Constructing Software Knowledge Graph from Software Text

Sirui Li
Supervisor: Dr. Zhenchang Xing

COMP8755 Individual Computing Project
Australian National University Semester 1, 2018

May 9, 2018
Acknowledgements

First and foremost, I would like to express my thanks of gratitude to my supervisor, Dr. Zhenchang Xing, who gave me the golden opportunity to do this project even though I do not have too much experience on researching. Without his assistance and guidance, the project and the report cannot be accomplished.

Secondly, I would also like to thank A/Prof Peter Strazdins and Prof Weifa Liang who are the course convener, for providing helpful assistance on academic writing and presentation.

Most importantly, my thanks go to my family and friends, without the understanding and support of them, the project cannot be finalized.
Abstract

API documentation provides important knowledge about the functionality and usage of APIs. API caveats are natural-language statement in API documentation that developers should be aware of in order to avoid unintended use of an API. However, 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. To address this issue, we propose natural language processing 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.

Keywords: API caveat, Android, API knowledge graph, Natural Language Processing
6.3 The Quality of API Caveats Knowledge Graph (RQ2) . . . . 26
6.3.1 Accuracy of Extracting API Caveat Sentences . . . . . 27
6.3.2 Accuracy of Co-reference Resolution . . . . . . . . . . 27
6.3.3 Accuracy of Caveat-Sentence-API Linking . . . . . . . 28
6.4 The Improvement of API Caveats Accessibility (RQ3) . . . 28

7 Conclusion and Future Work 33

Appendices 34

A Independent Study Contract and Project Description 35

B README 39
Chapter 1

Introduction

API documentation, such as Java API Documentation and Android Developers, is an important resource for developers to learn API usage. They provide important information on API’s directives, functionality and usage. There are many researches has been done on the quality of API documentation [17, 24, 28, 30, 37] and the traceability recovery of API documentation [11, 13, 27, 29, 34, 42].

In this report, we are concerned with a much less explored issue of API documentation, i.e., the accessibility of API documentation. We are cared about can developers effectively access the relevant API usage knowledge in these documents? In particular, we focus on the accessibility of “directives” which is a kind of knowledge types in API documentation defined by Maalej and Robillard [20]. As such API usage directives “specify what users are allowed/not allowed to do with the API element” [20]. When developers overlook them, it is very likely to cause some unexpected programming errors. To emphasize the fact that such directives are contracts, constraints, and guidelines of API usage that developers should be aware of [23], we call them as API caveats in this report.

To get a sense of the potential accessibility issue of API caveats, 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 Figure 2.1). 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 for-

1.1 Objectives

The goal of this project is to tackle the accessibility of API caveats by mining

an API caveats knowledge graph from multiple sources of API documenta-

tion.

We regard an API (e.g., class, method, field) as an entity and construct

an API skeleton graph of APIs and their declared relations 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 [5, 8, 14, 23] 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 res-
olution technique [19, 21, 25, 26] 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 sen-
tences. 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 documenta-
tion. 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.
1.2 Contributions

This report 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 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.

1.3 Outline

This report is structured as follows:

- Chapter 2 presents the formative study of some Stack Overflow questions.

- Chapter 3 presents the related work for API knowledge graph.

- Chapter 4 presents how the Android API knowledge graph constructs.

- Chapter 5 shows a proof-of-concept tool of our approach.

- Chapter 6 shows the evaluation based on the abundance of API caveats, the quality of the knowledge graph and the improvement of API caveats accessibility.

- Chapter 7 concludes this report and discusses the future work.
Chapter 2

Formative Study

In this formative study, we read 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.

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.
As summarized in Figure 2.1, 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 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 [30].

