).

REFERENCES

[1] Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. 2018. Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 319–330.

[2] Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. 2010. Learning mid-level features for recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Citeseer, 2559–2566.

[3] Chunyang Chen, Zhenchang Xing, and Ximing Wang. 2017. Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 450–461.

[4] Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. 2016. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 744–755.

[5] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. 2003. XRANK: Ranked keyword search over XML documents. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 16–27.

[6] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 293–304.

[7] Tom Kenter and Maarten De Rijke. 2015. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 1411–1420.

[8] J Richard Landis and Gary G Koch. 1977. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics* (1977), 363–374.

[9] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. 2018. Improving api caveats accessibility by mining api caveats knowledge graph. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 183–193.

[10] Jing Li, Aixin Sun, and Zhenchang Xing. 2018. To Do or Not To Do: Distill crowdsourced negative caveats to augment api documentation. *JASIST* 69 (2018), 1460–1475.

[11] Jing Li, Aixin Sun, Zhenchang Xing, and Lei Han. 2018. API Caveat Explorer – Surfacing Negative Usages from Practice: An API-oriented Interactive Exploratory Search System for Programmers. In *The 41st International ACM SIGIR Conference on Research &#38; Development in Information Retrieval (SIGIR '18)*. ACM, New York, NY, USA, 1293–1296. https://doi.org/10.1145/3209978.3210170

[12] Walid Maalej and Martin P. Robillard. 2013. Patterns of Knowledge in API Reference Documentation. *IEEE Trans. Softw. Eng.* 39, 9 (Sept. 2013), 1264–1282. https://doi.org/10.1109/TSE.2013.12

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[15] Martin Monperrus, Michael Eichberg, Elif Tekes, and Mira Mezini. 2012. What Should Developers Be Aware Of? An Empirical Study on the Directives of API Documentation. *Empirical Software Engineering* 17 (05 2012). https://doi.org/10.1007/s10664-011-9186-4

[16] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q&A in StackOverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 25–34.

[17] Stack Overflow. 2019. How do I write a good answer? (2019). https://stackoverflow.com/help/how-to-answer.

[18] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Institute of Technology, Tech. Rep* 11 (2012).

[19] X. Ren, Z. Xing, X. Xia, G. Li, and J. Sun. 2019. Discovering, Explaining and Summarizing Controversial Discussions in Community Q A Sites. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 151–162. https://doi.org/10.1109/ASE.2019.00024

[20] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[21] M. P. Robillard. 2009. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software* 26, 6 (Nov 2009), 27–34. https://doi.org/10.1109/MS.2009.193

[22] Martin P. Robillard and Robert Deline. 2011. A Field Study of API Learning Obstacles. *Empirical Softw. Engg.* 16, 6 (Dec. 2011), 703–732. https://doi.org/10.1007/s10664-010-9150-8

[23] Ravindra Singh and Naurang Singh Mangat. 2013. *Elements of survey sampling*. Vol. 15. Springer Science & Business Media.

[24] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API documentation. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 643–652.

[25] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API Documentation. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 643–652. https://doi.org/10.1145/2568225.2568313

[26] Christoph Treude and Martin P Robillard. 2016. Augmenting api documentation with insights from stack overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 392–403.

[27] Eric W Weisstein. 2004. Bonferroni correction. (2004).

[28] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics*. Springer, 196–202.

[29] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 51–62.

[30] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. 2016. Software-specific named entity recognition in software engineering social content. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 90–101.

[31] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*. ACM, 404–415.

[32] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1, 1-4 (2010), 43–52.

[33] Yu Zhou, Ruihang Gu, Taolue Chen, Zhiqiu Huang, Sebastiano Panichella, and Harald Gall. 2017. Analyzing APIs Documentation and Code to Detect Directive Defects. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 27–37. https://doi.org/10.1109/ICSE.2017.11