MINING STACK OVERFLOW TO RECOMMEND JAVA API CLASSES USING WORD EMBEDDING AND TOPIC MODELLING

LEE WAI KEAT

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

MINING STACK OVERFLOW TO RECOMMEND JAVA API CLASSES USING WORD EMBEDDING AND TOPIC MODELLING

LEE WAI KEAT

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SOFTWARE ENGINEERING (SOFTWARE TECHNOLOGY)

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: (I.C/Passport No:)

Matric No:

Name of Degree:

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Field of Study:

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

MINING STACK OVERFLOW TO RECOMMEND JAVA API CLASSES USING WORD EMBEDDING AND TOPIC MODELLING

ABSTRACT

To reduce development effort, today's software development technologies rely heavily on reusable components provided by Application Programming Interfaces (APIs). However, studies have found that APIs are of poor usability and programmers find it difficult to use them. A number of factors affect the usability and learning of an API. The most critical one is the API documentation. Therefore, it is unsurprising that developers look for alternative information sources to learn APIs. One such sources is the crowd documentation of APIs that are available in Community Question and Answer (CQA) websites, such as Stack Overflow (SO). Studies have shown that the large volume of data in SO make it suitable for data mining and analytics for APIs. Following that, this research aims to: 1) identify Java programmers' common Java programming problems based on their level of expertise, by analyzing Java-related duplicate discussion posts in SO (Study 1); 2) to address the lexical gap between natural language queries and Java APIs documentation, and the lexical gap between natural language queries and the Java programming codes, by designing and implementing an approach for recommending Java API classes for programmers' natural language queries using data mined from SO (Study 2). Existing studies have found that SO questions/discussion posts have a wide coverage on Java API. Java was chosen in this research as it is a long established and popular programming language. Study 1 found that the novice group is the top contributor and the expert group contributes significantly lower to duplicate questions asked in SO, and the most common problem Java programmers face is understanding and/or fixing errors but expert programmers' question more about the reasons behind some Java programming concepts. The proposed approach in Study 2 employs Natural Language Processing techniques, namely, word embedding and topic modelling, and heuristic rules

to produce the Java API classes recommendations. The benchmarking of the performance of the proposed approach against existing state-of-the-art approach using four metrics (Top-K accuracy, Mean Recall @ K, Mean Reciprocal Rank @ K and Mean Average Precision @ K) shows that the proposed approach performs better. The proposed approach was implemented in a Java API classes recommender running on a server and an Eclipse IDE's plug-in (APIRecJ) was implemented as the front-end to access the recommender's functionalities. The results of the user evaluation study show that APIRecJ is generally useful in searching for Java API classes relevant to the programmers' queries. In summary, the contribution of this research are: a set of common Java programming problems and Java API classes that Java programmers struggle with, that Java educators and learning resources can devote more attention to; an approach for recommending relevant Java API classes for programmers' queries that outperforms existing approaches; a Java API classes recommender; and an Eclipse IDE's plug-in that provides assistance on Java API classes relevant to the programmers' queries within the IDE.

Keywords: Mining Stack Overflow, Java API Class Recommender, Word Embedding, Topic Modelling

MELOMBONG STACK OVERFLOW UNTUK MENGESYORKAN KELAS API JAVA DENGAN MENGGUNAKAN PENYEMATAN PERKATAAN DAN PEMODELAN TOPIK

ABSTRAK

Untuk meringankan beban dalam membina aplikasi perisian, cara untuk membina aplikasi perisian pada zaman ini adalah menggunakan komponen yang boleh diguna semula melalui *Application Programming Interface* (API). Walaubagaimanapun, kebanyakan kajian mendapati bahawa API sukar diguna oleh pengatur cara. Terdapat pelbagai faktor menyebabkan API sukar diguna dan dipelajari. Faktor yang terutama ialah dokumentasi API. Oleh sebab itu, pengatur cara mencari sumber maklumat alternatif untuk mempelajari API iaitu melalui laman web yang berkaitan dengan komuniti soal dan jawab, seperti Stack Overflow (SO). Kajian menunjukkan bahawa jumlah data yang besar dalam SO menjadikannya sesuai untuk perlombongan data dan analisis untuk API. Dengan itu, kajian ini bertujuan untuk: 1) mengenal pasti masalah pengaturcaraan umum di kalangan pengatur cara Java berdasarkan tahap kepakaran mereka, dengan menganalisis duplikasi perbincangan yang berkaitan dengan Java dalam SO (Kajian 1); 2) menangani jurang leksikal antara pertanyaan bahasa semulajadi dan dokumentasi API, dan jurang leksikal antara pertanyaan bahasa semulajadi dan kod pengaturcaraan Java, dengan mereka bentuk dan melaksanakan kaedah untuk mengesyorkan kelas API Java untuk pertanyaan bahasa semulajadi pengatur cara menggunakan data SO (Kajian 2). Kajian dahulu telah mendapati bahawa soalan atau perbincangan dalam SO mempunyai liputan yang luas dalam Java API. Oleh itu, Java telah dipilih dalam kajian ini kerana ia adalah bahasa pengaturcaraan yang mantap dan popular. Kajian 1 mendapati bahawa kumpulan pemula adalah penyumbang utama kepada soalan-soalan yang ditanyakan dalam SO dan kumpulan pakar menyumbang jauh lebih rendah daripada kumpulan pemula. Masalah yang paling biasa dihadapi oleh pengaturcara Java adalah memahami

atau membetulkan kesilapan tetapi pengaturcara yang berpengalaman mempersoalkan lebih lanjut mengenai sebab di sebalik beberapa konsep pengaturcaraan Java. Kaedah yang dicadangkan dalam Kajian 2 menggunakan teknik pemprosesan bahasa semula jadi, iaitu, kata penyematan perkataan dan pemodelan topik, dan peraturan heuristik untuk menghasilkan cadangan kelas API. Kajian 2 juga menggunakan empat metrik (Top-K accuracy, Mean Recall @ K, Mean Reciprocal Rank @ K dan Mean Average Precision @ K) untuk melaksanakan penandaarasan prestasi terhadap kaedah yang dicadangkan dan kaedah yang sedia ada. Kaedah yang dicadangkan telah menghasilkan keputusan dengan lebih baik. Kaedah yang dicadangkan dilaksanakan dalam pengesyorkan kelas API yang berfungsi sebagai pelayan dan pemalam untuk pembangunan persekitaran bersepadu Eclipse (APIRecJ) berfungsi sebagai antaramuka untuk mengakses fungsi pelayan. Hasil kajian penilaian pengguna menunjukkan bahwa APIRecJ bermanfaat dalam mencari kelas API Java yang relevan dengan pertanyaan para programmer. Secara ringkas, sumbangan penyelidikan ini termasuk: satu set masalah pemprograman Java biasa dan kelas API Java yang didapati sukar kepada para pengatur cara Java dan memerlukan perhatian daripada para pengajar dan sumber belajar Java; satu kaedah untuk mengesyorkan kelas API Java yang berkaitan untuk pertanyaan pengaturcara; pengesyor kelas API Java; dan APIRecJ yang memberikan bantuan dengan mengesyor kelas API Java yang berkaitan dengan pertanyaan pengatur cara dalam pembangunan persekitaran bersepadu Eclipse.

Keywords: Melombong Stack Overflow, Mengesyorkan Kelas API Java, Penyematan Perkataan, Pemodelan Topik.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest appreciation to my supervisor, Dr Su Moon Ting for her supervision and assistance from early stages of this research through to the completion of this thesis. Without her insightful comments, this research would have never been accomplished.

Most importantly, I wish to express my gratitude to my parents and family for their continuous support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

API	:	Application Programming Interface
CBOW	:	Continuous Bag of Words
CQA	:	Community Question and Answer
CSV	:	Comma-Separated Value
EDM	:	Education Data Mining
IDE	:	Integrated Development Environment
IDF	:	Inverse Document Frequency
IR	:	Information Retrieval
JavaEE	:	Java Enterprise Edition (EE)
JavaSE	:	Java Standard Edition (SE)
JDK	:	Java Development Kit
JSON	:	JavaScript Object Notation
KAC	:	Keyword-API Co-occurrence
KKC	:	Keyword-Keyword Coherence
LDA	:	Latent Dirichlet Allocation
MAP	:	Mean Average Precision
ML	:	Machine Learning
MR	:	Mean Recall
MRR	:	Mean Reciprocal Rank
NLP	:	Natural Language Processing
NLTK	:	Natural Language Toolkit
PRF	:	Pseudo-Relevance Feedback
RSSE	:	Recommendation Systems in Software Engineering
SDK	:	Software Development Kit

SEDE	:	Stack Exchange Data Explorer
SO	:	Stack Overflow
SQL	:	Structured Query Language
TCP/IP	:	Transmission Control Protocol/Internet Protocol
TF-IDF	:	Term Frequency - Inverse Document Frequency

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

The use of Application Programming Interfaces (APIs) in software development projects has become inevitable. However, a number of factors affected the usability of an API, and it has been found that API documentation is the most severe obstacle faced by developers in learning and using a new API. One of the contributing reasons is the lexical gap between the programmers' expressions of their programming problem queries and the descriptions used in the official APIs documentation. The increasing popularity of Community Question and Answer (CQA) websites such as Stack Overflow (SO) (2019b) shows that developers have turned to crowd API documentation to seek help for their programming problems.

Following that, this research leverages SO's crowd documentation of Java API to investigate what are the most common programming problems faced by Java programmers, and to address the issue of lexical gap between natural language queries and Java API documentation, and lexical gap between the natural language queries and the programming codes, by developing an approach that employs Natural Language Processing (NLP) techniques (in particular, word embedding and topic modelling) to recommend Java API classes for the developers' programming queries.

This chapter presents the background that motivates this research, problem statement, research objectives, research questions, and scope of the research. It also outlines the research methodology, research contributions and the remaining chapters of this thesis.

1.1 Background and Motivation

The proliferation of computers and mobile devices has opened up programming to the mainstream. Nowadays, anyone interested in developing software can learn programming by using resources available online. To reduce development effort, today's software development technologies rely heavily on reusable components provided by Application Programming Interfaces (APIs) (Robillard, 2009). APIs include frameworks, libraries, toolkits and software development kits (Myers & Stylos, 2016). The core advantage of using APIs is developers could reuse or extend code done by others without the need to start from scratch (Myers & Stylos, 2016)

There are a lot of publicly available API resources for programmer, such as Java Software Development Kit (JDK) which contains the official APIs for the Java programming language. As JDK is continually being developed and improved from time to time, the Java API has become larger and more diversified as more features are included in the newer versions. For example, JDK 6 has 3793 classes, JDK 7 has 4024 classes and JDK 8 has 4240 classes (2010) . This could result in poor API usability, which means, the API is difficult to use (Robillard, 2009).

API usability not only related to the learnability of APIs unfamiliar to developers but also includes providing the appropriate functionality and ways to access it (Myers & Stylos, 2016). A few studies had investigated the reasons of why APIs are difficult to use and identified what could be done to address the issues. For example, there are studies on API usability and API learning obstacles (Myers & Stylos, 2016; Robillard, 2009; Robillard & Deline, 2011).

There is a variety of factors that impact API usability: the complexity of the API, naming convention, support of caller's perspective, documentation, API consistency and so on (Zibran, Eishita, & Roy, 2011). The complexity of an API is related to its size; the

larger the size, the higher the complexity and the lower the usability. In terms of API naming convention, descriptive names are preferred over abbreviated names (Zibran et al., 2011). To support caller's perspective, API should always explicitly show how to invoke functions or features. As for the API documentation, it should always be clear, complete and up to date. At the same time, API should be designed consistently by adhering to common conventions.

Among the aforementioned factors, API documentation plays an essential role in API usability. A survey conducted by Robillard and Deline (2011) found that the most severe obstacle faced by developers learning a new API is the API documentation. This could be due to most programmers learn APIs by reading the corresponding documentations, but these documentations have a number of limitations in supporting the learning of APIs: insufficient examples, incomplete content, lack of reference on how to use the API to achieve specific tasks, not in desired format, lack of documentation on high-level aspects of the API such as design or rationale (Robillard, 2009).

Due to the limitation of APIs documentation, it is unsurprising that developers look for alternative information sources to learn APIs. One of the sources is Community Question and Answer (CQA) websites, such as Stack Overflow (SO) (Parnin, Treude, Grammel, & Storey, 2012).

1.2 Problem Statement

Using API is not only a difficult task for novice programmers, even experienced developers could find it difficult (Myers & Stylos, 2016). In the area of APIs documentation, there exist three types of gaps or mismatches between the programmers' expressions of their programming problem queries and the descriptions used in official

APIs documentation: the lexical gap between the programmers' natural language queries and the APIs documentation, the lexical gap between the programmers' natural language queries and programming code, and the Task-API knowledge gap.

Generally, being new to a programming language, programmers might not know the right terms to use to search for relevant API elements (such as class, interface, or method) from official API documentations. This could be due to a lexical gap or mismatch between the terms programmers use in their natural language queries (English language) and the terms used in API documentations (programming languages) (Ye, Shen, Ma, Bunescu, & Liu, 2016). Since different terms or words could be used to express the same meaning, the terms programmers use to search could be different from the terms used in API documentations even though both are referring to the same thing, causing a futile search.

Moreover, Java programmers with insufficient knowledge in programming terminology such as API classes possibly could not describe their programming problems properly. They spend a lot of effort in searching for explanations for unknown terminologies and explanations for exceptions or error messages to solve their program errors (Xia et al., 2017).

Besides learning about APIs from APIs documentation, several studies have discovered that programmers often spend their time in searching for reusable code examples by using web search engines or code search engines (Bajracharya & Lopes, 2012; Rahman, Roy, & Lo, 2018; Xia et al., 2017). One contributing reason is APIs documentation do not provide sufficient code usage examples (Parnin et al., 2012).However, traditional web search engines or code search engines usually perform poorly with natural language queries that use natural terms only, compare to queries that use code terms (Bajracharya & Lopes, 2012). A term is a natural term if it contains only

the alphabets from the English language and it can be found in a dictionary of English words, whereas, code terms also contain numerical symbols or other symbols and could not be found in a English dictionary (Bajracharya & Lopes, 2012). This could be regarded as a lexical gap between natural language queries and programming source code.

Besides that, there exists a Task-API knowledge gap between programmers' task descriptions and official API documentations (Huang, Xia, Xing, Lo, & Wang, 2018). The API documentations focus on describing the API structures and functionalities and leave out the information on their purposes which could be matched to the programmers' tasks descriptions to return API elements relevant to the tasks.

1.3 Research Questions

The research questions (RQs) for this research are:

RQ1: What are Java programmers' common Java programming problems?

RQ2: How to design an approach that recommends relevant Java API classes for Java programming questions by mining discussion posts in SO?

RQ3: What is the performance of the approach?

RQ4: How to develop a plug-in for an Integrated Development Environment (IDE) to serve as the front-end that interact with the Java API class recommender?

RQ5: How useful is the plug-in?

1.4 Research Objectives

This research aims to identify the common programming problems faced by Java programmers, and to address the first two lexical gaps mentioned in the previous section, namely, the lexical gap between natural language queries and Java APIs documentation, and the lexical gap between natural language queries and the Java programming codes, by using data mined from Stack Overflow (SO). The reasons of choosing Java and SO are given in Section 1.5.

The specific research objectives (ROs) are:

RO1: To identify Java programmers' common Java programming problems based on their level of expertise, by analyzing Java-related duplicate discussion posts in SO.

This objective involves mining and analyzing duplicate Java questions/ discussion posts in SO. Since duplicate questions in SO are in fact the same questions that different programmers repeatedly asked in different contexts, they can be used as surrogates to common questions asked in SO. These common questions are in fact common Java programming problems that Java programmers struggle with. The level of expertise is determined based on the askers' reputation scores in SO.

RO2: To develop an approach that recommends relevant Java API classes for Java programming questions by mining discussion posts in SO.

The proposed approach employs NLP techniques, namely, word embedding and topic modelling, on discussion posts extracted from SO to produce the Java API classes recommendations. The proposed approach is implemented in a Java API class

recommender to recommend Java API classes in the form of a ranked-list for the respective user's query or programming question described in natural language.

RO3: To evaluate the performance of the approach.

Performance metrics from other existing studies are employed to evaluate the performance of the proposed approach.

RO4: To develop a plug-in for an IDE to serve as the front-end to the proposed approach implemented in the Java API class recommender.

The IDE's plug-in serves as the front-end that provides a user interface for programmers to interact with the Java API class recommender.

RO5: To evaluate the usefulness of the plug-in.

A user evaluation study where participants were recruited to use the plug-in was used to evaluate the usefulness of the plug-in.

1.5 Scope of Research

Java programming language is an object-oriented, class-based and architecture neutral programming language which works similarly to C and C++ programming languages but less complicated (Gosling & McGilton, 1995). In the 2019 SO developers survey, Java was ranked the 5th most popular technology, C was ranked the 11th and C++ was ranked the 9th (2019a). Since 2008, the number of Java questions been asked in SO yearly is also higher compared to C and C++ questions (2019c). In addition, Java programming language is a long established and a popular programming language for many years (2019).

CQA websites have gained a lot of popularity recently due to the development of web technologies that improve the interactions between Internet users. A popular example is Stack Exchange (2019c), which is a network of over a hundred of CQA websites covering different topics. SO is the earliest website created in Stack Exchange network in 2008 and has become the most popular computer programming related website. Since its inception, SO has been providing a knowledge sharing platform between inexperienced programmers and experienced programmers through the asking and answering of numerous programming questions. As of 6 July 2019, SO has 18 million questions asked with 71% answered, 27 million answers, 11 million registered users, 9.2 million visits per day, and a traffic of 6.2k questions asked daily (2019d). The massive volume of crowd-generated data in SO makes it a suitable repository for data mining and analytics of crowd documentation of APIs (Parnin et al., 2012).

Some existing studies have shown that SO questions/posts have a wide coverage on Java API. The study by Parnin et al. (2012) shows that questions in SO have covered 77% of the total Java API classes. Furthermore, a recent work by Rahman, Roy, and Lo (2016) shows that about 65% of the classes from each of the 11 core Java API packages of Standard Java Edition 6 (2019) were used in Java posts in SO. The core Java API packages are: *java.lang, java.util, java.io, java.math, java.nio, java.applet, java.net, java.security, java.awt, java.sql* and *javax.swing*.

The popularity of the Java programming language, the huge number of Java questions available in SO and the wide coverage on Java API in SO posts, motivated this research to focus on the Java programming language and on mining Java questions and associated Java API from SO.

1.6 Research Methodology

This section gives a brief explanation of the research methodology adopted in this study. The full details can be found in Chapter 3. This research started with a literature review on APIs documentation usability, limitations of the official or conventional APIs documentation, crowd documentation and its benefits, existing work on finding common Java Programming problems, recommendation systems in Software Engineering, existing work on API recommenders. Besides that, the review also included data mining in SO and popular data mining techniques.

From the research gaps identified from the literature review, the research objectives and research questions were formulated. To address RO1, duplicate Java questions in SO were mined and analyzed to identify Java programming questions that were repeatedly asked as this would give insights to common Java programming problems faced by Java programmers.

To address RO2, an approach that recommends relevant Java API classes for programming questions or queries described in natural language, was designed and implemented by making use of NLP techniques and discussion posts extracted from SO. The performance of the proposed approach was benchmarked (RO3) against existing work by using four established metrics, namely, Top-10 Accuracy, Mean Recall @ 10 (MR@10), Mean Reciprocal Rank @ 10 (MRR@10) and Mean Average Precision @ 10 (MAP@10).

Subsequently, a plug-in for Eclipse IDE was developed to serve as the front-end to the proposed approach implemented in the Java API class recommender (RO4). The usefulness of the plug-in was evaluated in a user evaluation study (RO5).

The results of the benchmarking of the proposed approach and the user evaluation of the plug-in were analyzed and findings were reported. In addition, the common Java Programming problems identified were also reported.

1.7 Research Contributions

The key contributions of this research are:

1. Common Java programming problems encountered by programmers of different levels of expertise, identified from SO, the most popular computer programming related website. In addition, the top Java API classes related to these common Java programming problems were also found. These provide insights on common Java programming problems/topics and Java API classes that Java programmers struggle with. Java educators and learning resources can devote more attention to these areas (for example, understanding and fixing errors) to help learners in picking up the required knowledge and skills.
2. An approach that employs heuristic rules, word embedding and topic modelling techniques in recommending relevant Java API classes for Java programming questions described in natural language was developed. The approach outperforms existing approaches in terms of four performance metrics, by achieving 84.83% in Top-10 Accuracy, 0.58 in MRR@10, 50.68% in MAP@10 and 58.76% in MAP@10. These results demonstrate that the proposed approach has improved the existing state-of-the-art approach by 3.22% in Top-10 Accuracy, 0.03 in MRR@10, 2.83% in MAP@10 and by 0.89% in MR@10.
3. A Java API class recommender that incorporated the proposed approach was developed.

4. A plug-in for Eclipse IDE that serves as front-end to the Java API class recommender was developed. The use of this plug-in when writing Java programs in Eclipse IDE allows the programmers to describe their Java programming problems in natural language and search for Java API classes that are relevant to the programming problems and view similar questions that have been asked in SO. All these actions can be performed within the Eclipse IDE without leaving the IDE. The user evaluation of the plug-in shows that it is a useful tool for programmers particularly in answering questions that search for relevant Java API classes.

1.8 Thesis Organization

This thesis consists of eight chapters. The remaining of this thesis is structured into the chapters described below:

Chapter 2 presents the literature review performed by this research. This includes APIs documentation usability, limitations of the official or conventional APIs documentation, crowd documentation and its benefits, existing work on finding common Java Programming problems, recommendation systems in Software Engineering, existing work on API recommenders. Besides that, this chapter also provides an overview of data mining in SO and popular data mining techniques.

Chapter 3 gives the full details of the research methodology adopted in this research. It describes the key steps conducted in this research to achieve the research objectives, and how the key steps were conducted in terms of techniques, tools, technologies used, where applicable.

Chapter 4 presents the study on the common Java programming problems by mining discussion posts from SO (Study 1). It articulates the relevant database structure in SO, data extraction, and analysis conducted to identify the common Java programming problems. It also includes the results and comparison with related work.

Chapter 5 describes the proposed approach for recommending Java API classes. It describes the two phases involved, with the steps in each phase, and the techniques employed. This chapter also explains the implementation of a Java API class recommender that incorporates the proposed approach. The Java API class recommender runs on a server and serves as the back-end that processes a programmer's query and returns the recommended Java API classes. It also describes the implementation of an Eclipse's plug-in that serves as the front-end to the back-end recommender. Take note that Study 2 refers to all the things related to the proposed approach (the approach itself, its benchmarking, the Java API classes recommender and plug-in developed, and the user evaluation study conducted to evaluate the usefulness of the plug-in).

Chapter 6 presents the benchmarking of the proposed approach against existing approaches. It explains the four performance metrics used, the benchmarking results, and comparison with existing studies.

Chapter 7 describes the user evaluation study conducted to evaluate the usefulness of the plug-in. It includes the design of pilot study and user evaluation study, the results of both and discussion of user evaluation results.

Chapter 8 presents the conclusion of this research. It includes the answering of the research questions, a revisit of research contributions, threats to validity of the results, and outline possible future work.

CHAPTER 2: LITERATURE REVIEW

Studies have shown the poor usability of APIs and found API documentation to be one of the main obstacles that affects API usability. The literature review begins with a critical review on the inadequacies of conventional API documentation and highlights how a recently-emerged API documentation style, known as, crowd documentation, can complement the conventional API documentation. Besides that, the literature review also includes existing work on finding common Java Programming problems, recommendation systems in software engineering, existing work on API recommenders, an overview of data mining in SO and popular data mining techniques, and summarizes the gaps found.

2.1 Usability of APIs Documentation

An API documentation is a software documentation that is designed by a small group of people for many potential users to refer and learn about how to use the API (Parnin et al., 2012). Generally, it is not an easy task to maintain API documentation as the documentation contains highly-structured information. If software engineers do not update documentation in a timely manner, information in API documentation would most likely become stale or obsolete (Parnin et al., 2012). Sometimes, API documentation could contains incorrect information such as minor typos and incorrect description, for instance inconsistencies between a function description and what actually the function does (Zibran et al., 2011).

Moreover, it has been found that developers face a number of obstacles when learning new APIs, such as obstacles related to learning resources, API structure, developer background, technical environment or process (Robillard & Deline, 2011).