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.
Chapter 3

Related Work

There has been much research on the quality of API documentation [17, 24, 29, 31, 37, 43], which investigate 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 [15, 22, 34]. As the online Q&A websites become popular, they provide an alternative way of documentation, i.e., crowd documentation [9, 10, 38]. Studies [1, 33, 35] 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. [2, 3] develop an API extraction and linking infrastructure, called Miler. Dagenais and Robillard [13] 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. [34] use code context information to link an API mention in a partial code fragment to APIs in a knowledge base. Ye et al. [40] 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 \cite{16,36} 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 is the two empirical studies on the knowledge types in API documentation \cite{20,23}. 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 \cite{23}. However, these two studies focus on empirical observations of API knowledge in API documentation, with no specific application objectives. In contrast, our work proposes 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.
Chapter 4

Approach

Figure 4.1 presents the overview of our approach. Our approach 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 4.1), build an API skeleton graph from semi-structured API reference documentation (Section 4.2), extract API caveat sentences from API textual descriptions (Section 4.3), and construct the API-caveats knowledge graph by linking API caveat sentences to relevant APIs (Section 4.4). Based on the mined API-caveats knowledge graph, the search engine can recommend API caveats for the API(s) of interests (Section 4.5).
4.1 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, 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.

Online API documentation can be easily crawled using web crawling tools such as BeautifulSoup\textsuperscript{1} \footnote{https://www.crummy.com/software/BeautifulSoup/}. 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.

API textual descriptions have to be tokenized for further natural language processing. 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” 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. \footnote{Ye et al.} 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 to split texts into sentences.

4.2 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.

4.3 Extracting API Caveat Sentences

To extract API caveat sentences, we define a taxonomy of API caveats and develop corresponding sentence syntactic patterns. The taxonomy and syntactic patterns have been developed based on the literature survey [5-8,14,23] and our own observation of API caveats in API documentation.

<table>
<thead>
<tr>
<th>Table 4.1: API-Caveats Categories and Syntactics Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Restricted</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Generic</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
As summarized in Table 4.1, 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. Following [23], we use word stem to match different infected variants of a word, for example “assum*” for “assume”, “assumes”, “assuming”. 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.

4.3.1 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”, “non thread safe”, “illegal”, “insecure”. Typical examples include: “FileUriExposedException is thrown when an application exposes a file://Uri to another app”, “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 ...”.

**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”.

16
**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 ...”.

### 4.3.2 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”.

### 4.3.3 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”.

### 4.4 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.

#### 4.4.1 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. To address this issue, we use co-reference resolution technique (as implemented by Stanford CoreNLP\(^2\)) 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.

\(^2\)https://stanfordnlp.github.io/CoreNLP/coref.html
4.4.2 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.

**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\(^3\) 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 \(^{12}\).

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. \(^{10}\), 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

---

\(^3\)https://nlp.stanford.edu/software/openie.html
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.

4.5 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 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 issue, 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.
Chapter 5

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 6.1 for details).

![Figure 5.1: The Search UI of the API Caveats Knowledge Graph](image)

- 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.
We implement a web interface for searching API caveats in our knowledge graph (see Fig. 5.1). 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. 5.1 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.
Chapter 6

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?

### 6.1 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 4.1). 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.

6.2 The Abundance of API Caveats (RQ1)

Table 6.1: The Abundance of Different Subcategories of API Caveats

<table>
<thead>
<tr>
<th>Subcategory</th>
<th>Numbers</th>
<th>Frequency</th>
<th>Average Abundance Frequency</th>
<th>ACFOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error/Exception</td>
<td>11,973</td>
<td>0.045</td>
<td>0.100</td>
<td>R</td>
</tr>
<tr>
<td>Recommendation</td>
<td>19,209</td>
<td>0.072</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alternative</td>
<td>4,032</td>
<td>0.015</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Imperative</td>
<td>6,410</td>
<td>0.024</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>8,183</td>
<td>0.031</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Conditional</td>
<td>70,404</td>
<td>0.263</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>8,479</td>
<td>0.032</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Affirmative</td>
<td>61,952</td>
<td>0.231</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>36,136</td>
<td>0.135</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>Emphasis</td>
<td>41,113</td>
<td>0.153</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td>267,891</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 [23], we adopt the ACFOR scale (Abundant, Common, Frequent, Occasional, Rare). This scale is originated from ecology for measuring species abundance within a given area [4]. 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 \( F \geq A \), frequent if \( F \geq 0.5A \) and rare otherwise.

