ANALYZING SOURCE CODE IDENTIFIERS FOR CODE REUSE

Ponnampalam Pirapuraj

158021B

Thesis submitted in partial fulfilment of the requirements for the

Degree of Master of Science (Part Time) (By Research)

LIBRARY
UNIVERSITY OF MORATUWA, SRI LANKA
MORATUWA

Department of Computer Science & Engineering

University of Moratuwa

Sri Lanka

TH3497 + CD ROM

October 2017

004 "17"
004 (043)

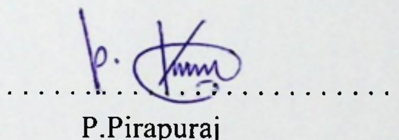
TH3497

Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement the material previously submitted for a Degree or Diploma in the other University or institute of higher learning and to the best of my knowledge and belief it does not contain the material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Candidate



..11-10-2017....... Date

.... P.Pirapuraj

The above candidate has carried out research for the Masters thesis under my supervision.

Supervisor

11.10.2017.............. Date

UOM Verified Signature

Dr. Indika Perera

Acknowledgements

I would like to take this opportunity to acknowledge and extend my heartfelt gratitude to my supervisor Dr. Indika Perera for his long support and guidance throughout this project work, He has given me many stimulating suggestions and his valuable comments have helped me to solve most of the problems during the project development. Many thanks goes out to our MSc research degree coordinator Dr. A. Shehan Perera for his dedication and valuable comments and suggestion as a progress review panel board member. Thanks to another progress review panel board member Dr. Lochandaka Ranathunga, for his valuable suggestion and comments. Thanks to all the lecturers at the Faculty of Computer Science and Engineering, University of Moratuwa, for their valuable advice. Also thanks to my parents, my wife, and brothers, and friends to give support and motivation.

Abstract

Today a massive amount of source code is available on the Internet and open to serve as a means for code reuse. Developers can reduce the time cost and resource cost by reusing these external open source code in their own projects. Even though a number of Code Search Engines (CSE) are available, finding the most relevant source code is often challenging. In this research, we proposed a framework that can be used to overcome the problem faced by developers in code searching and reusing. The framework starts with the software architecture design in XML format (Class Diagram), extracts information from the XML file, and then based on the extracted information, fetches relevant projects using three types of crawler from GitHub, SourceForge, and GoogleCode. We will have a huge amount of projects by downloading process using the crawlers and need to find most relevant projects among them.

In this research, we particularly focus on projects developed using Java language. Each project will have a number of .java files, and all files will be represented as Abstract Syntax Trees (AST) to extract identifiers (class names, method names, and attributes name) and comments from the .java files. Then, on one hand, we will have the identifiers which are extracted from the XML file (Class diagram), and the other hand the identifiers and the action words (verbs) extracted from downloaded projects. Action words are extracted from comments using Part of Speech technique (POS). These two group of identifiers need to be analyzed for matching, if the identifiers are matched, an amount of marks will be given to these identifiers, likewise marks will be added together and then if the total marks is greater than 50%, the .java file belongs to these identifier will be suggested as relevant code. Otherwise, synonyms of the identifiers will be discovered using WordNet, and the matching process will be repeated for the synonyms. For the composite identifiers, camel case splitter is used to separate these words. If the programmers do not follow camel case naming convention, N-gram technique is used to separate these word. The Stanford Spellchecker is used to identify abbreviated words. Evaluation of our developed framework resulted in 95.25% of average accuracy of four subsystem [project downloader (100%), identifier analyzer (94%), word finder (87%), and comments analyzer (100%)] accuracy.

Keywords— Software Architecture, WordNet, N-gram technique, Part of Speech Tagging, Camel Splitter, and Abstract Syntax Tree.

Table of Contents

Declaration i
Acknowledgements ii
Abstract iii
List of Figures vii
List of Tables ix
List of Abbreviations x

1 Introduction 1
1.1 Source code Reuse 1
1.2 Code Search Engine (CSE) 4
1.3 Problem Statement 5
1.4 Research Question and Objectives 7
1.5 Contribution 8
1.6 Challenges 9
1.7 Organization of the Thesis 10

2 Literature Review 11
2.1 Code searching 11
2.2 GitHub API Integration 19
2.2.1 GitHub API 19
2.2.2 Search API and repository search 23
2.2.3 Crawler, and Java JSOUP API 27
2.3 NLP techniques, APIs and tools 32
2.3.1 N-gram Technique 33
2.3.2 Dynamic Time Warping 34
2.3.3 Stanford SpellChecker 36
2.3.4 WordNet 37
2.3.5 Stanford POS tagger 40