Among these obstacles, learning resources is the most severe obstacle and factor that needs to be considered when designing API documentation (a type of learning resource) have been identified: documentation of intent, code examples, matching APIs with scenarios, penetrability of the API, and format and presentation (Robillard & Deline, 2011).

The first factor, documentation of intent, requires that API documentation includes information about the rationale behind API design decisions, and how the API is supposed to be used as intended by the API designers. The second factor is the complexity of code examples given in API documentation. Small examples showing API usage patterns that involves more than one method call will be more useful than single-call examples as the formers show how methods/classes can be used together to achieve specific goals. Examples should also demonstrate "best practices" for using an API.

The third factor is matching task scenarios with specific API elements that support the scenarios. For example, "drawing a circle on the screen" scenario should be matched with the exact method that draws a circle on the screen. The fourth factor, penetrability, requires that the internal working of API to be made transparent or penetrable to the developers (such as, methods that perform multiple high-level tasks in a single operation, and the performance aspect), while maintaining certain opacity through encapsulation and information hiding.

The last factor is the documentation format. API documentation with insufficient information, trivial examples showing a single method call, over emphasis on member-level completeness rather than conceptual level, fragmented collections of hyperlinked pages versus coherent continuous documents, make it an undesirable resource to learn the API.

It is important to note that the factors above were derived from the obstacles that developers faced when learning new APIs using APIs documentation. This shows that there are weaknesses in APIs documentation.

2.2 Crowd Documentation of APIs

CQA websites create a socially-mediated form of software documentation, namely, crowd documentation, which "is a collection of web resources, where a large group of contributors, the crowd, curate and contribute to the collection." (Parnin et al., 2012, p. 3). Parnin et al. (2012) found that APIs documentation are often lack of examples and explanations. In contrast, crowd documentation of APIs has the advantages over official APIs documentation because: many code examples and explanation on API elements, numerous questions asked that lead to the same API elements, different opinions on the different solutions, votes on answers and questions, and tags for searching (Parnin et al., 2012). An API element refers to "a named entity belonging to an API, such as a class, interface, or method" (Parnin et al., 2012, p. 4).

Using crowd documentation of APIs could lower API learning curve as it is able to complement the insufficient API usage examples provided in APIs documentation. This is because crowd documentation such as SO contains knowledge that is written by many and read by many, for example, a question asked in SO could be answered by many people (Parnin et al., 2012). The mechanism of crowd documentation relies on minimal contributions from individual through social media, such as asking a question or answering a question.

The programming questions created in SO are significantly diverse and covering different types of topics and technologies, for example, programming problems in

different languages, software algorithm, software tools and so on (2019i). An existing study has performed topic modelling analysis on SO questions to find insights into what aspects of programming are difficult to understand (Allamanis & Sutton, 2013). Besides that, several studies have indicated that the large volume of data in SO make it suitable for data mining and analytics for APIs (Rahman et al., 2016; Rigby & Robillard, 2013; Subramanian & Holmes, 2013). One supporting reason is SO discussion posts contain a large amount of high quality source code snippets (Subramanian & Holmes, 2013).

In fact, some of the existing studies have shown that SO questions/posts have a wide coverage on Java API. The study by Parnin et al. (2012) shows that questions in SO have covered 77% of the total Java API classes. Furthermore, a recent work by Rahman et al. (2016) shows that about 65% of the classes from each of the 11 core Java API packages of Standard Java Edition 6 (2019) were used in Java posts in SO. The core Java API packages are: *java.lang, java.util, java.io, java.math, java.nio, java.applet, java.net, java.security, java.awt, java.sql* and *javax.swing*. The findings from the two studies mentioned in this paragraph show that SO is the right place for this research to mine for questions related to Java API.

2.3 Common Java Programming Problems

Existing studies that focus on Java programming problems are mainly targeted at students who are enrolled in introductory programming subject or novice programmers who started to learns Java programming (Hristova, Misra, Rutter, & Mercuri, 2003; Mow, 2012). These existing studies found that the most common Java programming problems faced by students are closely linked to programming errors such as syntax error(for example, confusion in using the assignment operator) (Hristova et al., 2003). Moreover,

the syntax error could probably lead to another common programming error, which is, "Variable not found", where students fail to declare a variable (Mow, 2012).

Other than the studies that focus on novice programmer's Java programming problems, there is a recent study on common Java programming problems that focuses on secure coding practice. The study investigated what are popular security features being frequently asked, common obstacles that prevent developers from implementing secure code and common security vulnerabilities in Java programming (Meng, Nagy, Yao, Zhuang, & Arango-Argoty, 2018). Another study focuses on common problems in using cryptography Java API. This study identified the common cryptography tasks developers performed and the reasons developers having difficulties in using cryptographic algorithms correctly (Nadi, Krüger, Mezini, & Bodden, 2016).

In addition, other existing work that focus on finding common problems related to subjects taken by students (such as design patterns, software architecture, and so on) and programming problems by mining SO can be found in Section 2.6.

The review of the existing work on common Java programming problems shows that there is limited work in this area, particularly in terms of core Java API usage and the common Java programming problems faced by Java programmers of different levels of expertise. This research aims to address this in the first research objective.

2.4 Recommendation Systems in Software Engineering

Recommendation Systems in Software Engineering (RSSE) refers to software applications that recommend valuable information items for a software engineering task in a given context (Robillard, Walker, & Zimmermann, 2009). RSSE helps in many kinds of software developer activities, from code reuse to bug reporting (Robillard et al., 2009).

This is mainly because software engineering domain has a large information space comprising different sources such as project source code, project history, communication archives and others, causing software developers to spend a lot of time in searching for relevant information (Robillard et al., 2009).

2.4.1 Recommended Items

RSSE equipped with data mining techniques is popular and effective as many different types of information items can be recommended. Some of the information items (Robillard et al., 2009) are:

Source Code within a Project: A recommender can help a developer in navigating the source code in one's own project, such as by predicting which parts of the source code the developer would like to reuse or view, or by assisting in completing code by recommending methods that have been defined in the project.

Reusable Source Code: A recommender can assist a developer to discover inherently reusable API elements (such as classes and methods) that can be used to complete a task, by ranking results containing API elements that are relevant to the developer's task.

Code Examples: A recommender can also return reusable source code examples or snippets that match a developer's requirements, to demonstrate the correct usage of API elements.

2.4.2 Steps in RSSE design

Generally, there are four major steps in designing RSSE: pre-processing of data, capturing of context, producing the recommendations, and presenting the recommendations (Robillard et al., 2009).

Data pre-processing is the step used to convert the raw data retrieved from data sources into a standardized format, for example, replacing missing values and detecting outliers. In designing a recommender system that uses the posts in SO, a lot of irrelevant information existed in the posts that are expressed in natural language and this has to be removed.

Capturing of context is the step used to extract task information from a user query in order to produce the recommendation. For example, capturing the user's intent of looking for specific source code example or API elements.

Producing recommendations involves executing the recommendation algorithms to select and recommend the more relevant instead of the less relevant items.

Presenting recommendations is the step used to summarize and present the recommendations to the user, for example, in the form of a ranked list of items based on the user's potential interest.

2.5 Existing studies on API Recommender

This section reviews existing studies on API recommender that aim at improving API usability. These studies can be classified into four categories: API elements search, API documentation navigation, API discoverability and API invocation. It is important to note that some of these studies used the same term to refer to their approaches and the

corresponding recommender tools that they built that incorporated the approaches. For example, in Rahman et al. (2016), the same term "*RACK"* was used to refer to the approach and also the recommender tool.

2.5.1 API Elements Search

API elements search focuses on recommending API elements from code examples or snippets. Rahman et al. (2016) first discovered the limitation of existing search engines in searching for code examples and developed an API recommender called *RACK*. *RACK* provides Top-1 search or Top-K search on API classes that match a user's query (Rahman et al., 2016). In a more recent study, the creators of *RACK* focused on reformulating a user's query with relevant API keywords for a better code search result and produced *NLP2API* tool (Rahman et al., 2018). Another recent study in this category is Huang et al. (2018) that focused on recommending API methods and summarizing output with supplementary information such as API descriptions and code examples that are related to a user's query. They produced an API recommender for their study, named *BIKER*.

2.5.2 API Documentation Navigation

Treude, Robillard, and Dagenais (2015) is the first study that aimed to produces a new documentation structure by conceptualizing tasks as specific programming actions that have been described in the documentation. They developed a prototype named *TaskNavigator* that extracts development tasks from software documentation automatically and provides assistance in navigating API documentation. A field study proved that *TaskNavigator* provides meaningful and helpful solution to developers

compared to conventional API documentation with web page links that describe everything from design philosophies to the APIs (Treude et al., 2015).

Similarly, there is another study, Zhu, Hua, Zou, Xie, and Zhao (2017) aimed to generate task-oriented API learning guide by using discussion threads and source code in SO. Their study used a similar conceptual technique as in *TaskNavigator* but differed slightly by categorizing the extracted tasks in hierarchy order with a tool named *APITasks*.

2.5.3 API Discoverability

API discoverability studies differs from API elements search studies in the sense that the former aim at discovering patterns in code when a user is performing the coding and suggesting relevant API elements, whereas the latter require a user's query as the input to perform searching for and returning of relevant API elements.

Santos and Myers (2017) focused on discovering design patterns in the source code to provide assistance on how to use API. Their study used information related to design pattern and code completion mechanisms (Santos & Myers, 2017). They produced a code completion Eclipse plug-in named *Dacite* for the discovery of API elements (Santos & Myers, 2017). *Dacite* complements APIs with design annotations, which document design decisions for API types, methods, and parameters. Java annotation refers to "marker which associates information with a program construct, but has no effect at run time" (Gosling, Joy, Steele, Bracha, & Buckley, 2014, p. 310). By using *Dacite*, developers would be able to discover and use the common design patterns suggested by the tool. The result of their user study showed that *Dacite* helps programmers to accomplish given tasks in shorter time (Santos & Myers, 2017).

Besides that, (Ichinco, Hnin, & Kelleher, 2017) tried to address the issue of novice programmers' frequent unawareness of available API methods. They developed a prototype named *Example Guru*, which is a tool that suggests context-relevant API methods by inspecting programmers' code written when using Looking Glass API, a block-based programming language. *Example Guru* was tested with novice programmers and showed promising results, for example, novice programmers who used it learned to use more API methods compared to those who did not use it. However, the authors noted that the use of hand-coded rules to check for code changes in order to suggest API information makes it non-scalable.

2.5.4 API Invocation

Zamanirad, Benatallah, Barukh, Casati, and Rodriguez (2017) implemented robotic automatic processing that is able to understand natural language user expressions and perform API invocations in RESTful programming language. They developed *BotBase* that converts natural language user expressions into API invocations. *BotBase* enables beginners learn how to perform application development without any prior programming knowledge. The bot, which is the processor of *BotBase,* is able to identify the most relevant user's intention from the user's input expression and select the APIs that meet the user's requirement (Zamanirad et al., 2017). However, the bot has two limitations. It can only invoke a single API call at a time, meaning it cannot execute dynamic process workflow that calls a series or combination of API calls. Secondly, it is a conversational bot that works in stateless environment. This indicates that the bot will not have a record of API calls that have been invoked in previous conversation.

2.5.5 Summary of API Recommenders

Table 2.1 summarizes the corresponding API recommender tools produced by the studies reviewed in the previous section in terms of the type of API language supported, SO mining and whether the recommenders was developed as IDE plug-ins or standalone applications. Most of these API recommenders were developed for the Java programming language and they were often developed as a standalone application rather than a plug-in for an IDE. Half of these API recommenders mined and used data from SO for recommendation purpose. API elements search tool (such as *NLP2API, BIKER, RACK*) are the most relevant to this research because these API recommenders also focus on mining information from SO and recommending API elements (such as API classes or API methods) to the users. As mentioned earlier, it is important to note that some of these studies used the same term to refer to their approaches and the corresponding recommender tools that they built that incorporated the approaches. For example, in Rahman et al.'s study, the same term "*RACK"* was used to refer to the approach and also the recommender tool (Rahman et al., 2016).

Table 2.1 : Summary of API Recommenders

	Name of API recommender	Category	IDE Plug-in	API Language	Mining from SO (Yes/No)
1	***RACK*** (Rahman et al., 2016)	API Elements Search	✓ Eclipse	Java	Yes
2	***NLP2API*** (Rahman et al., 2018)	API Elements Search	X	Java	Yes
3	***BIKER*** (Huang et al., 2018)	API Elements Search	X	Java	Yes
4	***TaskNavigator*** (Treude et al., 2015)	API Documentation Navigation	X	Python	No
5	***APITasks*** (Zhu et al., 2017)	API Documentation Navigation	X	Java	Yes
6	***Dacite*** (Santos & Myers, 2017)	API Discoverability	✓ Eclipse	Java	No
7	***Example Guru*** (Ichinco et al., 2017)	API Discoverability	X	Looking Glass	No
8	***BotBase*** (Zamanirad et al., 2017)	API Invocation	X	RESTful	No

2.6 Data Mining in SO

The term data mining means "the process of discovering interesting patterns and knowledge from large amounts of data" (Han, Pei, & Kamber, 2011, p. 8). As SO community is growing larger and generating vast number of users' data, it has become a valuable source for data mining.

In recent years, there are studies that aimed to discover trends or insights from SO data by using different types of techniques, such as statistics, machine learning, information retrieval (IR), pattern recognition and others (Ahasanuzzaman, Asaduzzaman, Roy, & Schneider, 2016; Ahasanuzzaman, Asaduzzaman, Roy, & Schneider, 2018; Joorabchi, English, & Mahdi, 2016). The main purposes of data mining are characterization and discrimination; mining of frequent patterns, associations, and correlations; classification and regression; clustering analysis; and outlier analysis (Han et al., 2011).

This section reviews some of the existing studies that utilize data mining for characterization and discrimination, mining of frequent patterns, associations, classification and regression, clustering analysis, and outlier analysis.

2.6.1 Characterization and Discrimination

There are many classes of data within data entries, for example, in SO, there are classes of discussion topics related to different programming languages. Thus, data characterization and discrimination are used to derive concise and precise classes within the data. Data characterization aims to summarize the general characteristics or features of a target class of data; whereas data discrimination is referring to the comparison of the

general features of the target class with one or a set of comparative classes (Han et al., 2011).

There is a study that focuses on finding the characteristics of SO posts by using tags since tags could reveal the topics covered in SO. Their findings include what kinds of questions have been asked, the most used tags, and, number of answer per question (Treude, Barzilay, & Storey, 2011).

Apart from that, there is also Education Data Mining (EDM) study that mines SO to discover subject-related difficulties, for example, calculating the frequency of questions related to software design patterns, software architecture, computer network security problems and others (Joorabchi et al., 2016). Analyzing these discussion posts in SO community could reveal interesting insight on common problems faced by both experienced and novice programmers (Joorabchi et al., 2016). Furthermore, mining the vast amount of user-generated data in SO community able to provide educators an in-depth look on the challenges faced by programming learner and address any gaps in their teaching (Joorabchi et al., 2016). Joorabchi et al. (2016) has performed text mining in SO to retrieve frequently-asked topics and categories in computer programming. The study listed the difficult topics in learning programming that require more attention, for example the top 3 highest occurrence topics in SO are “Same-origin policy”, “SQL injection” and “Model–view–controller” (Joorabchi et al., 2016). Hence, the result from text mining in SO can also be used as supplementary material to enhance students’ programming learning process by educators.

2.6.2 Mining of Frequent Pattern, Associations, and Correlations

Frequent patterns are patterns that occur frequently in data and often lead to association and correlations relationships within the data (Han et al., 2011). The main purpose of mining frequent patterns is to identify frequent co-occurrence patterns of a set of items from dataset. For example, milk and bread are frequently purchased together in grocery stores, and therefore, there is a strong association relationship between milk and bread. On the other hand, correlations measure the strength and confidence level of the association relationship.

Mining SO to discover frequent patterns can be seen in API elements search tools. For example, *RACK* mines associations and correlation patterns from SO data, where it extracts word tokens from the SO users' questions and associates these tokens to API candidates to form a 'Token-API' pair (Rahman et al., 2016). *BIKER* also mines for association patterns in SO, and calculates the similarities between users' queries and SO questions (Huang et al., 2018).

2.6.3 Classification and Regression

Classification means "the process of finding a model (or function) that describes and distinguishes data classes or concepts" (Han et al., 2011, p. 18). The process of classification includes building a model from a set of training data where the data is labelled with known classes and the built model is used to predict data with unknown class. The main difference between classification and regression is classification predicts the categorical labels for item, while a regression model predicts continuous values, for example, numerical data values rather than discrete class labels (Han et al., 2011).

A study has performed classification in SO to identify duplicate posts as this could help to automatically detect repeated questions or problems asked by SO community (Ahasanuzzaman et al., 2016). A more recent study classified discussion posts in SO in terms of API issues, such as, documentation errors, backward incompatibility, incompatibility of the APIs with underlying operating systems, and so on (Ahasanuzzaman et al., 2018).

2.6.4 Clustering Analysis

Cluster analysis is the process of partitioning a set of data objects into subsets by using clustering algorithm and leads to the discovery of previously unknown groups within the data (Han et al., 2011). An existing study, (Allamanis & Sutton, 2013) applied Latent Dirichlet Allocation (LDA) on SO data and found interesting clusters (word co-occurrences), such as general topics, problem-specific topics and topics related to specific technologies, for example, topics in Java programming language.

2.6.5 Outlier Analysis

Outlier refers to "data object that deviates significantly from the rest of the objects, as if it were generated by a different mechanism" (Han et al., 2011, p. 544). There are various approaches in outlier detection including statistical methods, proximity-based methods, clustering-based methods and classification-based methods. A prior study, (Xia, Lo, Correa, Sureka, & Shihab, 2016) performed outlier analysis to detect poor quality questions in SO, such as off-topic questions that ask questions not related to programming or software engineering activities.

2.7 Data Mining Techniques

Data mining incorporates different types of techniques from other domains such as statistic, information retrieval (IR), machine learning (ML) and others (Han et al., 2011). Among these techniques, IR is one of the essential techniques to find relevant information from vast amount of data by using probabilistic approaches such as building a language model and building a topic model (Han et al., 2011). Language model is a probability function that calculates word occurrence in documents, while topic model is a probability function that calculates topics distributed over the vocabularies in documents (Han et al., 2011).

Besides IR techniques, machine learning becomes widely used in data mining research as it allows a program to learn complex patterns automatically and make intelligent decision based on the input data. For example, a computer program would be able to recognize and label the duplicate questions after learning from a set of existing questions. Neural network is one of the prominent approaches in machine learning for data classification (Han et al., 2011). It is a collection of connected neuron-like processing units with the advantages of high tolerance of noisy data and a great ability in classifying patterns.

In recent years, researchers incorporate IR techniques with machine learning techniques to train better language model and topic model. In the area of API recommenders using SO data, *BIKER* and *NLP2API* tools were built using IR techniques incorporated with machine learning techniques (Huang et al., 2018; Rahman et al., 2016; Rahman et al., 2018). This could be due to training model using machine learning algorithm enables automatic learning of word similarity between user queries and questions in SO. During the recommendation phase, these API recommenders would be

able to select and retrieve a set of similar questions based on the similarity score calculated.

The next section reviews two popular IR approaches that incorporate machine learning techniques, namely, word embedding and topic modelling.

2.7.1 Word Embedding

Traditional IR method such as one-hot encoding is commonly used for building language model (Zhang et al., 2016). One-hot encoding transforms terms in numeric representation. For example, given two sentences, “I enjoy playing TT” and “I like playing TT”, Figure 2.1 shows the unique words and one-hot encoding for both sentences (Ayyadevara, 2018). One-hot encoding often creates a high-dimensional vector when the vocabulary size is big (where the number of unique words is large). This subsequently leads to a vocabulary mismatch problem due to the difficulty in identifying similar terms, for example, “like” and “enjoy”, which are synonymous to each other (Ayyadevara, 2018; Zhang et al., 2016).

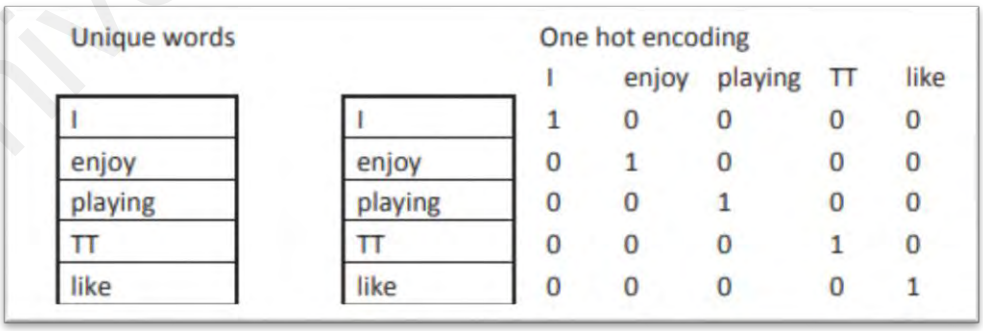


Figure 2.1: Example of One-hot Encoding (Ayyadevara, 2018)

To mitigate the aforementioned problem, researchers began to incorporate neural network in IR techniques to train better language model, known as word embedding. *Word2Vec* is a word embedding model that uses a simple three layers of neural network to learn words representations (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). The three layers are input layer, hidden layer and output layer. All the layers are fully connected and produce a single output. When training a language model with word embedding technique, the final output of the neural network is a vector representation for all the terms in the input document. There are two different word embedding algorithms in Word2Vec, which are Continuous Bag of Words (CBOW) and skip-gram, meant for different usage. CBOW predicts the current word for the given context while skip-gram predicts the surrounding words for the given current word (Mikolov, Chen, Corrado, & Dean, 2013). Taking the following sentence as an example, "The quick brown fox jumped over the dog.", Figure 2.2 and Figure 2.3 show the input and output of CBOW and skip-gram (Ayyadevara, 2018).

Input words	Output word
{*The, quick, fox, jumped*}	{*brown*}
{*quick, brown, jumped, over*}	{*fox*}
{*brown, fox, over, the*}	{*jumped*}
{*fox, jumped, the, dog*}	{*over*}

Figure 2.2: Example of CBOW (Ayyadevara, 2018)

Input words	Output word
{brown}	{The, quick, fox, jumped}
{fox}	{quick, brown, jumped, over}
{jumped}	{brown, fox, over, the}
{over}	{fox, jumped, the, dog}

Figure 2.3: Example of Skip-gram (Ayyadevara, 2018)

Word2Vec has some disadvantages: CBOW loses the order of words and skip-gram has little sense about the semantics of the words as it considers words order within a short context (Le & Mikolov, 2014). Therefore, a more recent study proposed *ParagraphVector*, also known as *Doc2Vec*, which learns the continuous distributed vector representations for pieces of texts (Le & Mikolov, 2014). The concept of Doc2Vec is based on *Word2Vec*'s but *Word2Vec* learns similarities between words while Doc2Vec learns similarities between sentences, paragraphs or documents (Le & Mikolov, 2014; Mikolov, Sutskever, et al., 2013).

Ye et al. (2016) is the first study that acknowledged the lexical gap between natural language queries and descriptions in API documentation, and they addressed it by applying word embedding technique to train a language model using API documentation. Later, Huang et al. (2018) and Rahman and Roy (2018) also addressed the lexical gap by applying word embedding technique to train a language model but they used SO data since SO posts is large in number and contain a mixture of API elements' terms and natural language terms.

2.7.2 Topic Modelling

Topic models are developed to automate extracting, indexing and searching information from large structured and unstructured text documents (Chen, Thomas, & Hassan, 2016). Besides that, topic models can be used to perform additional text analysis tasks such as, clustering, summarizing, and inferring links within the corpus (Chen et al., 2016). Topic modelling, also known as document clustering, is an unsupervised learning technique used to discover topics within a documents or paragraphs. When there is a lot of documents and limitation in summarizing all the documents, topic modelling technique proves helpful by extracting the topics within the documents automatically.

Latent Dirichlet Allocation (LDA) is one of the popular topic modelling technique that automatically discovers unobserved or hidden structures from text corpus using statistical properties, such as word frequency (Blei, Ng, & Jordan, 2003; Chen et al., 2016; Porteous et al., 2008). LDA is also known as a generative probabilistic model that calculates topic probability within a text corpus (Blei et al., 2003). Generative probabilistic model is a combination of generative process and probabilistic modelling. Generative process is "the imaginary random process by which the model assumes the documents arose" (Blei, 2012, p. 77). Probabilistic topic modelling is defined as "a suite of algorithms that aim to discover and annotate large archives of documents with thematic information" (Blei, 2012, p. 78). In the simplest form, the main concept of LDA topic modelling is the reasoning about how text corpus is represented by a mixture of topics and topics are characterized from the distribution of the words within the corpus (Blei et al., 2003). Figure 2.4 shows an example of applying LDA model to a text corpus, where topic is a collection of words, document is a mixture of corpus-wide topics and word is drawn from one of those topics (Blei, 2012).The advantage of LDA is fast, simple and does not require training data (Chen et al., 2016). LDA can be applied directly to raw,

unstructured text and the processing size is scalable to millions of documents (Porteous et al., 2008).