Results: Table 6.1 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 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.

6.3 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 [32]. According to [32], 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: $MIN = \frac{n_0}{1+(n_0-1)/\text{populationsize}}$, $n_0$ depends on the selected confidence level and the desired error margin: $n_0 = \frac{Z^2 * 0.25}{e^2}$, 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 [18] 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.
6.3.1 Accuracy of Extracting API Caveat Sentences

We perform accuracy analysis for each subcategory of API caveats. Table 6.2 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).

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.

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

6.3.2 Accuracy of Co-reference Resolution

Based on the number of API caveat sentences that require co-reference resolution and the above sampling formula, 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 for 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.

6.3.3 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-sentence-API link for the two linking scenarios respectively. The annotator determines if the link correctly associate an API caveat sentence to the corresponding API. The Cohen’s kappa metric between the two annotators 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\%.

6.4 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
Table 6.3: Ten Android API Related Questions in Our User Study

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

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 6.3 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.

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.

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

**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 [39] 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 6.4 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.

Figure 6.1: Question-Difficulty Ratings and Answer-Confidence Ratings

Fig. 6.1 compares the question-difficulty ratings and the answer-confidence
Table 6.4: Performance of Control Group versus Experimental Group

<table>
<thead>
<tr>
<th>Control Group</th>
<th>AveQCT(second)</th>
<th>AveCP(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>170.8</td>
<td>55.83</td>
</tr>
<tr>
<td>P2</td>
<td>132.3</td>
<td>32.5</td>
</tr>
<tr>
<td>P3</td>
<td>143.9</td>
<td>37.5</td>
</tr>
<tr>
<td>P4</td>
<td>63.4</td>
<td>15.83</td>
</tr>
<tr>
<td>P5</td>
<td>214.4</td>
<td>49.83</td>
</tr>
<tr>
<td>P6</td>
<td>129.2</td>
<td>54.16</td>
</tr>
<tr>
<td>Ave±stddev</td>
<td>142.3±50.02</td>
<td>40.94±15.42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experimental Group</th>
<th>AveQCT(second)</th>
<th>AveCP(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P7</td>
<td>94.3</td>
<td>57.5</td>
</tr>
<tr>
<td>P8</td>
<td>70.9</td>
<td>54.16</td>
</tr>
<tr>
<td>P9</td>
<td>89</td>
<td>73.33</td>
</tr>
<tr>
<td>P10</td>
<td>76.2</td>
<td>65.82</td>
</tr>
<tr>
<td>P11</td>
<td>124.6</td>
<td>65</td>
</tr>
<tr>
<td>P12</td>
<td>61.8</td>
<td>61.66</td>
</tr>
<tr>
<td>Ave±stddev</td>
<td>86.13±22.26</td>
<td>62.91±6.76</td>
</tr>
</tbody>
</table>

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. One 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.
Chapter 7

Conclusion and Future Work

In this report, 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 search 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 improve the accuracy of co-reference resolution and validate our approach with more API documentation and more users.
Appendices
Appendix A

Independent Study Contract and Project Description
INDEPENDENT STUDY CONTRACT

Note: Enrolment is subject to approval by the projects co-ordinator

SECTION A (Students and Supervisors)

| UnilD:  | u5831882 |
| Surname: | Li         |
| First Names: | Sirui     |
| Project Supervisor (may be external): | Zhenchang Xing |
| Course Supervisor (a RSCS academic): |                     |
| Course Code, Title and Unit: | COMP8755 12 units |

Semester: S2 2017

Project Title:
 Constructing Software Development Knowledge Graph from Software Text

Learning Objectives:
 See attached

Project Description:
 See attached
ASSESSMENT (as per course’s project rules web page, with the differences noted below):