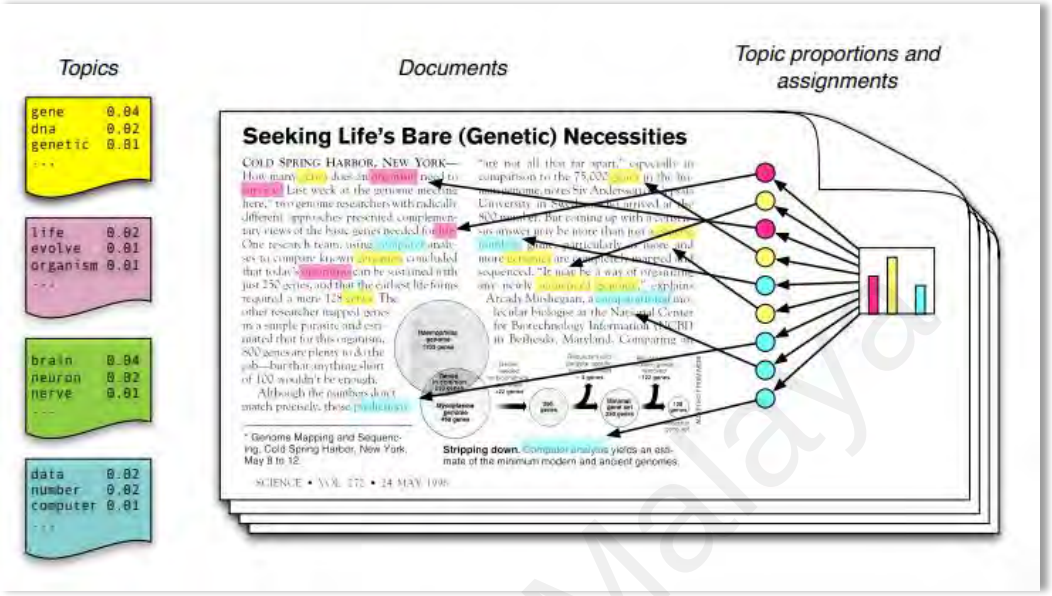


Figure 2.4: Example of LDA Model Application (Blei, 2012)

Some prior studies (Allamanis & Sutton, 2013; Joorabchi et al., 2016) have successfully applied LDA model in SO to discover common programming problems. In addition, Chen et al. (2016) surveyed over hundreds of studies (167 articles) in SE domain that used topic models and found that nearly two-third (66%) of the studies employed LDA for topic modelling. They also found that most of the studies used the basic topic models as black boxes without changing the underlying implementation or parameters (Chen et al., 2016).

2.8 Comparison of Existing Approaches in API Elements Search

This section compares the approaches adopted by three API recommender studies (*RACK, NLP2API, BIKER*) in terms of the techniques used. These studies fall under the API elements search category (refer to Section 2.5.1) and also recommend Java API elements for users' queries using SO posts. As mentioned in Section 2.5.1, these studies used the same term to refer to their approaches and the corresponding recommender tools that they built that incorporated the approaches.

All these studies including this research incorporated their approaches into the corresponding recommender tools that they built. Following that, the techniques were organized under the four major steps of designing RSSEs (Section 2.4.2). Table 2.2 summarizes the comparison of the techniques of the approaches.

Table 2.2: Summary of Techniques used in API Elements Search

Studies	Data Pre-processing	Capturing of context	Producing Recommendations	Presenting Recommendations
RACK (Rahman et al., 2016)	• Tokenization • Removal of Stop words • Stemming	• Token-API Mapping Database • Island parsing	• Keyword-API Co-occurrence (KAC) • Keyword-Keyword Coherence (KKC) • API Likelihood • API Coherence	• Top-K results
NLP2API (Rahman & Roy, 2018)	• Tokenization • Removal of stop words, punctuation marks and programming keywords	• Pseudo-Relevance Feedback (PRF) • Lucene • FastText • TF-IDF • PageRank	• Borda score calculator • Query-API proximity	• Top-K results
BIKER (Huang et al., 2018)	• Tokenization • Stemming	• Word2Vec • Inverse Document Frequency (IDF) • Two heuristic methods	• Similarity Score for Ranking Candidate APIs (SimSO and SimDoc)	• Top-K results • API methods descriptions • Similar questions

For data pre-processing, *RACK* (Rahman et al., 2016) performs tokenization, stop words removal, and stemming. Stemming analyses inflected word forms and extracts the root of each of the words by stripping the suffixes from the words (Kettunen, Kunttu, & Järvelin, 2005). For example, extract "generat" from "generates" by deleting "es" from the word "generates".

For capturing of context, *RACK* creates Token-API mapping pairs database that associates the natural language word tokens (from the respective SO question) and API classes tokens (from the SO question's accepted answer). This created Token-API mapping pairs. An example is the list of natural language tokens "generat, md5, hash" is associated to a API token "*MessageDigest*".

RACK adopts island parsing method for API classes extraction. Island parsing is a method that specifies rules to extract items of interest (such as code elements) and ignore uninteresting items (such as free form text) (Rigby & Robillard, 2013). A prior study has used island parser with Java Language Specification (Gosling et al., 2014) to identify code terms from text using regular expressions (Rigby & Robillard, 2013).

For producing the recommendations, *RACK* employs two heuristic metric calculation, which are Keyword-API Co-occurrence (KAC) and Keyword-Keyword Coherence (KKC) (Rahman et al., 2016). KAC helps to capture relationships between keywords and APIs such as co-occurrences or associations. KKC identifies coherent keyword pairs which are then used for obtaining candidate API classes that are functionally relevant to those pairs. *RACK* measures API Likelihood and API Coherence to produce Top-K results. API Likelihood estimates the probability of co-occurrence of a candidate API with an associated keyword. API Coherence estimates the relevance of a candidate API to multiple keywords from the query simultaneously. For presenting the recommendations, *RACK* presents a list of Top-K results.

For data pre-processing, *NLP2API* (Rahman & Roy, 2018) applies removal of stop words, punctuation marks and programming keywords, tokenization but not stemming. For capturing of context, *NLP2API* uses Pseudo-Relevance Feedback (PRF), *Lucene*, and *FastText (Rahman & Roy, 2018)*. PRF is employed to extract software-specific words that are relevant to a given query, and to use these words for query

reformulation (Nie, Jiang, Ren, Sun, & Li, 2016). *Lucene* is an open source API for building applications with search-related tasks such as indexing and querying (Białecki, Muir, Ingersoll, & Imagination, 2012). *FastText* is a word embedding algorithm that focuses on sub-word representations and does not require any pre-processing (Bojanowski, Grave, Joulin, & Mikolov, 2017). In addition, *NLP2API* uses two term weighting methods, Term Frequency - Inverse Document Frequency (TF-IDF)(Singhal, 2001) and PageRank (Brin & Page, 1998) to extract Java API classes from SO answers. TF-IDF is a term weighting algorithm formulated based on term frequency and document frequency. Term Frequency (TF) implied that words that repeat multiple times in a document are considered salient; whereas, document frequency implied that words that appear in many documents are considered common and not indicative of document content, this weighting method is called inverse document frequency (IDF) (Singhal, 2001). For producing the recommendations, *NLP2API* uses *Borda* score calculation and Query-API proximity. *Borda* count is a popular election method where the voters sort their political candidates in order of preference (Black, Hashimzade, & Myles, 2009). Query-API proximity analyses the global contexts of keywords within query, and measures the semantic proximity between the query and the candidate API classes. (Rahman & Roy, 2018). For presenting the recommendation, *NLP2API* displays the Top-K API classes.

BIKER (Huang et al., 2018) recommends two types of Java API elements, namely, classes and methods (Huang et al., 2018). For data pre-processing, *BIKER* performs tokenization and stemming on data retrieved from SO. *BIKER* performs two major steps in capturing of context, which are, retrieval of similar questions and detection of Java API elements. *BIKER* uses Inverse Document Frequency (IDF) and Word2Vec to retrieve similar questions. In addition, *BIKER* uses two heuristic methods to detect Java API elements. The first is using regular expressions to check whether every hyperlink in each

answer links to the official Java API documentation site. Secondly, *BIKER* checks the plain text contained in each answer against a dictionary that stores the names of all Java API elements from the official documentation site to identify whether there is any match of API elements. For producing the recommendations, BIKER calculates the similarity score for ranking candidate Java API elements by using a combination of two scores, namely SimSO and SimDoc (Huang et al., 2018). SimSO measures the similarity between the query and the question title of a similar question. SimDoc measures the similarity between the query and the Java API element's description in the official Java API documentation. For presenting the recommendations, BIKER displays a list of Top-K results which include Java API methods (and corresponding classes), description of the API methods and similar questions that matched the query.

2.9 Limitation of Existing Approaches in API Elements Search

To sum up, the first research gap found in existing studies on API Elements Search is the language model used by the corresponding API recommenders were often trained at the "word" level rather than the "sentence" level. *NLP2API* applied *FastText* and *BIKER* applied *Word2Vec*, which are word embedding algorithms targeting the "word" level. *RACK* study did not apply language model in its approach. As mentioned in Section 2.7.1 earlier, the limitations of word embedding algorithms that learn similarities at "word" level are: loses the order of words or has little sense about the semantics of the words due to considering words order within a short context. To cope with these limitations, word embedding algorithm that learns similarities at "sentences", "paragraphs" or "documents" has been proposed. This study employed word embedding algorithm due to the reason above.

The second research gap is these API recommenders often employed multiple techniques in API recommendation. As can be seen in Table 2.2, *RACK* and *NLP2API* employed ten different techniques and *BIKER* employed nine in their approaches. The number of different techniques employed increases the complexity of the respective framework. Employing fewer techniques will help to simplify the framework.

Lastly, there is a lack of work that provides API recommenders as plug-ins of an IDE, which could promote better usability and user friendliness of the recommenders. Only *RACK* did that but not *NLP2API* and *BIKER*.

Table 2.3: Summary of Limitation of Existing Approaches in API Elements Search

API Element Search Study/Tool	Language Model	Total Techniques Employed	IDE Plug-in
RACK (Rahman et al., 2016)	Not Applicable	10	Yes
NLP2API (Rahman & Roy, 2018)	Word-level (FastText)	10	No
BIKER (Huang et al., 2018)	Word-level (Word2vec)	9	No

2.10 Chapter Summary

As programmers are getting more involved in web collaboration communities, these communities have contributed a large amount of user-generated data with useful information and produced crowd documentation for different APIs. Many studies aimed to obtain useful insights through crowd documentation but there is limited work on common programming problems particularly on using core Java API. Besides that, recommendation systems or recommenders employing data mining techniques have been used in the SE domain. However, there are limitations found in existing API

recommenders such as language model used by the corresponding API recommenders were often trained at the "word" level rather than the "sentence" level, employed multiple techniques, and lack of work that provides API recommenders as plug-ins of an IDE. Thus, this research aimed to bridge these gaps by conducting two studies and the key steps involved are explained in details in the following chapter.

CHAPTER 3: RESEARCH METHODOLOGY

This chapter details the research methodology employed in this research. It describes the key steps conducted in this research to achieve the research objectives, and how the key steps were conducted in terms of techniques, tools, technologies used, where applicable. Figure 3.1 shows the nine (9) key steps (together with the associated research objectives where applicable) of the research methodology. To simplify the writing, the "identification of common Java programming problems" is called Study 1 and those related to the proposed approach (its development, benchmarking, plug-in, and user evaluation study) is called Study 2.

The research methodology employed in Study 1 is quantitative research. Study 1 involved research activity such as using counter to measure occurrence for duplicate question. On the other hand, research methodology employed in Study 2 is mixed method which consists of both and qualitative research. Quantitative research involved research activity such as calculation for performance benchmark, while qualitative research involved research activity such as gathering feedback from users and draw conclusion based on the result of user evaluation study.

Quantitative research was employed in Study 1, where data mined from Stack Overflow was analysed quantitatively to find the number of duplicate Java questions, top duplicate Java questions and their corresponding top Java API classes, based on the askers' level of expertise.

Study 2 employed both quantitative and qualitative research. Quantitative research was used in the benchmarking of the proposed approach where the metrics were calculated based on their formulas. Besides that, quantitative research was also used in user evaluation study where the correctness of Java API classes found by participants are

calculated. Qualitative research was used in user evaluation study where questionnaire was used to asks the participants' opinions on the features of APIRecJ.

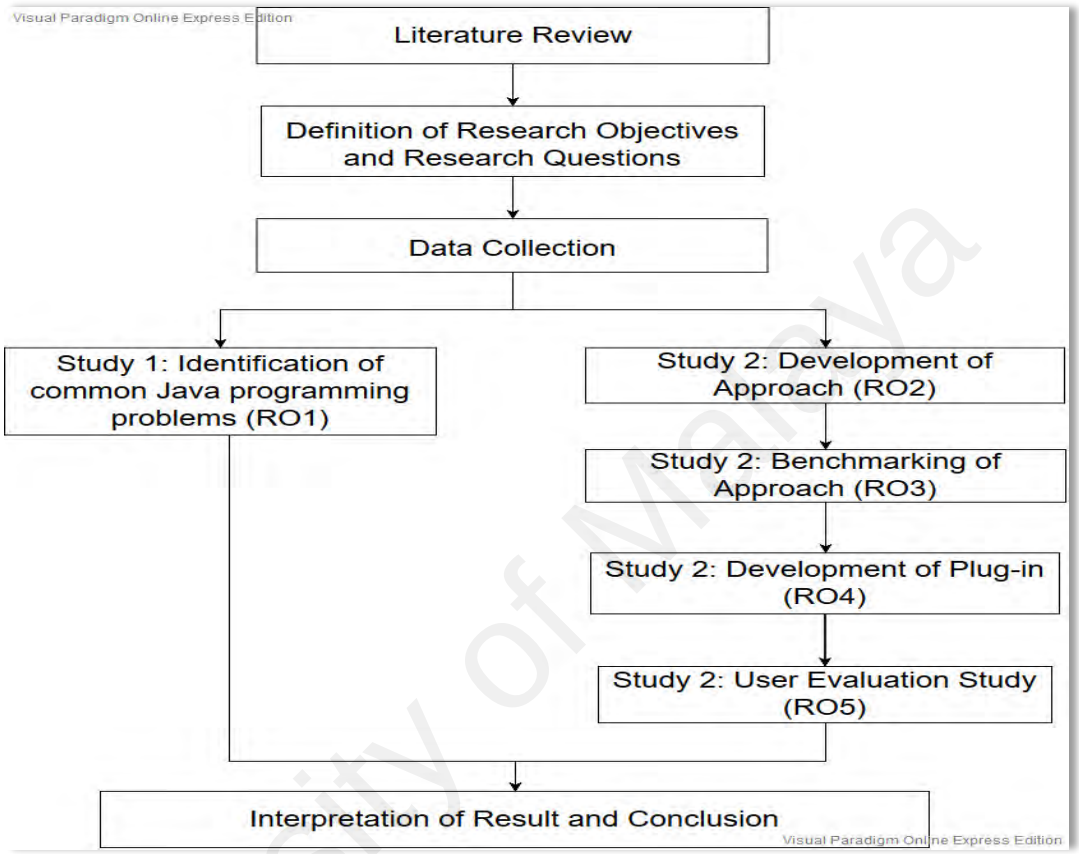


Figure 3.1: Research Methodology

3.1 Literature Review

The first key step is the literature review conducted on APIs documentation usability, limitations of the official or conventional APIs documentation, crowd documentation and its benefits, existing work on finding common Java Programming problems, recommendation systems in Software Engineering, existing work on API recommenders. Besides that, the review also included data mining in SO and popular data mining techniques.

3.2 Definition of Research Objectives and Research Questions

Based on the literature review, the key limitations found in existing studies are:

i) Limited work on common problems in Java programming, particularly in terms of core Java API usage and the common Java programming problems faced by Java programmers of different levels of expertise.

ii) In terms of API recommenders,

   a. A lack of language models at sentence level instead of word level.

   b. Multiple techniques employed in the design of existing API recommenders.

iii) Lack of work that provides API recommenders as plug-ins of an IDE, which could promote better usability and user friendliness of the recommenders.

Based on the problems highlighted above, the research objectives and research questions were defined for this research, focusing on identifying common Java programming problems from SO, and developing an approach for Java API classes recommendation that uses word embedding and topic modelling techniques on discussion posts extracted from SO.

3.3 Data Collection

This research made use of data extracted from SO. This data consists of data generated by or from SO's users. This data such as questions and answers in SO discussion posts, information about the registered users, and so on, is stored and maintained by SO in multiple databases.

There are two methods used to retrieve data from SO in existing studies, which are, using the data dump (Ahasanuzzaman et al., 2016) and a query tool (Rahman et al.,

2016). The first method is getting the data dump archive files released by Stack Exchange (2019e) and replicating the databases in a local machine. The second method is using the "Stack Exchange Data Explorer (SEDE)" (2019b), a web-based query tool provided by Stack Exchange. SEDE displays a list of data dictionary that describes the schema of database and query input for retrieving data by using Structured Query Language (SQL) statements (Chamberlin & Boyce, 1974). The advantages of using SEDE as compared to using data dump files are SEDE retrieves the latest and updated data, and provides instant access to SO data using SQL queries. Contrarily, data dump files are released periodically and require additional setup and installation to load the database and access the data. However, SEDE can only retrieve at most 50,000 records of data.

This research employed SEDE in extracting data from SO. This was done by retrieving the data in batches using the unique identifier of each discussion post and merging all the batch files into a single file. The extracted data comprises of SO data dating from the inception of SO (15 September 2008) to the data retrieval date (6 July 2019).

It is important to note that the data collection for Study 1 and Study 2 are different. The data collection for Study 1 retrieved a set of duplicate posts related to Java, while the data collection for Study 2 retrieved a set of posts related to Java. The codes to extract the data for Study 1 and Study 2 are given in Appendix B and Appendix C, respectively.

3.4 Development of Approach

The next key step is the development of an approach that aimed to assist programmers by recommending a list of relevant Java API classes for their natural language queries. This is to achieve RO2. There are two major phases in the proposed

approach which are preparation phase and recommendation phase. The techniques employed in the proposed approach are word embedding, topic modelling and heuristic rules. The proposed approach is incorporated into a Java API class recommender created as a backend server which exposes the services for performing Java API classes recommendation. The technologies used in the development of the approach are listed in Table 3.1.

Table 3.1: Technologies Used in the Development of the Approach

Programming Language	Technology Name and Description
Python 3.6	• Gensim - Open source NLP and ML libraries • BeautifulSoup - Open source library for XML text processing • Pandas - Open source library for CSV file processing • Flask - Open source library for server • PyInstaller - Open source library for packaging python script into executable window platform file

3.5 Identification of common Java programming problems

To achieve RO1, the data collected for Study 1 was analyzed to identify the common Java programming problems. There are three questions formulated at the beginning of the study, which are related to the distribution of duplicate Java questions in SO based on the askers' level of expertise, top duplicate Java questions in SO based on askers' level of expertise and top Java API classes required by the top duplicate Java questions in SO based on the askers' level of expertise. A duplicate question refers to a question that has been asked and answered before by the SO community (Stack Overflow, 2019j).

As a result, Study 1 answered the first question with a group of SO users who frequently ask duplicate Java questions in SO. Besides that, Study 1 answered the second question with the Top-10 duplicate Java questions in SO and revealed the significant

critical area in using Java API. In addition, the most common problem faced by Java programmers are derived from these top-10 duplicate Java questions in SO based on askers' level of expertise. Finally, Study 1 answered the third question with the Top-30 Java API classes required by the Top-10 duplicate Java questions in SO and discovered the Java API classes that Java programmers struggle with.

3.6 Benchmarking of Approach

In this key step, the proposed approach is evaluated in term of its performance by using 4 established performance metrics used by existing studies. The measured metrics were compared to existing baseline studies in Java API class recommendation. This is to achieve RO3.

The four metrics are: (a) Top-K accuracy, (b) Mean Recall @ K (MR@K), (c) Mean Reciprocal Rank @ K (MRR@K), (d) Mean Average Precision @ K (MAP@K). These metrics measure the information retrieval (IR) performance and the recommendation performance of the proposed approach. For the former, these metrics assess the performance of the proposed approach in retrieving set of relevant API classes. For the latter, these metrics assess the performance of the proposed approach in returning relevant API classes at the top positions and less relevant API classes at the bottom positions of the ranked results list.

3.7 Development of Plug-in

To make it easier for the programmers to use the developed Java API class recommender, a plug-in for Eclipse IDE that functions as the front-end or client for programmers to access the recommender's functionalities, was developed. Table 3.2

shows the technologies used in implementing the plug-in. The plug-in was developed by creating an Eclipse plug-in project. Its user interface was designed using Java Swing components that receive user query and display query result. The plug-in functions as the requestor, which sends user query to Java API class recommender and returns a set of Java API classes and similar questions to user.

Table 3.2: Technologies Used in the Development of the Plug-in

Programming Language	**Technology Name and Description**
Java 1.8	• Eclipse Oxygen – IDE for java development • Eclipse Plug-in Developer SDK – Libraries for developing Eclipse plug-in

3.8 User Evaluation Study

A user evaluation study was conducted to evaluate the usefulness of the plug-in (*APIRecJ*) including the features implemented in the developed Java API class recommender. The user evaluation study involved recruiting participants to use Google search engine and to use the plug-in to search for and state Java API classes that are relevant to three pre-defined programming questions/tasks. The participants then completed a questionnaire survey comprising multiple-choice questions, Likert scale questions and open-ended questions. The questionnaire asks about participants' educational background, level of Java programming skill, other programming languages known and the level of skill, and Software Development Kits (SDKs) familiar with, opinions on the features of *APIRecJ*, whether they prefer Google search engine or *APIRecJ* and their reasons for their preference, and usefulness of having API class recommender such as *APIRecJ* and the reasons.

3.9 Interpretation of Result and Conclusion

Study 1 provided the distribution of duplicate Java questions based on askers' level of expertise, top-10 master Java questions based on askers' level of expertise and top-30 Java API classes required by the top duplicate Java questions based on askers' level of expertise. These results were discussed.

For Study 2, the benchmarking results against existing studies were discussed in terms of aspects of improvement; the user evaluation study's results in terms of the usefulness of the plug-in were also discussed.

3.10 Chapter Summary

The research methodology provides a comprehensive detail on how this research was carried out. This covers the key steps conducted in this research to achieve the research objectives, and how the key steps were conducted in terms of techniques, tools, technologies used, where applicable.

CHAPTER 4: COMMON JAVA PROGRAMMING PROBLEMS

This chapter presents Study 1 conducted in this research, which is, the identification of common Java programming problems from SO. It explains the relevant terms and concepts, questions for Study 1, database structures in SO, and how the extraction of duplicate questions, and code snippets and Java API classes were performed. This chapter also includes the results found for each questions and comparison with related work.

4.1 Questions for Study 1

This Study 1 aimed to leverage SO's crowd documentation of Java APIs to investigate what are the most common programming questions asked by or frequently asked questions of the Java community based on their level of expertise. This was done by mining and analysing duplicate Java questions/posts in SO. The level of expertise was determined based on the askers' reputation scores in SO. A duplicate question refers to a question that has been asked and answered before by the SO community (2019j). Duplicate questions can also be regarded as "questions that are asked to solve the same problem" (Ahasanuzzaman et al., 2016).

Since duplicate questions are the same questions that different programmers repeatedly asked, they can be used as surrogates to common questions asked in SO. Doing that can help in identifying common Java programming problems/topics that Java programmers struggle with.

The specific questions (Qs) for Study 1 are:

Q1: What is the distribution of duplicate Java questions in SO based on the askers' level of expertise?

Q2: What are the top duplicate Java questions in SO based on askers' level of expertise?

Q3: What are the top Java API classes required by the top duplicate Java questions in SO based on the askers' level of expertise?

4.2 Database Structure in SO

SO works as a discussion forum where registered users in SO can ask programming questions by creating new discussion threads to be answered by others. Hence, discussion posts in SO typically consists of a question and multiple replies. SO has given guidelines on how to construct a question **Error! Bookmark not defined.**. A q uestion's title should record the description that summarizes the specific problem. The question body should start with a more detailed description of the problem faced; has just enough code for reproducing the problem, has link to live example of the problem if possible; code, data or error log in text and not image forms. The question should include all relevant tags, and proof-read before posting.

Figure 4.1 shows a partial data model for the SO database. The script used to visualize the data model is provided by SO community (2019; 2019a). The tables used for extracting the duplicate Java questions are: *Posts, PostTypes, PostLinks, Tags, PostTags* and *Users* tables. The *Posts* table stores all the discussion posts in SO with unique primary key *Ids*. The *PostTypeId* attribute with a value of "1" indicates that the post is a question and a value of "2" means that the post is an answer. If the post is a

question post, the *Title* attribute records the question's title and the *Body* attribute records the description of the question. If the post is an answer post, the *Title* attribute will be empty and the *Body* attribute will store the answer description. Each question in SO can be tagged with a set of labels and all the tags are stored in the *Tags* table with unique *Ids*. The tags for a question are recorded in *Tags* attribute of the Posts table and the tag's name are stored in *TagName* attribute of the *Tags* table. For example, discussion posts with "java" tag can be regarded as Java-related posts, which are related to the Java programming language.

The *Users* table stores all registered SO users with unique *Ids*. The *Reputation* attribute records the "reputation" the user acquired since he or she joined SO. "Reputation is a rough measurement of how much the community trusts the user" (2019h). It is earned by convincing the community that you know what you are talking about. The higher the reputation of a user, the more privileges the user gets and will have access to more features on the site. A user's reputation score will increase or decrease depending how the community perceives the quality of the user's posts. For example, if the user's question is voted up, his or her reputation will increase by 5; if the user's answer is marked "accepted", the user's reputation will increase by 15; and if the user's question or answer is voted down, his or her reputation will decrease by 2.

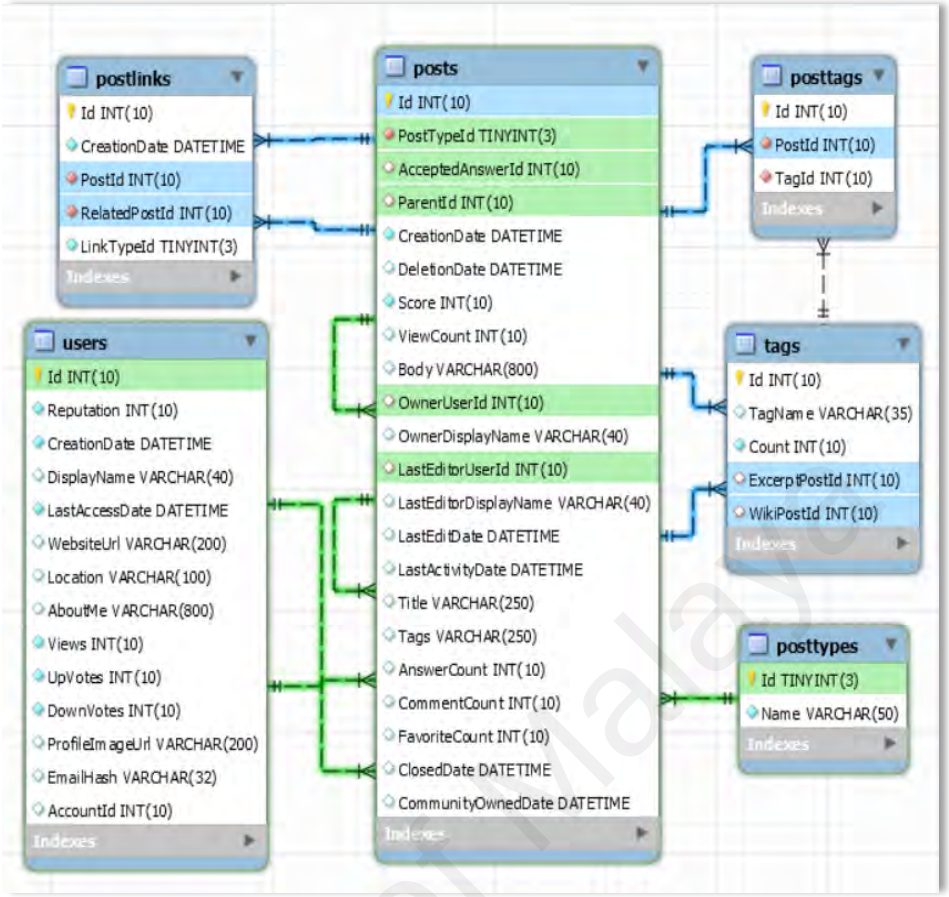


Figure 4.1: Partial Data Model for SO Database

4.3 Extraction of Duplicate Questions

SO provides a set of guidance to its users on how to ask questions in SO, for example, what types of questions should be avoided, how to ask a good question, and so on (2019d). Despite SO recommendation that the askers search previous posts before asking a new question, duplicate questions sharply increase after September 2012 and analysis of tags showed that the “java” tag has the highest number of duplicate questions (Ahasanuzzaman et al., 2016). The reasons that duplicates happened are (Ahasanuzzaman et al., 2016): askers did not search SO first before asking a question, titles of master questions do not match askers intended questions, domain difference despite task

similarity, descriptive and difficult to understand, too concise to properly understand, lack of knowledge about the problem and terminology/buzzwords.

Duplicate questions are seen as problematic due to the following reasons: may cause the asker unnecessarily wait in getting answer when the question has already been asked and answered previously (Ahasanuzzaman et al., 2016) (Zhang, Lo, Xia, & Sun, 2015), and difficulty in finding correct answers in one place (Silva, Paixão, & de Almeida Maia, 2018). SO community moderators' do not remove or delete duplicate questions since similar questions can be described differently or exist in different contexts. However, this research regards duplicate questions as an opportunity to identify the common programming questions asked by programmers which could give us insights on the common programming problems/topics that programmers struggle with.

A duplicate question is a question that has been asked and answered before in SO. The site moderators (elected by the SO community through popular vote) and users with high reputation (more than 3000) are able to close later questions as duplicate questions (i.e. the non-master question) and provide the reference links to the respective first-time asked question (i.e. the master question) (Ahasanuzzaman et al., 2016).

The relationships between a master question and its duplicate questions are recorded in the *PostLinks* table. Each master-duplicate relationship is captured as a record in the *PostLinks* table with a unique Id. The *PostId* attribute is a foreign key which is linked to *Id* attribute in *Posts* table that records the identifier of the master question. The identifier of the duplicate question is stored in *RelatedPostId* attribute. The attribute *LinkTypeId* describes the type of relationship between the question identified by *PostId* and the question identified by *RelatedPostId*. For instance, a master-duplicate relationship is signified by a value of "3" in the *LinkTypeId* attribute.

The collection of duplicate Java questions was retrieved using SEDE with relevant SQL statements. For Q1, all the master questions were retrieved and their duplicate questions were counted based on the groups of users defined using the reputation score (Section 2.2.2). For Q2, the top-10 master questions based on the groups of users were extracted. The scripts used for extraction are provided in Appendix B.

Figure 4.2 illustrates an example of a master question in SO whereas Figure 4.3 illustrates an example of a duplicate question (non-master question) in SO.

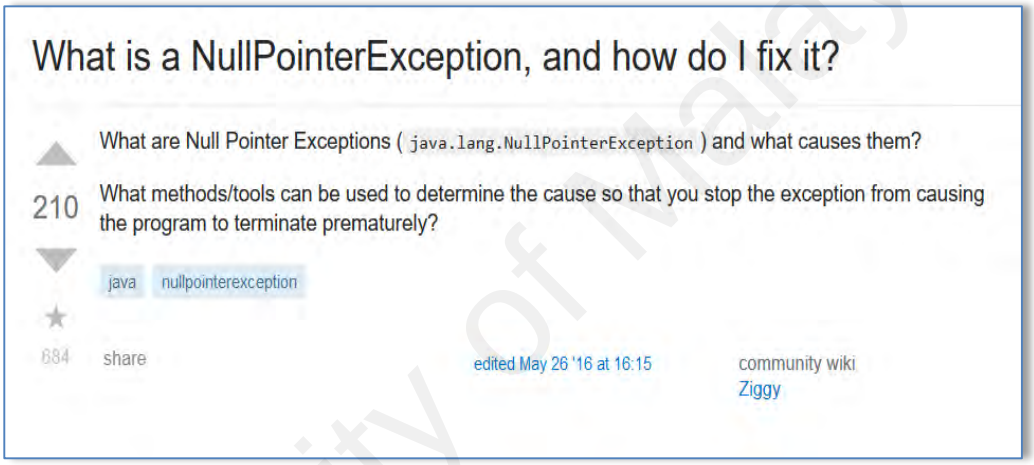


Figure 4.2: Master Question (2019g)

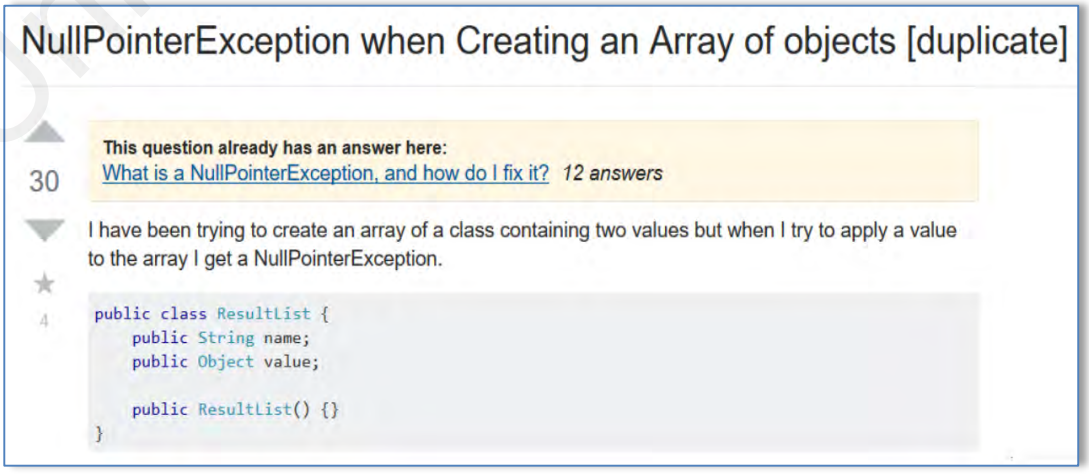


Figure 4.3: Duplicate Question (Non-Master Question) (2019e)

4.4 Extraction of Code Snippets and API Classes

For Q3, after extracting the duplicate Java questions and saved them in CSV files, the following filtering steps were performed:

Step 1: Extract all master questions that have accepted answers.

Step 2: Extract code snippets from the accepted answers.

Step 3: Extract Java API classes from the code snippets by using a set of heuristic rules implemented in Python scripts.

Among all the answers to the master question, only the accepted answer of the master question is considered because an existing study on one domain showed that 65% of accepted answers contain source code (Subramanian & Holmes, 2013). Since code snippets frequently refer to specific API (Subramanian & Holmes, 2013), these filtering steps focused on Java code snippets and ignored other information in the accepted answer (such as plain text description, hyperlink, logs, comments and others). The code snippets were obtained by extracting the content enclosed by the "<code>" and "</code>" tags. As in (Subramanian & Holmes, 2013), the extraction of API classes considered only code snippets that had at least 3 lines of code because anything less is lack of surrounding context needed to understand an API. Table 4.1 shows the heuristic rules used in Step 3, together with the justifications and some examples.

Table 4.1: Heuristic Rules for Extracting Java API Classes

No.	Heuristic Rule	Justification	Example	
			Input	**Java API class extracted by the heuristic rule**
1	Extract class names from 'import' statements	One way to use/access a Java API class/interface in a Java program is by using an "import" statement to specify the packages where the class/interface resides. Class names extracted from the 'import' statements are classes that might be relevant to the respective question.	import java.io. ***FileInputStream;***	FileInputStream
2	Extract reference types of reference variables	The reference type of a reference variable is a class/interface that is relevant to the respective question.	***InputStream*** is = new FileInputStream();	InputStream
3	Extract the name of the constructor method located right after a 'new' keyword	The name of the constructor method used indicates the reference type of the object instantiated. This reference type is a class that is relevant to the respective question.	InputStream is = new ***FileInputStream***();	FileInputStream

4	Validate API classes extracted by the first three heuristic rules by checking with API classes extracted from the Java API documentation for Java Standard Edition 8 and Java Enterprise Edition 8.	Code snippets might contain Java API classes spelled wrongly or in the wrong letter cases. Therefore, candidate API classes have to be checked against the valid API class names extracted from the API documentation. This heuristic rule helps to eliminate invalid API classes.	inputStream INPUTStream InputStem	InputStream InputStream -

4.5 Results and Discussion

This section presents the results and discussion of Study 1 with regard to the 3 questions the study aimed to answer. This section also includes the comparison with related work.

4.5.1 Data Extracted

From the data, 27589 master questions that have duplicate questions are found and extracted. A user has the option to "accept" an answer to his or her question, thereby making the answer the "accepted answer" (2019f) . Only 21623 (78%) of the extracted master questions have accepted answers. This could probably due to accepting an answer is not mandatory and not all user came back to the site to accept an answer. Only 10763 (39%) master questions have code snippets in their accepted answers. This shows that 60% of the accepted answers do not use code examples in providing solutions for the questions. There are 6631 (24%) master questions found having Java API classes in their accepted answers' code snippets. This means that only about a quarter of the master

questions contain solutions that make use of Java API classes. Figure 4.4 illustrates the number of remaining master questions after each filtering step.

Figure 4.4: Overlapping Relationships within Data Collection

4.5.2 Q1: What is the distribution of duplicate Java questions in SO based on the askers' level of expertise?

Users' reputation scores in SO are used as an approximation to their level of expertise or experience. As in (Ahasanuzzaman et al., 2016); Ahasanuzzaman et al. (2018), those with lower reputation scores are regarded as having lesser expertise or experience, and those with higher reputation scores as more expert or experienced. Three groups of SO users are defined based on their reputation scores: novice group with reputation score less than 100; experienced group with reputation score in between 100 and 10000, inclusive; and expert group with reputation score above 10000.

To answer Q1, the number of duplicate Java questions asked by each of the three groups are identified. Table 4.2 shows the results. The total duplicate Java questions is

62,553, with two-third contributed by the novice group, 31% contributed by experienced group, and 3% contributed by the expert group.

Table 4.2: Distribution of Duplicate Java Questions Based on Askers' Level of Expertise

Level of Expertise	Number of duplicate Java questions
Novice Group (reputation < 100)	41452 (66%)
Experienced Group (100 < reputation < 10000)	19249 (31%)
Expert Group (reputation > 10000)	1852 (3%)

4.5.3 Q2: What are the top duplicate Java questions in SO based on askers' level of expertise?

To investigate Q2, the top-10 master Java questions for the three groups of users (Table 4.3) are identified. The middle column of Table 4.3 shows the titles of the master questions and the count column shows the number of duplicate questions for the corresponding master question.

As shown in Table 4.3, the top two duplicate Java questions of the novice and experienced groups are related to how to solve *NullPointerException* and how to compare *String*, with the frequency of novice asking these questions significantly higher than experienced group. Only a few from the expert group posted similar questions, with duplication counts below 10.

The majority of the top duplicate Java questions of the novice and the experienced groups, and one-third of the questions of the expert group are related to understanding and/or fixing exceptions and errors. For instances, compilation error, *NullPointerException*, and *ArrayIndexOutOfBoundsException*. On the contrary, the number of duplicate questions related to reasoning of the Java programming concepts are

the highest in the expert group, and rare in the experienced and novice groups. For example, "….Why are Java generics not implicitly polymorphic?", "Why don't Java's +=, -=, *=, /= compound assignment operators require casting", and so on.

The Top-10 duplicate questions can be regarded as the most common Java programming problems/topics that Java programmers struggle with. Our results show that the most common problem Java programmers face is understanding and/or fixing errors. However, this is not the case for the expert programmers, as they question more about the reason of some Java programming concepts. It is important to note that the expert rarely asked duplicate Java questions in SO. This study in fact identified the top-30 duplicate Java questions in SO based on the askers' level of expertise and this is provided in Appendix A.

Table 4.3: Top-10 Master Java Questions Based on Askers' Level of Expertise

Level of Expertise	Top-10 master Java Questions	Duplication Count
Novice Group	1. What is a NullPointerException, and how do I fix it?	5009
	2. How do I compare strings in Java?	1631
	3. What causes a java.lang.ArrayIndexOutOfBoundsException and how do I prevent it?	554
	4. Scanner is skipping nextLine() after using next() or nextFoo()?	420
	5. How do I fix android.os.NetworkOnMainThreadException?	345
	6. What does a "Cannot find symbol" compilation error mean?	297
	7. Unfortunately MyApp has stopped. How can I solve this?	223
	8. How do I print my Java object without getting "SomeType@2f92e0f4"?	210
	9. What's the simplest way to print a Java array?	203
	10. Why is my Spring @Autowired field null?	133
Experienced Group	1. What is a NullPointerException, and how do I fix it?	606
	2. How do I compare strings in Java?	274
	3. How do I fix android.os.NetworkOnMainThreadException?	116

	4. Is List<Dog> a subclass of List<Animal>? Why are Java generics not implicitly polymorphic?	96
	5. Why is my Spring @Autowired field null?	92
	6. How do I write a correct micro-benchmark in Java?	52
	7. How to fix java.lang.UnsupportedClassVersionError: Unsupported major.minor version	49
	8. Why does Spring MVC respond with a 404 and report "No mapping found for HTTP request with URI [...] in DispatcherServlet"?	49
	9. Iterating through a Collection, avoiding ConcurrentModificationException when removing objects in a loop	44
	10. Java string to date conversion	41
Expert Group	1. Is List<Dog> a subclass of List<Animal>? Why are Java generics not implicitly polymorphic?	10
	2. Iterating through a Collection, avoiding ConcurrentModificationException when removing objects in a loop	8
	3. How do I compare strings in Java?	8
	4. What is a NullPointerException, and how do I fix it?	7
	5. Why don't Java's +=, -=, *=, /= compound assignment operators require casting?	6
	6. How do I fix android.os.NetworkOnMainThreadException?	5
	7. When do you use Java's @Override annotation and why?	5
	8. In Java, what is the best way to determine the size of an object?	5
	9. How to create a generic array in Java?	5
	10. Why use getters and setters/accessors?	5

4.5.4 Q3: What are the top Java API classes required by the top duplicate Java questions in SO based on the askers' level of expertise?

Table 4.4 shows the top-30 Java API classes (and the packages they belong) required by the top duplicate Java questions in SO based on the askers' level of expertise. Note that only 27 Java API classes were found for the expert group.

The results show that some of the most frequent Java API classes required by the top duplicate Java questions of all expertise groups are from *java.lang, java.util, javax.swing, java.text, java.io* and *java.net* packages. Nevertheless, for these six packages, there are some overlaps and differences in terms of the specific Java API classes found for each expertise group, as shown in Table 4.4.

Other than that, Java API classes from *java.awt* package were also found for the novice group but not for the other two groups. This package mainly contains classes use for developing the graphical user interface of Java programs (2019). This shows that novice programmers require more help in using these API classes in developing user interface.

Another difference is, the *DateTimeFormatter* and *ResultSet* classes were found for the experienced and expert groups but not for the novice group. This could be due to these two classes are used in more advanced topics and novices have not encounter them. *DateTimeFormatter* is related to formatter for printing and parsing date-time objects. *ResultSet* is related to SQL query of database.

Table 4.4: Top-30 Java API Classes Required by the Top Duplicate Java Questions Based on Askers' Level of Expertise

Level of Expertise	Java API Class
Novice Group	***java.lang**: String, Object, StringBuilder*
	***java.awt**: EventQueue, Dimension, ActionEvent, BufferedImage, Graphics2D, Color, ActionListener, BorderLayout*
	***java.util**: ArrayList, Date, HashMap, Scanner, List, Calendar, Matcher, Pattern, Random*
	***javax.swing**: JFrame, JPanel, JButton, JLabel*
	***java.text**: SimpleDateFormat,*
	***java.io**: File, BufferedReader, IOException, InputStream,*
	***java.net**: URL*
Experienced Group	***java.lang**: String, Object, StringBuilder, Integer, Process, Thread*
	***java.util**: ArrayList, Date, HashMap, Scanner, List, Calendar, Matcher, Pattern, HashSet*
	***javax.swing**: JFrame, JPanel, JLabel, Document*
	***java.text**: SimpleDateFormat, DateFormat*
	***java.io**: File, BufferedReader, IOException, InputStream, FileInputStream, FileOutputStream*
	***java.net**: URL*
	***java.time**: DateTimeFormatter*
	***java.sql**: ResultSet*
Expert Group	***java.lang**: String, Object, StringBuilder, Integer, Process, Thread, Method*
	***java.util**: ArrayList, Date, HashMap, Scanner, List, Calendar, Pattern, HashSet*
	***javax.swing**: JPanel, JLabel*
	***java.text**: SimpleDateFormat, DateFormat*
	***java.io**: File, BufferedReader, IOException, InputStream*
	***java.net**: URL*
	***java.time**: DateTimeFormatter*
	***java.sql**: ResultSet*

4.5.5 Summary of Results

Study 1 investigated which group of SO users frequently ask duplicate Java questions in SO site. The results show that the novice group is the top contributor and the expert group contributes significantly lower to this. The top-10 duplicate Java questions in SO were identified and it was found that the most common problem Java programmers

face is understanding and/or fixing errors. However, this is not the case for the expert programmers, as they question more about the reason of some Java programming concepts. Additionally, the top-30 Java API classes required by the top-10 duplicate Java questions in SO were identified and the main findings are: some of the most frequent Java API classes required by the top duplicate Java questions of all expertise groups are from *java.lang, java.util, javax.swing, java.text*, java.io and *java.net* packages; novice programmers ask more duplicate questions related to classes in *java.awt* package compared to experienced and expert programmers; experienced and expert programmers but not the novice asked duplicate questions related to classes used in more advanced topics such as querying of database.

In summary, Study 1 provides insights on common Java programming problems/topics and Java API classes that Java programmers struggle with. Java educators and learning resources can devote more attention to these areas to help learners in picking up the required knowledge and skills.

4.6 Comparison with Related Work

Prior study has shown that SO users who have the least experience (less than 100 reputation score) posted the highest number of duplicate questions regardless of the programming language (Ahasanuzzaman et al., 2016). Our study focused only on Java questions and our finding shows similar result where novices tend to ask more duplicate questions compared to more experienced users.

An existing study investigating topics discussed in SO questions using clustering technique found that one of the major categories of questions asked is concepts that been coded but do not work (Allamanis & Sutton, 2013). The finding of Study 1 is in line with

this study, and found that Java programmers frequently ask questions related to understanding and/or fixing errors.

A study on the characteristics of SO posts on API issues found that SO users with less than 100 reputation score and those with more than 10000 reputation score ask less API issue-related questions, as compared to those having reputation score between 100 - 10000 (Ahasanuzzaman et al., 2018). Study 1 differs in the sense that its scope focused on duplicate questions and the Java domain.

4.7 Chapter Summary

Study 1 was conducted to identify the common Java programming problems using duplicate questions retrieved from SO. Duplicate questions from SO can be regarded as common questions asked in SO and used to identify common Java programming problems/topics that Java programmers face. The common problems/topics serve as important areas that Java educators and learners could pay more attention to. The next chapter explains the proposed approach for recommending Java API classes performed in the Study 2 of this research.

CHAPTER 5: THE PROPOSED APPROACH

This chapter presents the approach focused on Java API classes recommendation using SO data. There are mainly five subsections in this chapter. First subsection provides the overview of proposed approach with techniques employed; second and third subsection elaborate on two major phases in the proposed approach, which are preparation phase and recommendation phase, respectively. In fourth subsection, the API recommenders in API element search category (in Section 2.5.1) are discussed, particularly on the techniques used in their works. The fifth subsection explains on the development of recommender, which details on how to develop plug-in that serve as an interface for the proposed approach.

5.1 Overall Design of the Proposed Approach

Figure 5.1 illustrates the overall design of the proposed approach. The proposed approach comprises of a preparation phase and a recommendation phase. The preparation phase involves four steps and the recommendation phase involves three steps, which are explained in detail in the following sections. In Figure 5.1, the steps are represented by rectangles.