<table>
<thead>
<tr>
<th>Assessed project components</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artefact</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MEETING DATES (IF KNOWN):
Every Monday, 4-5pm

STUDENT DECLARATION:
I agree to fulfil the above defined contract

................................................. 25/07/2017
Date

SECTION B (Supervisor):
I am willing to supervise and support this project. I have checked the student’s academic record and believe this student can complete the project. I nominate the following reviewers and have obtained their consent to review the completed thesis (through signature or attached email)

................................................. 25/07/2017
Date

Reviewer 1:
Name: ................................................. Signature.................................................

Reviewer 2:
Name: ................................................. Signature.................................................

*Nominated reviewers may be subject to change on request by the supervisor.

REQUIRED DEPARTMENT RESOURCES:

SECTION C (Course coordinator approval)

................................................. .................................................
Signature Date

SECTION D (Projects coordinator approval)

................................................. .................................................
Signature Date

Research School of Computer Science

Form updated Jan 2017
1 Project Title

Constructing Software Development Knowledge Graph from Software Text

2 Project Description

Knowledge graph integrates structural information of concepts across multiple information sources, and links these concepts together [2]. It is used in many application domains like search result ranking, recommendation, exploratory search, etc. [1]. It provides users with detailed information about a topic in addition to a list of links to other sites. However, there is no related work focusing on constructing the knowledge graph for software development from software text.

In this project, we will focus on the Android Developer website (https://developer.android.com/training/index.html). The aim of this project is to construct a knowledge graph from Android development training materials and Android API documentation on the Android Developer website using Natural Language Processing techniques. The approach proposed by Zhao et al. [1] will be adopted and extend in this project. Such knowledge graph can help developers to have an overall understanding of how to build applications for Android system and what issues are notable in each step.

SPECIFIC TASKS
1. Crawl Data from https://developer.android.com/training/index.html
2. Pre-process the data and extract key textual content from web pages
3. Learn NLP techniques and the approach proposed by Zhao et al. [1]
4. Extend Zhao et al.\'s approach [1] to construct an Android development knowledge graph from Android development training materials
5. Write up a project report

3 Learning Objectives

- Develop skills in data crawling, processing and analytics
- Have a good understanding of NLP techniques and knowledge graph
- Be able to convert unstructured text into knowledge graph
- Be able to deliver a project report of research outcomes

4 Related Work


Appendix B

README
Constructing Software Knowledge Graph from Software Text

This project proposes natural language processing 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.

Getting Started

This project was written and tested in macOS Sierra. So the commands and instructions shown in this README are in mac format.

Prerequisites

1. Mysql. Server version: 10.1.26-MariaDB was used.
2. XAMPP. It can download from https://www.apachefriends.org/download.html. Version 7.0.23 / PHP 7.0.23 for OS X was used.

Running the Search Webpage

This section introduces how to search in our webpage which is based on our API caveats knowledge graph.

1. Save folder "db1" into path "~/Applications/XAMPP/htdocs"
2. Save file "index.html" into "~/Applications/XAMPP"
3. Open manager-osx in Applications
4. Start MYSQL Database and Apache Web Server (See Known Issue if occurring problems)
5. Import the database (in folder "database") by adding tables in http://localhost/phpmyadmin/ or running the command `mysql -u root [path of the sql file]` in the terminal
6. Go to http://localhost/ to use the search tool.

Known Issue

Cannot start MYSQL Database in XAMPP

1. Change the port number. The author used 3307. And try again.
2. Run
$ sudo killall mysqld
$ sudo /Applications/XAMPP/xamppfiles/bin/mysql.server start

**Author**

Database is built by Hongwei Li.

Codes of search tool and README are written by Sirui Li (u5831882).

Mail: u5831882@anu.edu.au

**Acknowledgements**

I really appreciate the work of our team (Zhenchang Xing, Hongwei Li, Jiamou Sun).
Bibliography