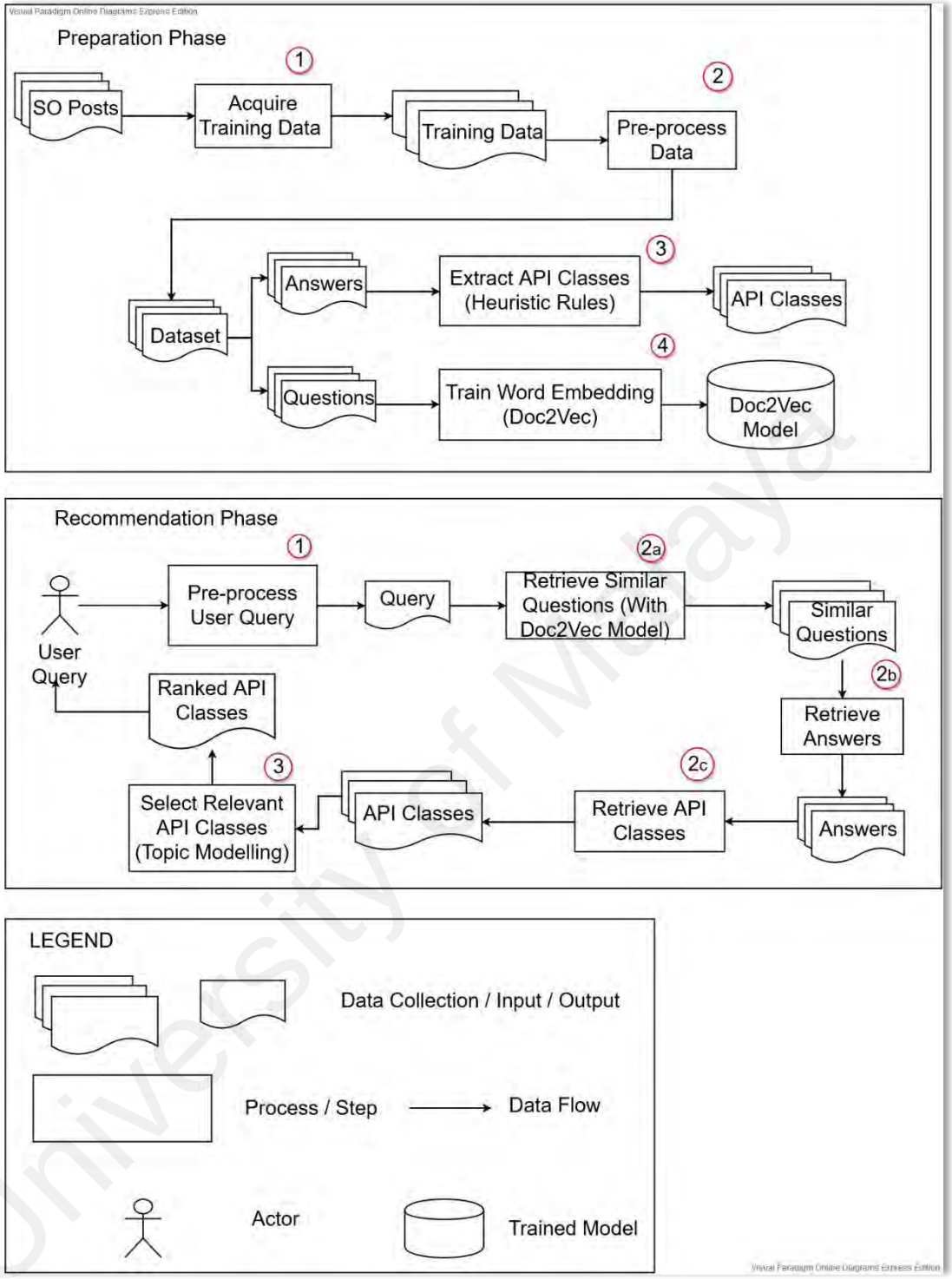


Figure 5.1: Overall Design of the Proposed Approach

5.2 Preparation Phase

There are four steps in the preparation phase:

1. Acquiring training data by extracting Java-related posts from SO;
2. Pre-processing extracted posts (training data) and split them into two datasets, namely, Questions dataset and Answers dataset;
3. Use a set of heuristic rules to extract Java API classes from the Answers dataset. The extracted Java API classes serve as candidate classes, which are classes that might be recommended to a user based on his or her query;
4. Use the Questions dataset as input to train and produce a *Doc2Vec* (Le & Mikolov, 2014) word embedding model.

There are three datasets produced in the preparation phase, which are training data, Answers dataset and Questions dataset. These datasets are stored in different files in Comma-Separated Value (CSV) format. The training data includes the questions' identifiers (QuestionId) and their title (QuestionTitle) and accepted answer (AcceptedAnswer). The Answers dataset includes the questions' identifier and their accepted answer and corresponding Java API classes extracted (APIClasses). The Questions dataset includes the questions' identifier and their title. Table 5.1 shows the columns contained in each of the dataset.

Table 5.1: Columns in Dataset

Dataset	**Column Name**
Training Data	QuestionId, QuestionTitle, AcceptedAnswer
Answers	QuestionId, AcceptedAnswer, APIClasses
Questions	QuestionId, QuestionTitle

5.2.1 Step 1: Acquiring Training Data

The training data was extracted from SO by using the same web-based query tool named SEDE (Section 3.3) used in Study 1 (Chapter 4). All the discussion posts tagged with a "java" tag and having an accepted answer were retrieved for this study. As a result, the training data comprises Java posts that have accepted answers.

It is important to note that this training data is much larger from the dataset used to find the common Java programming problems in Study 1. It contains 632,078 discussion posts comprising of all Java discussion posts with accepted answers, whereas the dataset in Study 1 only contains 27,589 discussion posts comprising only master and duplicate Java discussion posts. This larger amount of training data is suitable for building a word embedding model, as can be seen in the amount of data used by existing studies in building language models, namely, *RACK* that used 344,086, *BIKER* that used 1,347,908 and *NLP2API* that used 656,538 Java related discussion posts (Huang et al., 2018; Rahman & Roy, 2018; Rahman et al., 2016).

5.2.2 Step 2: Pre-processing Training Data

In this step, the training data was pre-processed to remove irrelevant information in order to train better word embedding model and to obtain better performance in recommendation. There are various pre-processing techniques, such as, data cleaning, data reduction, data transformation and data integration (Han et al., 2011). The purpose of data cleaning is to remove noise from data while data reduction is about reducing data size by aggregating and eliminating redundant features. Data transformation is related to data normalization, which is, transforming the data to fall within a smaller or common range such as [-1, 1] or [0.0, 1.0]. Data integration is related to merging data from multiple sources into a coherent data store. Only data cleaning was used in this research to pre-

process the training data, since the training data does not require elimination of redundant features, normalization or merging from multiple sources.

Data cleaning technique was applied to all the questions' titles in the training data to remove the noise in the titles. The data cleaning involves three steps: tokenization, removal of noise, and lemmatization. First, tokenization was carried out where a question's title was tokenized. For example, the tokenization of the title "How to convert Integer to String?" generates seven tokens, which are, "How", "to", "convert", "Integer", "to", "String", and "?".

Next, removal of noise was performed on the resulting tokens. The noise includes English stop words and punctuation marks. English stop words are words that have no significant meaning, such as 'the', 'this', 'a' and so on, and are of high occurrence. The noise can be regarded as random errors or variations in text (Han et al., 2011). After the noise removal, only four tokens remained for the given example: "How", "convert", "Integer", and "String". The tokenization and removal of noise were performed using the Natural Language Toolkit (NLTK) library, an open source statistical NLP library (Loper & Bird, 2002).

In addition, the remaining tokens was lemmatized with the WordNet model, a lexical database that describes semantic relationships for words in English language (Miller, 1995). Lemmatization is a well-known IR technique to retrieve the base form of a word, (Kettunen et al., 2005). For example, retrieve the base form "write" from "writing" and "wrote".

Lastly, the training data was split into a Questions dataset and an Answers dataset, stored tin two different csv files. As can be seen in Table 5.1, the Questions dataset contains the SO questions' identifiers and their titles. At this point of time, the Answers

dataset contains the SO questions' identifiers and their accepted answers but not the corresponding Java API classes. The scripts and algorithm used for extraction are provided in Appendix C.

5.2.3 Step 3: Extracting Java API Classes

To extract the Java API classes, the first three heuristic rules used by Study 1 (Table 4.1 in Chapter 4) and two additional heuristic rules were used (Table 5.2). The first additional heuristic rule is to extract and validate API classes candidates produced by the first three heuristic rules. All API classes candidates were checked against the classes' names found in Java API documentation for Java Standard Edition (JavaSE) 8 and Java Enterprise Edition (JavaEE) 8. If an API class candidate is in wrong letter cases, it will be converted to the correct letter cases. It is important to note that, different from heuristic rule 4 in Study 1, this heuristic rule does not remove API class candidates which are not listed in the Java API documentation. This is because these API classes candidate could be from third-parties Java API libraries and relevant to a user's query and should appear in the recommended API classes.

The second additional heuristic rule is removing two high occurrence API classes, which are, *String* and *ArrayList* classes. These two classes (especially the *String* class) are used in most programs and is less likely to be addressing any specific programming question or task.

The Java API classes that remain at the end of the application of the five heuristic rules are stored in the Answers dataset's "APIClasses" column at the respective row.

Table 5.2: Additional Heuristic Rules for Extracting Java API Classes

No.	Heuristic Rule	Justification	Example	
			Input	**Java API class extracted by the heuristic rule**
1	Extract and validate API class candidates produced by the first three heuristic rules by checking with API class names extracted from the Java API documentation for Java SE 8 and Java EE 8.	All valid Java API class candidates could be in the wrong letter cases. Therefore, all API class candidates were checked against the valid API class names extracted from the API documentation and converted to the correct letter cases if wrong letter cases occurred. This heuristic rule helps to eliminate invalid API class candidate.	inputStream INPUTStream InputStem	- InputStream InputStem
2	Remove *String* and *ArrayList* classes	These two classes (especially the *String* class) are used in most programs and is less likely to be addressing any specific programming question. Removing them could improve relevant API class candidates.	InputStream String ArrayList	InputStream - -

5.2.4 Step 4: Training Word Embedding Model

There is a number of steps involved in training the word embedding model: i) filtering of data; ii) creating tagged document; iii) training with Doc2Vec algorithm. The purpose of filtering of data is to reduce the Questions dataset to include only those questions having the occurrence of Java API classes in the code snippets in their accepted answers. This was done by checking against the Answers dataset that contains the Java

API classes based on the "QuestionId". The resulting Questions dataset consists of only 178,159 questions, which is 28% of the original size of the Questions dataset.

In the second step of training the word embedding model, a tagged document containing two information, the indexes of the questions and the questions' titles, was created. The indexes of the questions were created in an ascending order based on the order of the questions in the Questions dataset.

In the final step of training the word embedding model, the tagged document was used as the input data to train the *Doc2Vec* word embedding model by using the *Doc2Vec* algorithm. The *Doc2Vec* algorithm was implemented in Python programming language by using the open source *Gensim* library (Rehurek & Sojka, 2010). After the model was trained, it was saved into a binary file format to be used later by the recommendation phase in retrieving similar questions. The resulting trained model treated each of the question as a document and contained the similarity scores between each of the questions.

This research used *Doc2Vec* algorithm because it calculates word similarity at the "sentence level". It is able to calculate a continuous distributed vector representation for pieces of texts, which makes it a better algorithm than *Word2Vec*.

5.3 Recommendation Phase

There are three major steps in the recommendation phase:

1. Pre-process user query;
2. Retrieve similar questions using *Doc2Vec* model and retrieve their respective answers and Java API classes that have been extracted in the preparation phase,

3. Select relevant Java API classes from the set of Java API classes by using the topic modelling algorithm, Latent Dirichlet Allocation (LDA) (Blei et al., 2003) and return a list of ranked Java API classes to the user.

5.3.1 Step 1: Pre-processing User Query

In this step, a user's query is pre-processed the same way the training data was pre-processed in the preparation phase and for the same reason. This involves the tokenization, removal of stop words and punctuation marks, and lemmatization (Section 5.2.2).

5.3.2 Step 2: Retrieving Similar Questions and Their Respective Answers and Java API Classes

In this step, the remaining word tokens in the pre-processed user query are used to retrieve a set of similar questions from the questions dataset by using the *Doc2Vec* model. The *Doc2Vec* model calculates the similarity score between the query's tokens and returns a set of similar questions. At most one hundred questions having similarity score above seventy percent are returned because subsequent questions not in the top one hundred and having lower similarity score are less relevant to the respective user's query.

The following explains what happen in Steps 1 and 2 by using an example, with the results shown in Figure 5.2. The query is "How to create a digital signature and sign data?". First, the query is pre-processed where it was tokenized, stop words and punctuation marks were removed, and lemmatized. The five remaining word tokens are, 'create', 'digital', 'signature', 'sign' and 'data'. This is shown as the first line of output in Figure 5.2.

Subsequently, these word tokens were compared with the questions dataset by using *Doc2Vec* model. *Doc2Vec* returned a set of question indexes having similarity scores above seventy percent. The second line of output in Figure 5.2 shows the index ('92907') of the similar question from the tagged document and the similarity score of 0.758030891418457 between the word tokens and this similar question. The third line of output in Figure 5.2 shows the question's identifier or "QuestionId" ('10703416') of the similar question retrieved from the Questions dataset by using the question index ('92907') from the tagged document (Step 2a in Figure 5.1). The identifiers of similar questions are used to retrieve their corresponding answers and Java API classes from the Answers dataset (Step 2b and Step 2c in Figure 5.1).

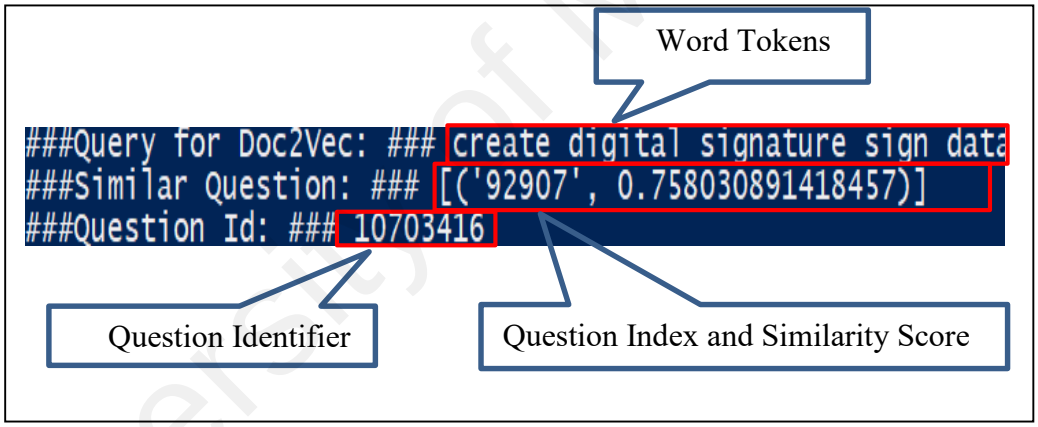


Figure 5.2: An Example of Output of Step 2 of Recommendation Phase

5.3.3 Step 3: Selecting Relevant Java API Classes and Return a Ranked List of Java API Classes

To select relevant Java API classes from the Java API classes produced by the previous step, topic modelling is used to measure how relevant the Java API classes are to the respective query. The basic LDA algorithm is used in this research for topic modelling since it is widely used for automatic extraction of topics from a corpus of text documents and a topic is a collection of words that co-occurred frequently in the

documents of the corpus (Chen et al., 2016). Therefore, the LDA algorithm was implemented using Python programming language and open source *Gensim* library (Rehurek & Sojka, 2010).

For each of query, the Java API classes extracted from the previous step are used as the input to the LDA algorithm to produce an output comprising of a single topic and ten words (with probability scores) that are related to the topic for the particular query. The rationale for a single topic is, generally a user's query contains the description of a single programming problem. The ten words that are related to the topic correspond to the top-10 Java API classes that are relevant to the query.

The following explains what happens in Step 3 by using the same example used to explain Steps 1 and 2. The output in Figure 5.3 shows a single topic at index 0 with 10 words (Java API class candidates) that are relevant to the topic of the query in descending order of probability scores: 'Signature' with the probability score of 0.022, 'InputStream' with the probability score of 0.017, 'CMSSignedData' with the probability score of 0.015, and so on. The higher the probability score of a word, the more likely the word is related to the topic. For example, 'Signature' word having the highest probability score is the most relevant word to the topic of the respective query.

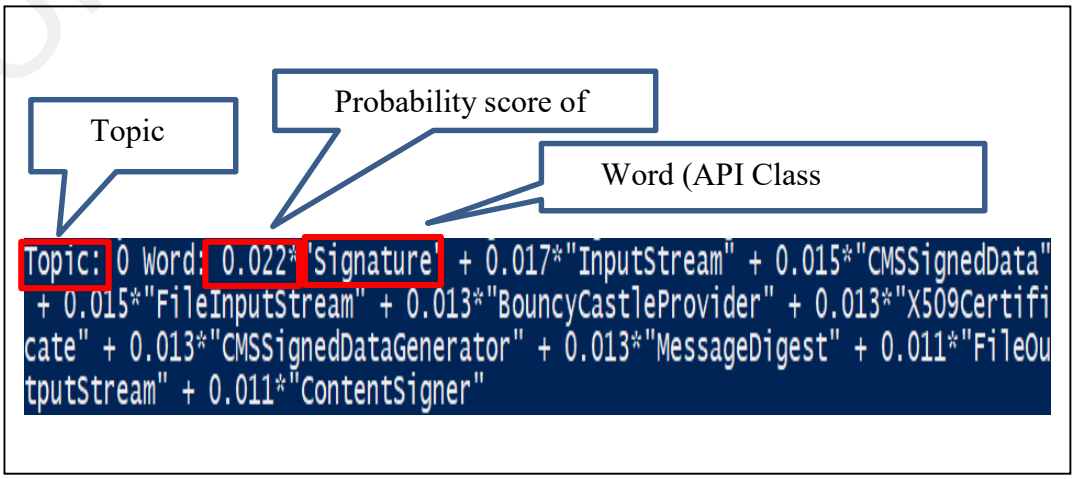


Figure 5.3: An Example of Output of Step 3 of Recommendation Phase

5.4 Summary of Techniques used in the Proposed Approach

Table 5.3 summarizes the techniques used in the proposed approach in the same format as Table 2.2 where other existing approaches of API Elements Search are compared.

Table 5.3: Summary of Techniques used in Proposed Approach

Studies	Data Pre-processing	Capturing of context	Producing Recommendations	Presenting Recommendations
The Proposed Approach	• Tokenization • Removal of stop words and punctuation marks • Lemmatization	• Doc2Vec • Five heuristic rules	• LDA Topic Modelling	• Top-K results

For the proposed approach, there is a total of seven techniques being employed. It is important to note that as illustrated in Table 2.2, existing studies employed nine to ten different techniques in their approaches. In comparison, with fewer techniques the proposed approach is less complex, and yet achieves better benchmarking results. For data pre-processing, the approach employs tokenization, removal of stop words and punctuation marks and lemmatization to the raw data. For capturing of context from the SO questions and answers, the approach employs Doc2Vec word embedding algorithm to calculate similarity scores between questions retrieved and produces a Doc2Vec model. In addition, five heuristic rules were employed to extract Java API classes from the answers of the questions. For producing the recommendations, the proposed approach employs LDA topic modelling algorithm in selecting relevant Java API classes. For presenting the recommendations, the proposed approach employs a list of top-10 ranked Java API classes.

5.5 Development of the Java API Class Recommender and the Plug-in

The approach was developed as a Java API class recommender, which is a web application that runs on a back-end server. A plug-in for Eclipse IDE, named *APIRecJ* was developed to serve as the front-end to the recommender.

5.5.1 Architecture Design of Java API Class Recommender and the Plug-in

The plug-in works as the front-end that interacts with the users to obtain their query and to display results to them. The plug-in passes the query and invokes the recommendation service provided by the Java API class recommender to initiate the searching of relevant Java API classes for the query.

The Java API class recommender is a standalone web application running on *Flask* web framework. The recommender accesses the pre-trained *Doc2Vec* model to find questions similar to the query, and retrieves their respective answers and Java API classes, selects relevant Java API classes and returns a ranked list of Java API classes to the plug-in for display on the plug-in's user interface.

Figure 5.4 illustrates the deployment architecture design of the plugin, *APIRecJ* and the Java API class recommender. The Java API class recommender web application and *APIRecJ* can be deployed on different physical machines running Windows operating system. Both of the machines are integrated in the same network using a client-server architecture style via Transmission Control Protocol/Internet Protocol (TCP/IP).

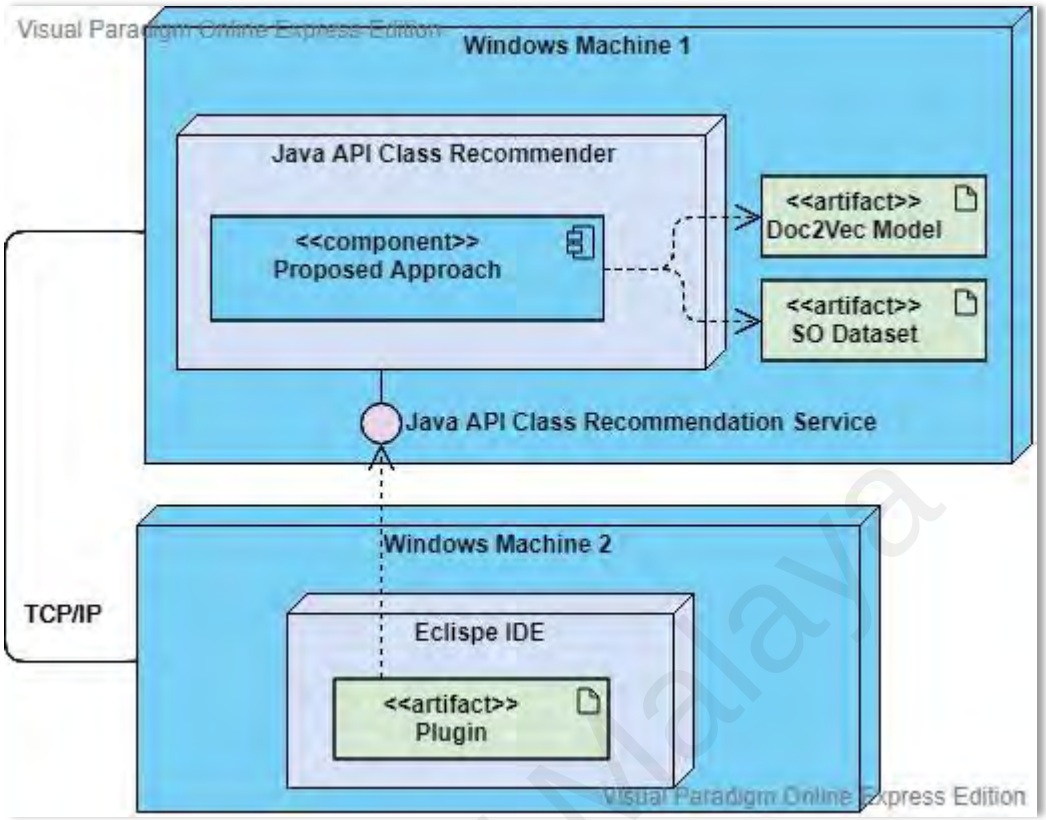


Figure 5.4: Deployment Architecture of the Plug-in and the Java API Class Recommender

5.5.2 Requirements of the Java API Class Recommender

This section presents the functional requirements and non-functional requirements of the Java API Class Recommender. Functional requirements describe the functionalities provided by the recommender. Non-functional requirements are requirements concerning the quality attributes of the recommender or the recommender's operation such as its performance.

Both functional and non-functional requirements of are tested using plug-in. These testing are manually verified by checking on the request message (user query) and response (list of API classes and similar questions) displayed on the plug-in user interface.

5.5.2.1 Functional Requirements

Table 5.4 shows the functional requirements of the Java API Class Recommender.

Table 5.4: Functional Requirements of Java API Class Recommender

No.	Functional Requirements	Description
1.	Get client request message	To obtain client request message which contains a user's query.
2.	Find Java API classes	To pass the user's query to the incorporated proposed approach for recommending Java API classes.
3.	Return Java API classes and similar SO questions' identifier	To return Java API classes and similar SO questions' identifier in a single response

5.5.2.2 Non-Functional Requirements

Table 5.5 shows the non-functional requirements of the Java API Class Recommender.

Table 5.5: Non-Functional Requirements of Java API Class Recommender

No.	Non-Functional Requirement	Description
1.	Performance	The recommender loads or re-loads the pre-trained *Doc2Vec* model within 10 seconds and returns the results within 2-5 seconds.

2.	Reliability	The recommender receives and processes a single request at a time with less than 10 seconds response time. If not, recommender will respond with an error message to the client's request.

5.5.2.3 Testing on Functional and Non-Functional Requirements

Table 5.6, Table 5.7 and Table 5.8 shows the test cases for functional requirements of the Java API Class Recommender. Table 5.9 and Table 5.10 shows the test cases for non-functional requirements of the Java API Class Recommender.

Table 5.6: Test Cases for Functional Requirement (1) of Java API Class Recommender

Test Objective:	Get client request message
Description:	Request message from plug-in is received.
Inputs:	Query 1: How to write an Object to file in Java? Query 2: How do I reverse the order of array elements?
Steps:	1. Open Eclipse. 2. Open APIRecJ plug-in. 3. Provide "Query 1" or "Query 2" in "Question" text box.

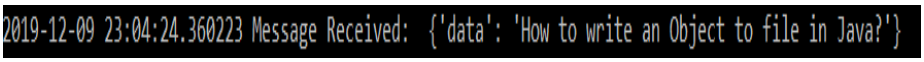
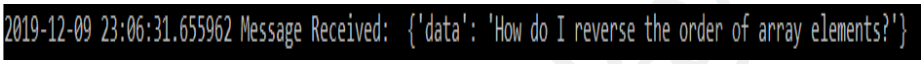
	4. Observe message on Java API class recommender's output console.
Expected Outputs:	Message received by Java API class recommender and print on output console.
Actual Outputs:	Query 1: How to write an Object to file in Java?  Query 2: How do I reverse the order of array elements? 
Result:	Pass

Table 5.7: Test Cases for Functional Requirement (2) of Java API Class Recommender

Test Objective:	Find Java API classes.
Description:	Extract user query from request message for recommendation search and perform recommendation.
Inputs:	Query 1: How to write an Object to file in Java? Query 2: How do I reverse the order of array elements?
Steps:	1. Open Eclipse. 2. Open APIRecJ plugin. 3. Provide "Query 1" or "Query 2" in "Question" text box. 4. Observe message on Java API class recommender's output console.

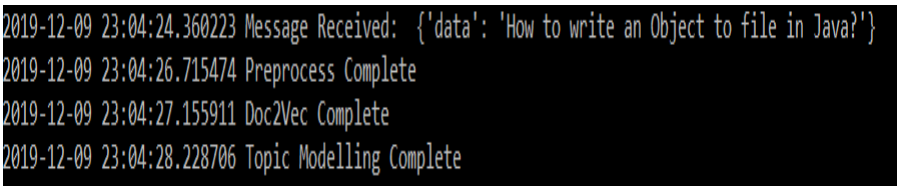
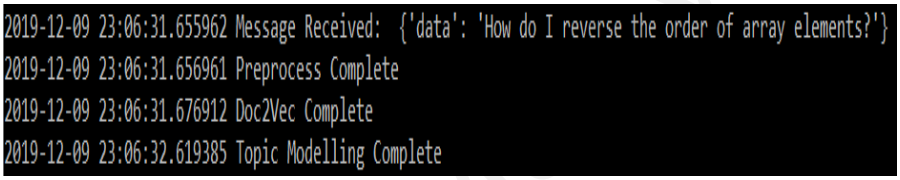
Expected Outputs:	Recommendation steps print on output console.
Actual Outputs:	Query 1: How to write an Object to file in Java?  Query 2: How do I reverse the order of array elements? 
Result:	Pass

Table 5.8: Test Cases for Functional Requirement (3) of Java API Class Recommender

Test Objective:	Return Java API classes and similar SO questions identifier
Description:	Returned response message with a list of Java API classes and similar SO questions identifier to plug-in.
Inputs:	Query 1: How to write an Object to file in Java? Query 2: How do I reverse the order of array elements?
Steps:	1. Open Eclipse 2. Open APIRecJ plugin 3. Provide “Query 1” or “Query 2” in “Question” text box.

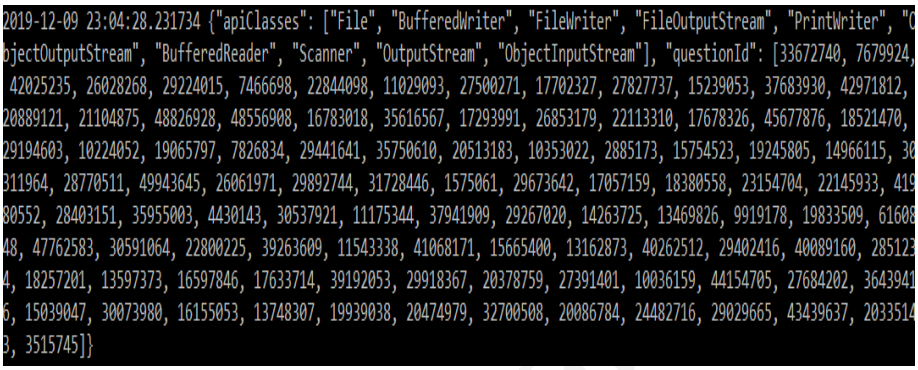
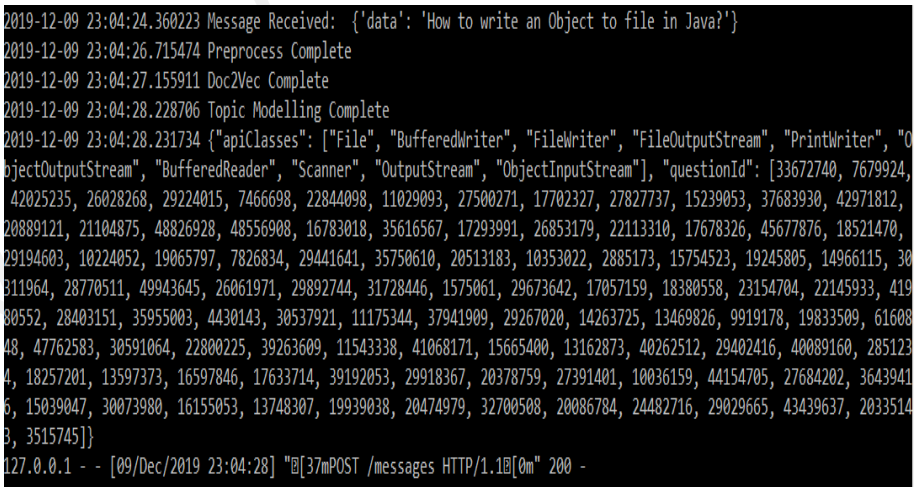
	4. Observe message on Java API class recommender's output console.
Expected Outputs:	Return message with a list of Java API classes and similar SO questions identifier print on output console.
Actual Outputs:	Query 1: How to write an Object to file in Java?  Query 2: How do I reverse the order of array elements? 
Result:	Pass

Table 5.9: Test Cases for Non-Functional Requirement (1) of Java API Class Recommender

Test Objective:	Performance
Description:	The recommender returned the results less than 10 seconds when loads or re-loads the pre-trained *Doc2Vec*. The recommender returned the

	results within 2-5 seconds without loads or re-loads the pre-trained *Doc2Vec*.
Inputs:	Query 1: How to write an Object to file in Java? Query 2: How do I reverse the order of array elements?
Steps:	1. Open Eclipse 2. Open APIRecJ plugin 3. Provide “Query 1” or “Query 2” in “Question” text box. 4. Observe message on Java API class recommender’s output console.
Expected Outputs:	Message returned within 10 seconds
Actual Outputs:	Query 1: How to write an Object to file in Java? 

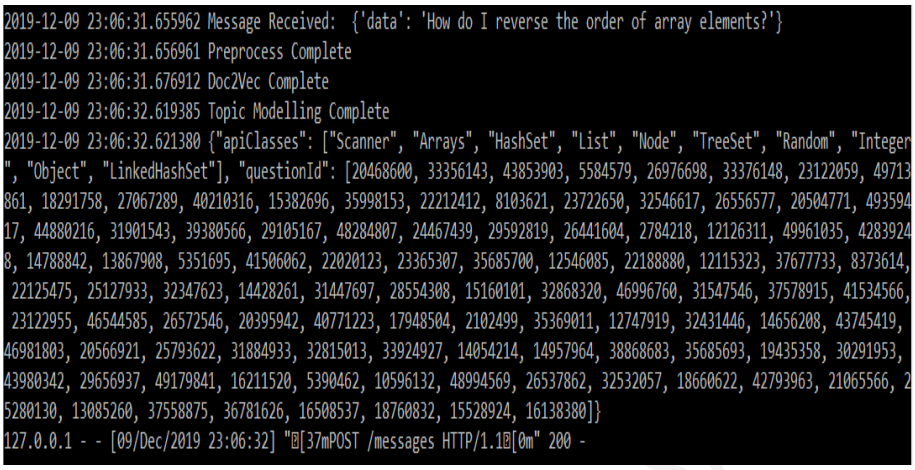
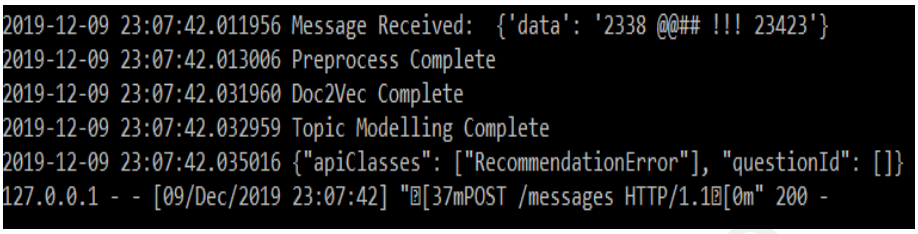
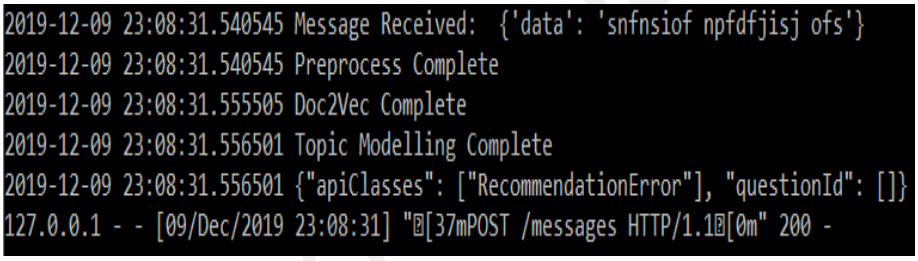
	Query 2: How do I reverse the order of array elements? 
Result:	Pass

Table 5.10: Test Cases for Non-Functional Requirement (2) of Java API Class Recommender

Test Objective:	Reliability
Description:	When recommendation failed, recommender responds with an error message to the client's request.
Inputs:	Query 1: 2338 @@## !!! 23423 Query 2: snfnsiof npfdfjisj ofs
Steps:	1. Open Eclipse 2. Open APIRecJ plugin 3. Provide "Query 1" or "Query 2" in "Question" text box. 4. Observe message on Java API class recommender's output console.

Expected Outputs:	Returns error message with "RecommendationError" value for API classes and empty value for list of question identifier
Actual Outputs:	Query 1: 2338 @@## !!! 23423  2019-12-09 23:07:42.011956 Message Received: {'data': '2338 @@## !!! 23423'} 2019-12-09 23:07:42.013006 Preprocess Complete 2019-12-09 23:07:42.031960 Doc2Vec Complete 2019-12-09 23:07:42.032959 Topic Modelling Complete 2019-12-09 23:07:42.035016 {"apiClasses": ["RecommendationError"], "questionId": []} 127.0.0.1 - - [09/Dec/2019 23:07:42] "[37mPOST /messages HTTP/1.1[0m" 200 - Query 2: snfnsiof npfdfjisj ofs  2019-12-09 23:08:31.540545 Message Received: {'data': 'snfnsiof npfdfjisj ofs'} 2019-12-09 23:08:31.540545 Preprocess Complete 2019-12-09 23:08:31.555505 Doc2Vec Complete 2019-12-09 23:08:31.556501 Topic Modelling Complete 2019-12-09 23:08:31.556501 {"apiClasses": ["RecommendationError"], "questionId": []} 127.0.0.1 - - [09/Dec/2019 23:08:31] "[37mPOST /messages HTTP/1.1[0m" 200 -
Result:	Pass

5.5.3 Requirements of the Plug-in

This section presents the functional requirements and non-functional requirements of the plug-in, named *APIRecJ*. There are a variety of Java IDE available in the market, for example, Eclipse (2019), IntelliJ IDEA (2019), NetBeans (2019) and others. In this research, the plug-in is specifically built for *Eclipse* IDE since it is more suitable for real world applications, execute Java APIs with high performance and provides free subscription as compared to *IntelliJ IDEA* which can be costly (Al-Jepoori & Bennett, 2018). Besides that, there is an existing study that claimed that Eclipse is better than NetBeans because less start up time is required and it is simple to get started with (Kavitha & Sindhu, 2015).

5.5.3.1 Functional Requirements

Table 5.11 shows the functional requirements of the plug-in.

Table 5.11: Functional Requirements of Plug-in

No.	Functional Requirements	Description
1.	Get user query	To obtain the user's query.
2.	Find relevant Java API classes	To pass the user's query to the Java API class recommender and access it's functionalities for recommending Java API classes.
3.	Display relevant Java API classes	To display Java API classes that are relevant to the user's query that are returned by the Java API class recommender.
4.	Display similar SO questions	To display SO questions similar to the user's query that are returned by the Java API class recommender.
5.	Filter similar SO questions	To display similar SO questions for Java API classes that are selected by the user only.

An exemplar usage of the plug-in is as follows: A user enters his or her natural language query describing the programming question or problem faced through the user interface of the plug-in. The plug-in obtains the query and sends a request (together with the query as input) to the Java API class recommender running on the server. The recommender processes the request and returns a response comprising a list of recommended Java API classes and the corresponding similar SO questions to the plug-

in. At most ten Java API classes would be recommended and at most one hundred similar SO questions would be returned to the plug-in.

5.5.3.2 Non-Functional Requirements

Table 5.12 shows the non-functional requirements of the plug-in. It is important to note that these non-functional requirements are also dependent on the performance and reliability of the recommender running on the server.

Table 5.12: Non-Functional Requirements of Plug-in

No.	Non-Functional Requirement	Description
1.	Performance	The plug-in will display the results within 2-5 seconds and within 10 seconds when the recommender loads or re-loads the pre-trained *Doc2Vec* model.
2.	Reliability	The plug-in will display an error message to the user if an error message is returned by the recommender or connection error occurred when connecting to the recommender.

5.5.4 Implementation of the Java API Class Recommender and the Plug-in

The proposed approach was implemented in the Java API class recommender developed as a web application. The web application was developed and deployed using *Flask (2019)*, which is a web framework written in Python programming language. A web framework is a software framework that supports that development of web

applications and provides a standard way to build and deploy web applications (2019). The recommender listens to requests from client and responds with a set of Java API classes and similar questions in JavaScript Object Notation (JSON) format. In order to run the web application on any Windows-based machine without Python installation, the web application and all its dependencies files were packaged into a single execution file with ".exe" extension by using the open source *PyInstaller (2019)* library.

The plug-in was developed by creating an Eclipse plug-in project. Its user interface was designed using Java Swing components. Input to the backend server is sent as a POST request. The completed Eclipse plug-in project was then exported as an Eclipse feature project to compile the Eclipse plug-in project into a single compressed file. The compressed file is used to perform a new plugin installation in an Eclipse IDE through the installation manager.

5.5.4.1 User Interface of Plug-in

Figure 5.5 illustrates the four main sections of the plug-in's user interface. In the figure, they are labelled with numbers. Section 1 comprises an input textbox for users to enter their programming task question or query, and a "Find" button for them to initiate the search for Java API classes relevant to the query.

Section 2 displays the top-10 Java API classes returned or recommended by the Java API Class Recommender developed by this research in terms of checkboxes. A user can choose the Java API classes that they are interested in by selecting the corresponding checkboxes. This will cause the plug-in to display only similar SO questions for the chosen Java API classes and exclude similar SO questions for non-chosen Java API classes.

Section 3 displays a list of the titles of the similar SO questions that are associated to the Java API classes selected by the user in Section 2. The user can click on a question title in the list and Section 4 will display the details of the corresponding question. The exact question on SO is located by appending the identifier to hyperlink in the following format, “https://stackoverflow.com/q/identifier”, where identifier value is the question identifier.

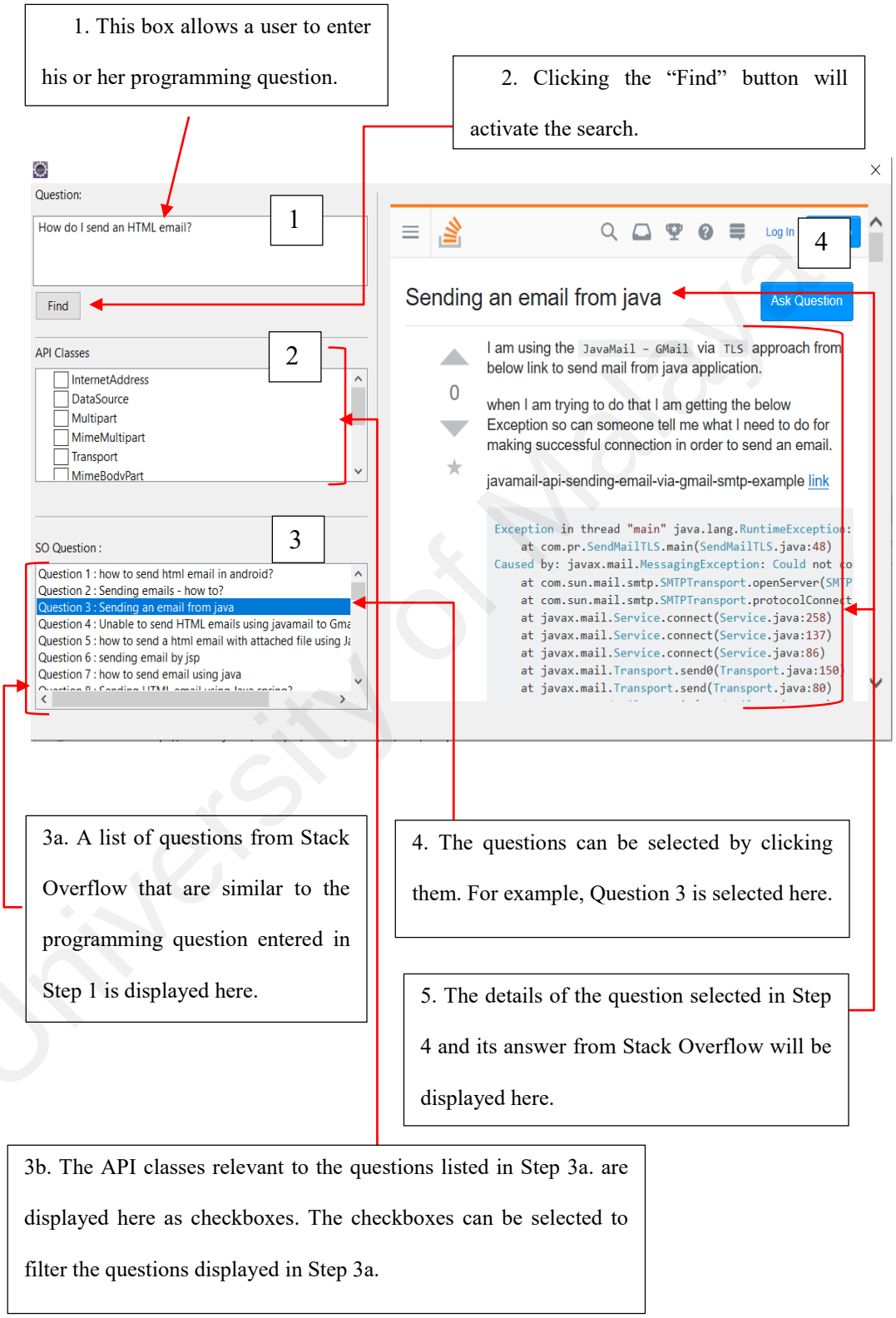


Figure 5.5: User Interface Design of the Plug-in

5.6 Chapter Summary

In brief, this chapter discusses on the overall design of the proposed approach and summarizes all the techniques employed in this study. Moreover, this chapter details the phases, input and output in each step of the proposed approach. Furthermore, the techniques used in the proposed approach and existing studies are summarized based on major steps of designing RSSE. Then, the proposed approach is incorporated into Java API class recommender and followed by developing an IDE plug-in. All the essential plug-in requirements are listed out, which include functional requirements and non-functional requirements. Next, architecture design of plug-in's integration with Java API class recommender using deployment diagram and the user interface of the plug-in are illustrated. The next chapter discusses on how evaluation is performed on the proposed approach.

CHAPTER 6: PERFORMANCE AND BENCHMARKING OF APPROACH

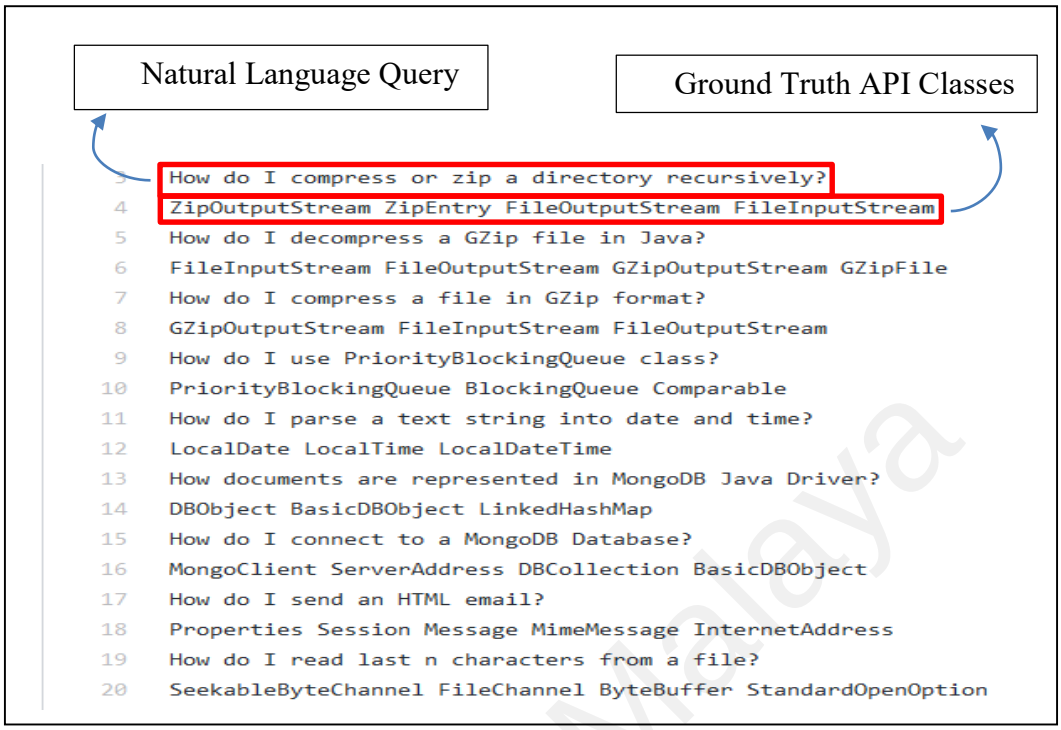
This chapter discusses the evaluation performed on the proposed approach. The evaluation study mainly intended to measure the performance of the proposed approach in recommending relevant API classes. Besides, the same evaluation dataset and evaluation metrics in prior study are used to obtain a comparable result. At the end of the evaluation, the results obtained are discussed and benchmarked against existing baseline studies.

6.1 Evaluation Dataset

To benchmark the proposed approach, this research used the evaluation dataset (2019a) provided by *NLP2API* study (Rahman et al., 2018), the most recent work on mining SO data for Java API classes recommendation. This evaluation dataset contains 310 code search queries and the corresponding Java API classes relevant to the respective query. Code search queries are queries that are described in natural language and used for searching for relevant code snippets, and are termed "natural language queries" in this research. The API classes found in the relevant code snippets are known as the ground truth API classes for the respective query.

The natural language queries in the evaluation dataset and their ground truth Java API classes were extracted from programming tutorial websites, for instances, KodeJava (2019), CodeJava (2019) and Java2s (2019). For example, the query "How do I compress or zip a directory recursively?" has "ZipOutputStream", "ZipEntry", "FileOutputStream" and "FileInputStream" as its ground truth Java API classes. Figure 6.1 shows some examples of the queries in the evaluation dataset.

Figure 6.1: Examples of the Queries in the Evaluation Dataset



6.2 Evaluation Metrics

The existing studies of the same genre as this research, namely, *RACK* (Rahman et al., 2016) and *NLP2API* (Rahman & Roy, 2018) used four metrics to evaluate the performance of their approaches in recommending API classes for queries. *BIKER* (Huang et al., 2018) used two out of the four metrics to evaluate their approaches in recommending API classes and methods.

This research used the same four metrics to benchmark the performance of the proposed approach. These metrics are:

(a) Top-K accuracy

(b) Mean Recall @ K (MR@K)

(c) Mean Reciprocal Rank @ K (MRR@K)

(d) Mean Average Precision @ K (MAP@K).

Top-K accuracy, MR@K, MRR@K and MAP@K are the evaluation metrics that measure on the recommendation performance and information retrieval (IR) performance of the proposed approach. These metrics not only use to assess the correctness of the proposed approach in retrieve set of relevant API classes but also examine on whether the proposed approach able to return ranked result by select the relevant result at top position and less relevant result at the bottom position. Thus, the proposed approach should archive a high score in all the performance metrics to prove that the approach has return majority highly relevant API classes and less irrelevant result. In addition, the approach should also able to return user a list of relevant items earlier and user need not to browse through the entire list to search for relevant answer.

In the following subsections, the metrics are explained by using two query examples. The depth of the returned result is set at ten for all the metrics. In other words, the value of K is ten and only the top-10 Java API classes returned by the approach are taken into consideration. This is because the baseline studies (*RACK* and *NLP2API*) achieved the best performance for the four metrics when k is 10.

6.2.1 Top-K accuracy

Top-K accuracy refers to the percentage of the search query for which at least one API class is correctly recommended within the Top-K results by recommendation technique (Rahman et al., 2018). Its formula is defined in Equation (Eq) 1 (Rahman et al., 2018).

Eq 1:

$$\text{Top-K Accuracy } (Q) = \frac{\sum_{q \in Q} isCorrect(q,\ K)}{|Q|}\ \%$$

Q denotes the set of all queries, q denotes a query that is a member of ($\in$) set Q, $|Q|$ denotes the size of the set of query, $\sum$ denotes the sum of *isCorrect* function for each q in Q, and K denotes the top-k Java API classes returned by the approach. The *isCorrect* function returns a value of 1 if the approach returns at least one relevant Java API class for query q, and a value of 0 if the approach returns none of the relevant Java API classes for query q. A relevant Java API class refers to a Java API class that can be found in the set of ground truth Java API classes for query q.

Table 6.1 shows an example of calculation for Top-K Accuracy metric with K equals to 10. For Query 1, *isCorrect* function returns a value of 0 since none of the Java API classes returned by the approach (i.e. "List" and "ArrayList") matches with any of the ground truth Java API classes. For Query 2, *isCorrect* function returns a value of 1 since at least one (in fact two in this case) of the Java API classes returned by the approach (i.e. "Properties" or "Session") matches with ground truth Java API classes. The Top-K accuracy of the approach for this dataset that comprises only two queries is 50%, and is calculated by dividing the number of queries having value 1 returned by the *isCorrect* function with the total number of queries in the dataset and then multiplying the output with 100.

Table 6.1: Example of Calculation for Top-10 Accuracy

Query	**Ground Truth Java API Class**	**Results Returned by Approach**	**Calculation**
Query 1: How to create a digital signature and sign data?	Signature, PrivateKey, KeyPairGenerator SecureRandom, KeyPair	List, ArrayList	$isCorrect(\text{q1}, (\text{"List"}, \text{"ArrayList"}))$ = 0
Query 2: How do I send an HTML email?	**Properties**, **Session**, Message, MimeMessage, InternetAddress	**Properties**, **Session**	$isCorrect(\text{q2}, (\text{"Properties"}, \text{"Session"}))$ = 1

		Top-10 Accuracy (Q)	$\frac{\sum_{q \in Q} isCorrect(q,K)}{\lvert Q \rvert} x\ 100$ $= \frac{0+1}{\lvert 2 \rvert} x\ 100$ $= \frac{1}{2} x\ 100$ = 50 %

6.2.2 Mean Recall @ K (MR@K)

Recall@K refers to the percentage of ground truth Java API classes that are correctly recommended for a query in the Top-K results by an approach (Rahman et al., 2018). MR@K averages such measures for all queries in the dataset. Its formula is defined in Eq 2 (Rahman et al., 2018).

Eq 2:

$$\text{MR@K}\ (Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{|result(q,K) \cap ground(q)|}{|ground(q)|}$$

Q denotes the set of all queries, q denotes a query that is a member of ($\in$) set Q, $|Q|$ denotes the size of the set of query, *result(q, K)* refers to Top-K recommended APIs by the approach, and *ground(q)* refers to ground truth API classes for each query q $\in$ Q. The larger the value of MR@K, the better the recommendation approach is.

Table 6.2 shows an example of calculation for MR@K metric with K equals to 10. The recall@10 for Query 1 is 0 since none of the Java API classes returned by the approach (i.e. “List” and “ArrayList”) matches with any of Query 1’s ground truth Java API classes. The recall@10 for Query 2 is 0.40 since the two Java API classes returned

by the approach (i.e. “Properties” or “Session”) matches with two of the five ground truth Java API classes of Query 2. The MR@10 of the proposed approach for this dataset that comprises only two queries is 0.20, and is calculated by dividing the sum of recall@10 for each query in the dataset with the total number of queries in the dataset.

Table 6.2: Example of Calculation for MR@10

Query	**Ground Truth Java API Class**	**Results Returned by Approach**	**Calculation**
Query 1: How to create a digital signature and sign data?	Signature, PrivateKey, KeyPairGenerator SecureRandom, KeyPair	List, ArrayList	$\frac{\lvert result(q1,K) \cap ground(q1) \rvert}{\lvert ground(q1) \rvert} = \frac{0}{5} = 0$
Query 2: How do I send an HTML email?	**Properties**, **Session**, Message, MimeMessage, InternetAddress	**Properties**, **Session**	$\frac{\lvert result(q2,K) \cap ground(q2) \rvert}{\lvert ground(q2) \rvert} = \frac{2}{5} = 0.40$
		MR@10 (Q)	$\frac{1}{\lvert Q \rvert} \sum_{q \in Q} \frac{\lvert result(q,K) \cap ground(q) \rvert}{\lvert ground(q) \rvert}$ $= \frac{1}{\lvert 2 \rvert} (0 + 0.4)$ $= \frac{0.4}{2}$ $= 0.2$

6.2.3 Mean Reciprocal Rank @ K (MRR@K)

Reciprocal Rank @ K refers to the multiplicative inverse of the rank of the first relevant API class in the Top-K results returned by the approach (Rahman et al., 2018). Mean Reciprocal Rank @ K averages such measures for all queries (q ∈ Q) in the dataset. Its formula is defined in Eq 3 (Rahman et al., 2018).

Eq 3:

$$\text{MRR@K}\ (Q) = \frac{1}{|Q|}\sum_{q \in Q}\frac{1}{rank(q,K)}$$

Q denotes the set of all queries, q denotes a query that is a member of ($\in$) set Q, $|Q|$ denotes the size of the set of query, *rank(q, K)* returns the rank of the first correct API from a ranked list of size K. If no correct API class or code segment is found within the Top-K positions, then *rank(q, K)* returns ∞. It returns 1 for the correct result at the topmost position of a ranked list. Following that, MRR can take a maximum value of 1 and a minimum value of 0. The bigger the MRR value is, the better the approach is.

Table 6.3 shows an example of calculation for MRR@K metric with K equals to 10. For Query 1, *rank(Query 1, 10)* returns a value of ∞ since none of the Java API classes returned by the approach (i.e. "List" and "ArrayList") matches with any of the ground truth Java API classes. Therefore, Reciprocal Rank @ 10 for Query 1 is 0. For Query 2, *rank(Query 2, 10)* returns a value of 1, since the two Java API classes returned by the approach (i.e. "Properties" or "Session") matches with two of the five ground truth Java API classes and the rank of the first relevant API class returned (i.e. "Properties") is 1, which is the topmost position of the ranked list returned by the approach. The MRR@10 of the proposed approach for this dataset that comprises only two queries is 0.5, and is calculated by dividing the sum of Reciprocal Rank @ 10 for each query in the dataset with the total number of queries in the dataset.

Table 6.3: Example of Calculation for MRR@10

Query	Ground Truth Java API Class	Results Returned by Approach	Calculation
Query 1: How to create a digital signature and sign data?	Signature, PrivateKey, KeyPairGenerator SecureRandom, KeyPair	List, ArrayList	$\frac{1}{rank(q,K)} = \frac{1}{\infty} = 0$
Query 2: How do I send an HTML email?	**Properties**, **Session**, Message, MimeMessage, InternetAddress	**Properties**, **Session**	$\frac{1}{rank(q,K)} = \frac{1}{1} = 1$
		MRR@10 (Q)	$\frac{1}{\lvert Q\rvert}\sum_{q\in Q}\frac{1}{rank(q,\ K)}$ $= \frac{1}{\lvert 2\rvert}(0+1)$ $= \frac{1}{2}$ $= 0.5$

6.2.4 Mean Average Precision @ K (MAP@K)

Precision @ K calculates the precision at the occurrence of every single relevant API class in the ranked list (Rahman et al., 2018). Average Precision @ K (AP@K) averages the precision @ K for all relevant items within Top-K results for a particular query. Mean Average Precision @ K is the mean of Average Precision @ K for all queries (Q) from the dataset and the formula is defined in Eq 4 (Rahman et al., 2018).

Eq 4:

$$\text{AP @ K} = \frac{\sum_{k=1}^{K} P_k \times rel_k}{|RR|}$$

$$\text{MAP @ K} = \frac{\sum_{q \in Q} AP@K(q)}{|Q|}$$

K refers to number of top results considered, rel_k denotes the relevance function of k^{th} result in the ranked list that returns either 1 (relevant) or 0 (irrelevant) and P_k denotes the precision at k^{th} result. |RR| is the set of relevant results for a query. Q denotes the set of all queries, q denotes a query that is a member of ($\in$) set Q, $|Q|$ denotes the size of the set of queries.

Table 6.4 shows an example of calculation for MAP@K metric with K equals to 10. AP@10 for Query 1 is 0 since none of the Java API classes returned by the approach (i.e. “List” and “ArrayList”) matches with any of the ground truth Java API classes.

For Query 2, the first API class returned is relevant ($rel_1 = 1$) and it can be found at the first position of the five ground truth Java API classes, so $P_1 \times rel_1 = \frac{1}{1} \times 1 = 1$. The second API class returned is also relevant ($rel_2 = 1$) and can be found at the second position out of the four remaining ground truth Java API classes, so $P_2 \times rel_2 = \frac{2}{2} \times 1 = 1$. Following that, AP@10 for Query 2 is 0.4. The MAP@10 of the proposed approach for this dataset that comprises only two queries is 0.2, and is calculated by dividing the sum of AP@10 for each query in the dataset with the total number of queries in the dataset.

Table 6.4: Example of Calculation for MAP@10

Query	**Ground Truth Java API Class**	**Results Returned by Approach**	**Calculation**
Query 1: How to create a digital signature and sign data?	Signature, PrivateKey, KeyPairGenerator SecureRandom, KeyPair	List, ArrayList	$\frac{\sum_{k=1}^{K} P_k \times rel_k}{\lvert RR \rvert}$ $= \frac{(\frac{0}{1}\times 0)+(\frac{0}{2}\times 0)+(\frac{0}{3}\times 0)+(\frac{0}{4}\times 0)+(\frac{0}{5}\times 0)}{5}$ $= 0$
Query 2: How do I send an HTML email?	**Properties**, **Session**, Message, MimeMessage, InternetAddress	**Properties**, **Session**	$\frac{\sum_{k=1}^{K} P_k \times rel_k}{\lvert RR \rvert}$ $= \frac{(\frac{1}{1}\times 1)+(\frac{2}{2}\times 1)+(\frac{2}{3}\times 0)+(\frac{2}{4}\times 0)+(\frac{2}{5}\times 0)}{5}$ $= \frac{1+1+0+0+0}{5}$ $= 0.4$
		MAP@10 (Q)	$\frac{\sum_{q\in Q} AP@K(q)}{\lvert Q \rvert}$ $= \frac{(0+0.4)}{\lvert 2 \rvert}$ $= \frac{0.4}{2}$ $= 0.2$

6.3 Benchmarking of Proposed Approach

This research used the evaluation tool (2019b) published by *NLP2API* authors to benchmark the proposed approach by using the four metrics and the benchmarking only considered the top-10 results returned by the approach. The benchmarking results show that the proposed approach achieves 84.83% for Top-10 Accuracy, 0.58 for MRR@10,

50.68% for MAP@10 and 58.76% for MR@10. The proposed approach shows an improvement of 3.22% in Top-10 Accuracy, 0.03 in MRR@10, 2.83% in MAP@10 and 0.89% in MR@10, when compared to the best state-of-the-art approach in Java API classes recommendation, namely, *NLP2API*.

Table 6.5 shows the results of the benchmarking. However, the proposed approach was not compared against the *BIKER* study because it focused on API methods recommendation and was not benchmarked using the *NLP2API*'s evaluation dataset. Furthermore, the *BIKER* study used only the MRR and MAP metrics in evaluating their work.

There are two possible reasons the proposed approach achieves higher scores for the four metrics. The first possible reason is the proposed approach validates the Java API classes using the Java documentation and this removes some of the invalid API classes from the result returned. Hence, the result comprises more accurate Java API classes and this contributes to the improvement in Top-10 accuracy and MR@10.

The second possible reason is the proposed approach removes two Java API classes ("String" and "ArrayList") that are of high occurrence but are unlikely to be the relevant Java API classes for the query. These two classes would probably occupy the top positions in the ranked list of Java API classes returned. By removing them, other relevant Java API classes can be recommended at a higher position on the list, resulting in better MRR@10 and MAP@10.

The proposed approach does not show significant improvement in MR@10. The proposed approach still returns some false negative results and is unable to remove all irrelevant Java API classes effectively. This is because the heuristic rules employed are unable to remove non Java API classes (for example, custom classes such as Student,

ClassA, TestClass, or classes from other APIs) that might appear in the sample codes in SO.

Table 6.5 : Benchmarking of Proposed Approach against Existing Approaches

Approach	Top-10 Accuracy	MRR@10	MAP@10	MR@10
RACK (Rahman et al., 2016)	77.10%	0.39	36.38%	39.22%
NLP2API (Rahman & Roy, 2018)	81.61%	0.55	47.85%	57.87%
Proposed Approach	84.83%	0.58	50.68%	58.76%
Proposed Approach's improvement over *NLP2API*	3.22%	0.03	2.83%	0.89%

6.4 Chapter Summary

This chapter details the evaluation dataset and four metrics (with examples) used to evaluate the performance of the proposed approach. This chapter also presents the benchmarking of the proposed approach with the existing studies by employing the four metrics. The benchmarking results proved that the proposed approach slightly outperforms baseline studies. The next chapter explains the user evaluation study performed on the developed plug-in.

CHAPTER 7: USER EVALUATION STUDY

This chapter presents the evaluation of plug-in as well as the developed Java API class recommender which incorporates the proposed approach in this research. There are mainly five subsections in this chapter. The first subsection describes the pilot study design and the second subsection discusses on the pilot study results. The third subsection demonstrates user evaluation study design and the fourth subsection illustrates the user evaluation results. Lastly, the fifth subsection concludes on the discussion of user evaluation results.

7.1 User Evaluation Study

This section presents the user evaluation study conducted to evaluate the usefulness of the plug-in (front-end) and the Java API class recommender running on the backend server. It also describes the results collected from the pilot study conducted prior to the user evaluation study.

7.1.1 Pilot Study Design

The purpose of the pilot study was to gather feedback to refine the plug-in and the design of the user evaluation study. Basically, participants were recruited to use *APIRecJ* to find relevant Java API classes for three programming questions given and to use *Google* search engine for the same purpose. The estimated duration of participation for the study is forty to forty-five minutes.

The participant recruitment criteria are:

1) The first programming language learned is Java, and

2) Obtained at least a grade 'B' in an introductory Java programming course.

The data collection instrument of the pilot study consists of the following sections:

- Introduction Section – This section provides an overview and the purpose of the study, recruitment criteria, terms and condition of participation, and researchers' contact details.
- Section 1 (Introduction to *APIRecJ*) - This section explains the features of *APIRecJ* by showing its UI and how to use the features.
- Section 2 (Part A) that consists of the tasks to be performed, namely, using Google search engine to search for Java API classes that are relevant to the three programming questions given; to state the start time and finish time of working on each programming question; and to state three most relevant Java API classes for each of the programming questions given.
- Section 2 (Part B) that consists of the tasks to be performed, namely, using *APIRecJ* to search for Java API classes that are relevant to the three programming questions given (same questions as Part A); and for each question, look for code snippets from the answers of the similar Stack Overflow questions returned by *APIRecJ* and list the Java API classes that are relevant to the question and state the three most relevant Java API classes for the programming question; to state the start time and finish time of working on each programming question.
- Section 3 comprising of a questionnaire with two parts. Part A of the questionnaire asks about participants' educational background, level of Java programming skill,

other programming languages known and the level of skill, and Software Development Kits (SDKs) familiar with. Part B of the questionnaire asks the participants' opinions on the features of *APIRecJ*, whether they prefer Google search engine or *APIRecJ* and their reasons for their preference, and usefulness of having API class recommender such as *APIRecJ* and the reasons.

The three programming questions given to the participants require the use of certain Java API classes. They are:

Programming Question 1: How to write an Object to file in Java?

Programming Question 2: How do I reverse the order of array elements?

Programming Question 3: How do I convert Date to String?

The order of Part A and Part B of Section 2 is reversed for alternate participant. In other words, the first participant will perform Part A followed by Part B, the second participant will perform Part B followed by Part A, the third participant will perform Part A followed by Part B, and so on.

7.1.2 Pilot Study Results

Two participants (denoted as P1 and P2) participated in the pilot study. Both of the participants are first year computer science undergraduate students from the Faculty of Computer Science & Information Technology, University Malaya, who scored an 'A' in the introductory programming subject (WIX1002 Fundamentals of Programming) they took in their undergraduate study. In terms of the level of Java programming skill, they chose the option of "junior" level that represents that they know about designing classes, interfaces and exception handling. In terms of other programming languages, P1 has

programming knowledge in C++ and Dart, while P2 only has programming knowledge in C++. In terms of the SDKs that they are familiar with, P1 and P2 selected Java SE, the standard/default API for Java programming.

Table 7.1 shows the results of using Google search engine and *APIRecJ* to search for Java API classes for the three programming questions given. Participants are required to provide three most relevant API classes for each of question.

Based on the findings, the average correctness of Java API classes found by using *APIRecJ* for Q1 is 66.67% and average time taken is 3.5 minutes. However, the average correctness of Java API classes found by using Google search engine for Q1 is 83.33% and average time taken is 2 minutes.

For Q2, the average correctness of Java API classes found by using *APIRecJ* is 50% and average time taken is 1.5 minutes. However, the average correctness of Java API classes found by using Google search engine is 66.67% and average time taken is 3 minutes.

For Q3, the average correctness of Java API classes found by using *APIRecJ* is 50% and average time taken is 2.5 minutes. However, the average correctness of Java API classes found by using Google search engine is 50% and average time taken is 2 minutes.

Table 7.1: Summary of Section 2 Result in Pilot Study

Participants	Programming Question	APIRecJ (Part A)		Google search engine (Part B)	
		Correct APIs	Time Required (min)	Correct APIs	Time Required (min)
P1*	Q1	2	3	3	2
	Q2	1	2	3	3
	Q3	2	2	1	2
P2#	Q1	2	4	2	2
	Q2	2	1	1	3
	Q3	1	3	2	2
Average Correctness (%)	Q1	66.67		83.33	
	Q2	50		66.67	
	Q3	50		50	
Average Time Required (min)	Q1	3.5		2	
	Q2	1.5		3	
	Q3	2.5		2	
	Note: * indicate participant perform Part A followed by Part B # indicate participant perform Part B followed by Part A				

Section 3 (Part B) Result: Both P1 and P2 agreed that it is easy to find the relevant Java API classes by using *APIRecJ* (Question 1), that it is helpful to have the Java API classes recommended as filters in searching for relevant Java API classes (Question 2) and that it is helpful to have similar Stack Overflow questions as they provide suggestions for searching for relevant Java API classes (Question 3).

Both P1 and P2 prefer to use *APIRecJ* instead of Google search engine for searching relevant Java API classes for their programming questions (Question 4) because of more efficient and ease of use (Question 5). P2 gave another reason, which is, "*By using Google, we have to go through each website one-by-one and we might not be able to find the answers that we wanted, however APIRecJ solved this by displaying the list of questions available and also the APIs used which is very clear to users*".

P1 stated a "Yes" when asked whether it is useful to have an API class recommender tool such as *APIRecJ* that helps to find relevant information on API (such as API classes) when doing programming (Question 6) and PI gave the reason of "*It helps to save the hassle of going back and forth looking and searching between different websites for guidance*", Similarly, P2 stated a "Yes" and gave the reason of "*I would be able to know which API can help me perform the job and I can make comparisons since all the options has been displayed. Also, I would be able to check whether I'm using them correctly*."

7.1.3 User Evaluation Study Design

Based on the findings from the pilot study, the answer options of Question 2 ("What is your level of Java programming skill?") of Part A of the questionnaire were improved. The original options were:

a) Junior – Design classes, interfaces and exception handling,

b) Intermediate – Knowledge on algorithm efficiency for Java collections framework and JVM

c) Advanced – Knowledge on multi-threaded programming, concurrency issues, managing life cycle and priority of threads.

The options were improved to better descriptions of the levels of Java programming skill comprising of four options, which are:

a) Beginner – Basic understanding on Java programming, for example, Hello World program

b) Junior – Design and programming using classes, interfaces and exception handling

c) Intermediate – Produce quality program and understand algorithm efficiency for each of the Java collections framework

d) Advanced – Programming using almost all Java framework component including multi-threaded programming, concurrency issues, managing life cycle and priority of threads

7.1.4 User Evaluation Results

Two participants (denoted as P3 and P4) participated in the user evaluation study. Both are first year computer science undergraduate students from the Faculty of Computer Science & Information Technology, University Malaya, who scored an 'A' in the introductory programming subject (WIX1002 Fundamentals of Programming) they took in their undergraduate study.

In terms of the level of Java programming skill, they chose the option of "junior" level that represents that they know about designing and programming using classes, interfaces, and exception handling. In terms of other programming languages, P3 has programming knowledge in three other programming languages, which are C++, Python and Dart, while P4 has programming knowledge in Visual Basic and Dart. Both are familiar with the Java SE SDK but P4 is also familiar with Android SDK, which is a Java API developed specifically for mobile phone operating system.

Table 7.2 shows the results of using Google search engine and *APIRecJ* to search for Java API classes for the three programming questions given. Participants are required to provide three most relevant API classes for each of question.

Based on the findings, the average correctness of Java API classes found by using *APIRecJ* for Q1 is 66.67% and average time taken is 4 minutes. However, the average correctness of Java API classes found by using Google search engine for Q1 is 100% and average time taken is 2 minutes.

For Q2, the average correctness of Java API classes found by using *APIRecJ* is 50% and average time taken is 5 minutes. However, the average correctness of Java API classes found by using Google search engine is 83.33% and average time taken is 2.5 minutes.

For Q3, the average correctness of Java API classes found by using *APIRecJ* is 100% and average time taken is 3 minutes. However, the average correctness of Java API classes found by using Google search engine is 50% and average time taken is 1.5 minutes

Table 7.2: Summary of Section 2 Result in User Evaluation Study

Participants	Programming Question	APIRecJ (Part A)		Google search engine (Part B)	
		Correct APIs	Time Required (min)	Correct APIs	Time Required (min)
P3#	Q1	2	4	3	2
	Q2	2	4	2	2
	Q3	3	3	1	2
P4*	Q1	2	4	3	2
	Q2	1	6	2	3
	Q3	3	3	2	1
Average Correctness (%)	Q1	66.67		100	
	Q2	50		83.33	
	Q3	100		50	
Average Time Required (min)	Q1	4		2	
	Q2	5		2.5	
	Q3	3		1.5	
	Note: * indicate participant perform Part A followed by Part B # indicate participant perform Part B followed by Part A				

Section 3 (Part B) Result: For Questions 1 – 3, participants were asked to rate the features of *APIRecJ* using a 5-point Likert scale (1 - "Strongly Disagree", 2 – "Disagree", 3 – "Neither Agree nor Disagree", 4 – "Agree" and 5 – "Strongly Agree"). Participants were asked to circle their chosen option for each of these Likert scale questions and to state the reason if he or she chose the option of 3 or below.

Question 1 states "It is easy to find the relevant Java API classes by *using APIRecJ. If your response is 3 and below, please state the obstacles of finding the relevant Java API classes." P3 agreed that it is easy to find the relevant Java API classes by using APIRecJ but P4 disagreed* on this and stated "*If I'm forced to only use APIRecJ, it's quite frustrating, like for question 3, it showed result of String to Date even I typed Date to String*.".

Question 2 states “It is helpful to have the Java API classes recommended as filters in searching for relevant Java API classes. If your response is 3 and below, please state why it is not helpful to have the Java API classes as filters in searching for relevant Java API classes.”. P3 agreed that it is helpful to have the Java API classes recommended as filters in searching for relevant Java API classes. However, P4 neither agreed nor disagreed and stated “*For most people, when we want to search for a solution, we don’t know an API for it existed, so showing a long list of Java API without reading its documentation doesn’t help much*”.

Question 3 states “It is helpful to have similar Stack Overflow questions as they provide suggestions for searching for relevant Java API classes. If your response is 3 and below, please state why the similar Stack Overflow questions are not helpful in finding relevant Java API classes.” Both P3 and P4 agreed that it is helpful to have similar Stack Overflow questions as they provide suggestions for searching for relevant Java API classes.

Question 4 states “Which one do you prefer to use for searching relevant Java API classes for your programming questions?” and there are two options given, which are *APIRecJ* and Google search engine. P3 chose *APIRecJ* and P4 chose Google search engine.

Question 5 states “What are your reasons for your answer for the previous question? You can select more than one option” and there are three options given, which are, “More efficient (Less searching effort)”, “Ease of use”, and “More relevant information”. P3 preferred to use *APIRecJ* because it is more efficient and because of ease of use and gave other reason of “*APIRecJ can get all the API class related to my question instantly*”. In contrast, P4 preferred to use Google search engine because of more relevant information.

Question 6 states “Is it useful to have an API class recommender tool such as *APIRecJ* that helps you to find relevant information on API (such as API classes) when you are doing programming? Yes/No. Please state your reason.”. P3 selected “Yes” and gave the reason of “*It is easy to find a list of useful API regarding my question. However, to understand more about on API's function, I will choose to use google and maybe refer more to Java Oracle*”. Similarly, P4 selected “Yes” and gave the reason of “*More option is always better, considering Google doesn’t always return Stack Overflow results, cannot show list of all API that could solve my problem*”.

7.1.5 Discussion of User Evaluation Results

The user evaluation results (Question 1 of Part B of questionnaire) show that *APIRecJ* occasionally recommends less relevant result. This is probably because it requires the query to be in the form of a longer text description in order to get a better result. For example, instead of using “How do I convert Date to String?” as the query, P4 used “Date to String” as the query and *APIRecJ* returned a less relevant results. Another possible reason could be the relevance of the recommended results is dependent on the programming question. This can be seen in the user evaluation results, where Google performed better for Q1 and slightly better for Q2 but *APIRecJ* performed better for Q3.

P3 (Question 6 of Part B of questionnaire) and P4 (Question 2 of Part B of questionnaire) also pointed out that returning a list of Java API classes might not be useful because programmers are frequently unfamiliar with Java API names and require more assistance in terms of API description and usage. This probably could be the reason for more time required to understand Java API when using *APIRecJ*. Therefore, it would be more helpful if *APIRecJ* could include more information such as API descriptions from the Java documentation.

Although the two participants have different preference of *APIRecJ* or Google (Question 4), P4 that preferred Google indicated that it is useful to have an API class recommender tool such as *APIRecJ* that helps to find relevant information on API (such as API classes) when are doing programming, because Google search engine does not always return Stack Overflow results and cannot show a list of all APIs that could solve the problem (Question 6). In addition, *APIRecJ*'s feature of providing the similar Stack Overflow questions received positive feedback (Question 3). This shows that *APIRecJ* is generally useful.

7.2 Chapter Summary

This chapter explains the evaluation of plug-in carried out to measure the usefulness of the developed plug-in. The evaluation is consisting of two major studies, which are pilot study and user evaluation study. Therefore, the evaluation begins with designing a data collection instrument for the pilot study and improved for the user evaluation study. As a result, four participants are recruited for both of the evaluation study, where two participants participated in the pilot study and another two participants participated in the user evaluation study. Participants were recruited to use *APIRecJ* to find relevant Java API classes for three programming questions given and to use the Google search engine for the same purpose. Based on the results gathered from data collection instrument, the developed plug-in can be considered useful for programmers.

CHAPTER 8: CONCLUSION

This chapter presents the conclusion of this research. It includes the answering of the research questions, a revisit of research contributions, threats to validity of the results, and outline of possible future work.

8.1 Answering of Research Questions (RQs)

This section re-states the RQs of this research and summarizes how they were achieved and the answers to these RQs.

RQ1: What are Java programmers' common Java programming problems?

RQ1 was achieved by mining and analyzing duplicate Java questions/posts in SO. Since duplicate questions are the same questions that different programmers repeatedly asked in different contexts, they can be used as surrogates to common questions asked in SO. These common questions are in fact common Java programming problems/topics that Java programmers struggle with.

The level of expertise was determined based on the askers' reputation scores in SO. It was found that the novice group is the top contributor and the expert group contributes significantly lower to duplicate questions. The top-10 duplicate Java questions in SO (Table 4.3) were identified and it was found that the most common problem Java programmers face is understanding and/or fixing errors. However, this is not the case for the expert programmers, as they question more about the reason of some Java programming concepts.

The top-30 Java API classes required by the top-10 duplicate Java questions in SO were also identified (Table 4.4) and the main findings are: some of the most frequent Java API classes required by the top duplicate Java questions of all expertise groups are from *java.lang, java.util, javax.swing, java.text* and *java.io* packages; novice programmers ask duplicate questions related to classes in *java.awt* package but not the experienced and expert programmers; experienced and expert programmers but not the novice asked duplicate questions related to classes used in more advanced topics such query of database.

RQ2: How to design an approach that recommends relevant Java API classes for Java programming questions by mining discussion posts in SO?

To answer RQ2, an approach in recommending Java API classes was proposed and the approach was implemented into the Java API class recommender developed in this research. The approach is built based on the common steps in designing RSSE and adopted some NLP techniques (Doc2Vec word embedding and LDA topic modelling) and a number of heuristic rules for Java API classes extraction.

The proposed approach is developed into a backend server that functions as a Java API classes recommender. A plug-in for Eclipse IDE, *APIRecJ* was developed to serve as the front-end to access the recommender's functionalities. *APIRecJ* is responsible for getting user's query, making service call to backend recommender and displaying the results to the user. Indirectly, the lexical gap between the programmers' natural language queries and the APIs documentation, and programming code, could be reduced by using *APIRecJ* and the Java API class recommender that links the programmers' natural language queries to relevant Java API classes.

RQ3: What is the performance of the approach?

RQ3 was achieved by benchmarking the performance of the proposed approach against the existing studies on Java API classes recommendation that made use of SO's discussion posts. For consistency and comparison, evaluation dataset and evaluation metrics from existing studies were used. The four performance metrics used were Top-10 accuracy, MR@10, MRR@10 and MAP@10. The benchmarking results demonstrate that the proposed approach has improved the existing state-of-the-art baseline result by 3.22% in Top-10 Accuracy, 0.03 in MRR@10, 2.83% in MAP@10 and 0.89% in MR@10.

RQ4: How useful is the plug-in?

To answer RQ4, a user evaluation study preceded by a pilot study was conducted. Participants were recruited to use *APIRecJ* to find relevant Java API classes for three programming questions given and to use Google search engine for the same purpose, and subsequently answer a questionnaire.

The user evaluation results show that *APIRecJ* occasionally recommends less relevant result. This could be due to the relevance of the recommended results is dependent on the programming question. The evaluation results also shows that: 1) it would be more helpful if *APIRecJ* could include more information such as API descriptions from the Java documentation, apart from returning the Java API classes.; 2) it is useful to have an API class recommender tool such as *APIRecJ* that helps to find relevant information on API (such as API classes) when are doing programming, because Google search engine does not always return Stack Overflow results and cannot show a

list of all APIs that could solve the problem; 3) *APIRecJ*'s feature of providing the similar Stack Overflow questions received positive feedback.

8.2 A Revisit of Research Contributions

The contributions of this research are as below:

1. Common Java programming problems encountered by programmers of different levels of expertise, identified from SO, the most popular computer programming related website. In addition, the top Java API classes related to these common Java programming problems were also found. These provide insights on common Java programming problems/topics and Java API classes that Java programmers struggle with. Java educators and learning resources can devote more attention to these areas (for example, understanding and fixing errors) to help learners in picking up the required knowledge and skills.
2. An approach that employs heuristic rules, word embedding and topic modelling techniques in recommending relevant Java API classes for Java programming questions described in natural language. The approach outperforms existing approaches in terms of four performance metrics, by achieving 84.83% in Top-10 Accuracy, 0.58 in MRR@10, 50.68% in MAP@10 and 58.76% in MAP@10. These results demonstrate that the proposed approach has improved the existing state-of-the-art approach by 3.22% in Top-10 Accuracy, 0.03 in MRR@10, 2.83% in MAP@10 and by 0.89% in MR@10.
3. A Java API class recommender that incorporated the proposed approach.
4. A plug-in (*APIRecJ*) for Eclipse IDE that serves as front-end to the Java API class recommender. The use of this plug-in when writing Java programs in

Eclipse IDE allows the programmers to describe their Java programming problems in natural language and search for Java API classes that are relevant to the programming problems and view similar questions that have been asked in SO. All these actions can be performed within the Eclipse IDE without leaving the IDE. The user evaluation of *APIRecJ* shows that it is a useful tool for programmers particularly in answering questions that search for relevant Java API classes.

In summary, the plug-in coupled with the back-end Java API class recommender that incorporates the proposed approach helps to reduce the lexical gap between the programmers' natural language queries and the Java APIs documentation (in particular, the Java API classes names), and the lexical gap between the natural language queries and Java programming codes, and helps to complement official APIs documentation which have poor usability, by applying data mining techniques on Stack Overflow crowd documentation of Java APIs.

8.3 Threats to Validity of Results

There are some threats that could possibly affect the results of this research. First of all, as there is an ongoing high volume of user-generated questions in SO, this could cause different results to be found since the results depend on the SO data retrieved at a particular point of time. As mentioned, information in SO are changing rapidly, using online query tool at different time could produce different result. Alternatively, the data can be retrieved by using data dump. Unfortunately, the limitation is the process to replicate the SO database requires high computing resources which is not available for this research. Secondly, the fourth heuristic rule in Study 1 (Table 4.1) and Study 2 (Table

5.2) validate the extracted Java API classes against a list of actual API classes extracted from the Java API documentation. The list needs to be updated if new classes are added to the Java API when the language evolves.

Lastly, the small number of participants who took part in the user evaluation study causing its results to be non-generalizable. Two were recruited for the pilot study. Two were recruited for the user evaluation study due to time constraint. All the participants are Computer Science undergraduate students from the Faculty of Computer Science and Information Technology, University Malaya, who have completed their first year of study and scored an 'A' in the introductory Java programming subject. Despite the small number, the participants have similar educational background, and good level of skill in Java programming, and, the user evaluation results show that it is generally useful to have a tool such as *APIRecJ* built in this research. It is important to note that the main focus of this research is the proposed approach and the benchmarking of the approach using the four performance metrics shows that it outperforms existing approaches in all the performance metrics. The plug-in evaluated in the user evaluation study is just the front-end to the underlying recommender that incorporates the approach.

8.4 Future Work

Possible future work includes designing a more generalized and robust method in extracting Java API classes rather than using heuristic rules on coding conventions; extend the research to also include recommendation of methods of Java API classes; extend the plug-in to include more information from the Java API documentation (as recommended by the participants in the user evaluation study).

REFERENCES

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C. K., & Schneider, K. A. (2016). *Mining duplicate questions in stack overflow.* Paper presented at the Proceedings of the 13th International Conference on Mining Software Repositories.

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C. K., & Schneider, K. A. (2018). *Classifying stack overflow posts on API issues.* Paper presented at the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER).

Al-Jepoori, M., & Bennett, D. (2018). Understanding of the programming techniques by using a complex case study to teach advanced object-oriented programming.

Allamanis, M., & Sutton, C. (2013). *Why, when, and what: analyzing stack overflow questions by topic, type, and code.* Paper presented at the Proceedings of the 10th Working Conference on Mining Software Repositories.

Apache NetBeans. (2019). Apache NetBeans. Retrieved from https://netbeans.org

Ayyadevara, V. K. (2018). *Pro Machine Learning Algorithms: A Hands-On Approach to Implementing Algorithms in Python and R*: Apress.

Bajracharya, S. K., & Lopes, C. V. (2012). Analyzing and mining a code search engine usage log. *Empirical Software Engineering, 17*(4-5), 424-466.

Białecki, A., Muir, R., Ingersoll, G., & Imagination, L. (2012). *Apache lucene 4.* Paper presented at the SIGIR 2012 workshop on open source information retrieval.

Black, J., Hashimzade, N., & Myles, G. (2009). *A Dictionary of Economics*: Oxford University Press.

Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM, 55*(4), 77-84. doi:10.1145/2133806.2133826

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research, 3*(Jan), 993-1022.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics, 5*, 135-146.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems, 30*(1-7), 107-117.

Chamberlin, D. D., & Boyce, R. F. (1974). *SEQUEL: A structured English query language.* Paper presented at the Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control.

Chen, T.-H., Thomas, S. W., & Hassan, A. E. (2016). A survey on the use of topic models when mining software repositories. *Empirical Software Engineering, 21*(5), 1843-1919.

CodeJava. (2019). CodeJava.net - Java Tutorials, Code. Retrieved from https://www.codejava.net/

Eclipse IDE. (2019). Eclipse IDE. Retrieved from https://www.eclipse.org/eclipseide/

Flask. (2019). Flask | The Pallets Projects. Retrieved from https://palletsprojects.com/p/flask/

GitHub. (2019). Stack Exchange Data Explorer Entity Relationship Diagram. Retrieved from https://github.com/leerssej/SEDESchema

Gosling, J., Joy, B., Steele, G. L., Bracha, G., & Buckley, A. (2014). *The Java Language Specification, Java SE 8 Edition*: Addison-Wesley Professional.

Gosling, J., & McGilton, H. (1995). The Java language environment. *Sun Microsystems Computer Company, 2550*.

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*: Elsevier.

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003). Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin, 35*(1), 153-156.

Huang, Q., Xia, X., Xing, Z., Lo, D., & Wang, X. (2018). *API method recommendation without worrying about the task-API knowledge gap*. Paper presented at the Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018, Montpellier, France.

Ichinco, M., Hnin, W. Y., & Kelleher, C. L. (2017). *Suggesting API Usage to Novice Programmers with the Example Guru*. Paper presented at the Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17, Denver, Colorado, USA.

IntelliJ IDEA. (2019). IntelliJ IDEA Retrieved from https://www.jetbrains.com/idea/

Java2s. (2019). Programming Tutorials and Source Code. Retrieved from http://www.java2s.com/

Java Platform SE 6. (2019). Java Platform, Standard Edition 6 API Specification Retrieved from https://docs.oracle.com/javase/6/docs/api/

Java Platform SE 8. (2019). java.awt (Java Platform SE 8). Retrieved from https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html

Joorabchi, A., English, M., & Mahdi, A. E. (2016). Text mining stackoverflow: An insight into challenges and subject-related difficulties faced by computer science learners. *Journal of Enterprise Information Management, 29*(2), 255-275.

Kavitha, S., & Sindhu, S. (2015). *Comparison of integrated development environment debugging tools : Eclipse vs Netbeans*.

Kettunen, K., Kunttu, T., & Järvelin, K. (2005). To stem or lemmatize a highly inflectional language in a probabilistic IR environment? *Journal of Documentation, 61*(4), 476-496.

KodeJava. (2019). Kode Java | Learn Java by Example. Retrieved from https://kodejava.org/

Le, Q., & Mikolov, T. (2014). *Distributed representations of sentences and documents.* Paper presented at the International conference on machine learning.

Loper, E., & Bird, S. (2002). NLTK: the natural language toolkit. *arXiv preprint cs/0205028*.

Meng, N., Nagy, S., Yao, D., Zhuang, W., & Arango-Argoty, G. (2018). *Secure coding practices in java: Challenges and vulnerabilities.* Paper presented at the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality.* Paper presented at the Advances in neural information processing systems.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM, 38*(11), 39-41.

Mow, I. T. C. (2012). Analyses of student programming errors in Java programming courses. *Journal of Emerging Trends in Computing and Information Sciences, 3*(5), 739-749.

Myers, B. A., & Stylos, J. (2016). Improving API usability. *Communications of the ACM, 59*(6), 62-69.

Nadi, S., Krüger, S., Mezini, M., & Bodden, E. (2016). *Jumping through hoops: Why do Java developers struggle with cryptography APIs?* Paper presented at the Proceedings of the 38th International Conference on Software Engineering.

Nie, L., Jiang, H., Ren, Z., Sun, Z., & Li, X. (2016). Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing, 9*(5), 771-783.

Parnin, C., Treude, C., Grammel, L., & Storey, M.-A. (2012). Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Institute of Technology, Tech. Rep*.

Porteous, I., Newman, D., Ihler, A., Asuncion, A., Smyth, P., & Welling, M. (2008). *Fast collapsed gibbs sampling for latent dirichlet allocation.* Paper presented at the

Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining.

PyInstaller. (2019). PyInstaller. Retrieved from https://www.pyinstaller.org/

Rahman, M. M. (2019a). RACK-Replication-Package/oracle-310.txt Retrieved from https://github.com/masud-technope/RACK-Replication-Package/blob/master/oracle-310.txt

Rahman, M. M. (2019b). Replication package of RACK : Automated Query Reformulation for Code Search using Crowdsourced Knowledge Retrieved from https://github.com/masud-technope/RACK-Replication-Package

Rahman, M. M., & Roy, C. (2018). *Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics.* Paper presented at the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).

Rahman, M. M., Roy, C. K., & Lo, D. (2016, 14-18 March 2016). *RACK: Automatic API Recommendation Using Crowdsourced Knowledge.* Paper presented at the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER).

Rahman, M. M., Roy, C. K., & Lo, D. (2018). Automatic query reformulation for code search using crowdsourced knowledge. *Empirical Software Engineering*, 1-56.

Rehurek, R., & Sojka, P. (2010). *Software framework for topic modelling with large corpora.* Paper presented at the In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.

Rigby, P. C., & Robillard, M. P. (2013). *Discovering essential code elements in informal documentation.* Paper presented at the 2013 35th International Conference on Software Engineering (ICSE).

Robillard, M., Walker, R., & Zimmermann, T. (2009). Recommendation systems for software engineering. *IEEE software, 27*(4), 80-86.

Robillard, M. P. (2009). What Makes APIs Hard to Learn? Answers from Developers. *IEEE software, 26*(6), 27-34. doi:10.1109/ms.2009.193

Robillard, M. P., & Deline, R. (2011). A field study of API learning obstacles. *Empirical Softw. Engg., 16*(6), 703-732. doi:10.1007/s10664-010-9150-8

Santos, A. L., & Myers, B. A. (2017). Design annotations to improve API discoverability. *Journal of Systems and Software, 126*, 17-33. doi:10.1016/j.jss.2016.12.036

Silva, R. F., Paixão, K., & de Almeida Maia, M. (2018). *Duplicate question detection in stack overflow: A reproducibility study.* Paper presented at the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER).

Singhal, A. (2001). Modern information retrieval: A brief overview. *IEEE Data Eng. Bull., 24*(4), 35-43.

Stack Exchange. (2019a). Meta Stack Exchange: Database schema documentation for the public data dump and SEDE. Retrieved from https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede

Stack Exchange. (2019b). Query Stack Overflow - Stack Exchange Data Explorer Retrieved from https://data.stackexchange.com/stackoverflow/query/new

Stack Exchange. (2019c). Stack Exchange. Retrieved from https://stackexchange.com

Stack Exchange. (2019d). Stack Exchange All Sites Traffic. Retrieved from https://stackexchange.com/sites?view=list#traffic

Stack Exchange. (2019e). Stack Exchange Data Dump. Retrieved from https://archive.org/details/stackexchange

Stack Overflow. (2010). Stack Overflow: How many classes are there in Java standard edition? Retrieved from https://stackoverflow.com/questions/3112882/how-many-classes-are-there-in-java-standard-edition

Stack Overflow. (2019a). Developer Survey Results 2019 Retrieved from https://insights.stackoverflow.com/survey/2019#technology

Stack Overflow. (2019b). Stack Overflow. Retrieved from https://stackoverflow.com/

Stack Overflow. (2019c). Stack Overflow Trends. Retrieved from https://insights.stackoverflow.com/trends?tags=java%2Cc%2Cc%2B%2B

Stack Overflow. (2019d). Stack Overflow: How do I ask a good question? Retrieved from https://stackoverflow.com/help/how-to-ask

Stack Overflow. (2019e). Stack Overflow: NullPointerException when Creating an Array of objects [duplicate]. Retrieved from https://stackoverflow.com/questions/1922677/

Stack Overflow. (2019f). Stack Overflow: What does it mean when an answer is "accepted"? Retrieved from https://stackoverflow.com/help/accepted-answer

Stack Overflow. (2019g). Stack Overflow: What is a NullPointerException, and how do I fix it? Retrieved from https://stackoverflow.com/questions/218384/

Stack Overflow. (2019h). Stack Overflow: What is reputation? How do I earn (and lose) it? Retrieved from https://stackoverflow.com/help/whats-reputation

Stack Overflow. (2019i). Stack Overflow: What topics can I ask about here? Retrieved from https://stackoverflow.com/help/on-topic

Stack Overflow. (2019j). Stack Overflow: Why are some questions marked as duplicate? Retrieved from https://stackoverflow.com/help/duplicates

Subramanian, S., & Holmes, R. (2013). *Making sense of online code snippets.* Paper presented at the Proceedings of the 10th Working Conference on Mining Software Repositories.

TIOBE. (2019). TIOBE Index for November 2019. Retrieved from https://www.tiobe.com/tiobe-index/

Treude, C., Barzilay, O., & Storey, M.-A. (2011). *How do programmers ask and answer questions on the web?: Nier track.* Paper presented at the Software Engineering (ICSE), 2011 33rd International Conference on.

Treude, C., Robillard, M. P., & Dagenais, B. (2015). Extracting Development Tasks to Navigate Software Documentation. *IEEE Transactions on Software Engineering, 41*(6), 565-581. doi:10.1109/tse.2014.2387172

Wikipedia. (2019). Web Framework - Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Web_framework

Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., & Xing, Z. (2017). What do developers search for on the web? *Empirical Software Engineering, 22*(6), 3149-3185.

Xia, X., Lo, D., Correa, D., Sureka, A., & Shihab, E. (2016). *It takes two to tango: Deleted stack overflow question prediction with text and meta features.* Paper presented at the 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC).

Ye, X., Shen, H., Ma, X., Bunescu, R., & Liu, C. (2016). *From word embeddings to document similarities for improved information retrieval in software engineering*. Paper presented at the Proceedings of the 38th International Conference on Software Engineering - ICSE '16, Austin, Texas.

Zamanirad, S., Benatallah, B., Barukh, M. C., Casati, F., & Rodriguez, C. (2017). *Programming bots by synthesizing natural language expressions into API invocations*. Paper presented at the Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, Urbana-Champaign, IL, USA.

Zhang, Y., Lo, D., Xia, X., & Sun, J.-L. (2015). Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology, 30*(5), 981-997.

Zhang, Y., Rahman, M. M., Braylan, A., Dang, B., Chang, H.-L., Kim, H., . . . Khetan, V. (2016). Neural information retrieval: A literature review. *arXiv preprint arXiv:1611.06792*.

Zhu, Z., Hua, C., Zou, Y., Xie, B., & Zhao, J. (2017). *Automatically Generating Task-Oriented API Learning Guide*. Paper presented at the Proceedings of the 9th Asia-Pacific Symposium on Internetware - Internetware'17, Shanghai, China.

Zibran, M. F., Eishita, F. Z., & Roy, C. K. (2011). *Useful, but usable? factors affecting the usability of APIs.* Paper presented at the 2011 18th Working Conference on Reverse Engineering.