2.4 Identifiers and comments analyzing 42
2.5 Analyzing our research with related researches 47

3 Research methodology and Proposed Framework Architecture 50
3.1 Research Methodology 50
3.1 XML Parser 52
3.2 GitHub API Integration 54
3.2.1 Java project names dumber 54
3.2.2 Java class names dumber 55
3.3 Crawlers and Decompressor 56
3.4 Abstract Syntax Tree 57
3.5 Identifier Splitter 61
3.6 Spell Checker and Word Finder 63
3.6.1 Spell checker for Good Identifiers 64
3.6.2 Word Finder and Spell checker for Bad Identifiers 65
3.7 Action word extractor for comments 67
3.8 Matching and marks giving 69
3.9 Summary of all module and process 70

4 Implementation 74
4.1 Implementation of XML Parser 75
4.2 Integration of GitHub API 78
4.2.1 Implementation of Java project names dumber 79
4.2.1 Implementation of Java class names dumber 82
4.3 Implementation of Crawler and Decompressor 84
4.4 Abstract Syntax Tree and Identifier Splitter 87
4.4.1 Implementation of Abstract Syntax Tree 87
4.4.2 Implementation of Identifier Splitter 89
4.5 Implementation of Spell Checker and Word Finder 90
4.5.1 Implementation of Spell Checker for Good Identifiers 91

4.5.2 Spell Checker and Word Finder for Bad Identifiers 92
4.6 Implementation of Matching and Rating 93

5 Evaluation 96
5.1 Performance of crawlers 97
5.2 Performance of Spell Checker and word finder 98
5.2.1 Performance of Spell Checker for Good Identifiers 100
5.2.2 Performance of Spell Checker for Good Identifiers 103
5.3 Performance of Extraction of action verb from comments 104

6 Conclusions 106
6.1 Conclusion 107
6.2 Limitation and Future Work 110
6.2.1 Limitation of our work 110
6.2.2 Future work 111

Reference 112

List of Figures

Figure 1.1: Overview of our System 6

Figure 2.1: TDCS Process 12

Figure 2.2: CodeGenie Search View 15

Figure 2.3: Architecture of the JBender Prototype 18

Figure 2.4: Overview of GitHub page 20

Figure 2.5: Repository in GitHub 20

Figure 2.6: GitHub API implemented System overview 22

Figure 2.7: URL to access all java repository names and all .java file names in GitHub and its details 24

Figure 2.8: Client errors 1 26

Figure 2.9: Client errors 2 26

Figure 2.10: Client errors 3 26

Figure 2.11: The basic crawler architecture 29

Figure 2.12: Distributing the basic crawl architecture. 31

Figure 2.13: Overview of JSOUP package 32

Figure 2.14: Time alignment of two time-dependent sequences. 34

Figure 2.15: Composite identifiers recognition 34

Figure 2.16: Online Version of Stanford SpellChecker 37

Figure 2.17: A WordNet Noun Tree 40

Figure 2.18: Taggest list and example 41

Figure 3.1: Overview of our System 52

Figure 3.2: Web graph joined by a link 57

Figure 3.3: Abstract Syntax Tree example 61

Figure 3.4: Overview of our proposed algorithm 67

Figure 4.1: Example of XML Parser 77

Figure 5.1: Result of downloading relevant projects 99

Figure 5.2: Result of Connected and Single Words in Identifiers 102

Figure 6.1: Accuracy of all Modules 109

List of Tables

Table 2.1: The parameters used in Search API Query....................24

Table 2.2: The search qualifiers used in query....................25

Table 5.1: Result of downloading relevant projects....................98

Table 5.2: Sample result of Identifiers....................101

Table 5.3: Sample result of splitting and identifying the real words....................105

Table 5.4: Sample result of rating the project....................105

List of Abbreviations

CSE	Code Search Engine
AST	Abstract Syntax Tree
DTW	Dynamic Time Warping
API	Application Programming Interface
SVN	Subversion (Source code Management)
XML	Extensible Markup Language
DOM	Document Object Model
NLP	Natural Language Processing
TDCS	Test Driven code Searching
SME	Small and Medium Enterprise
AQE	Automatic Query Expansion
SVG	Scalable Vector Graphics
SSI	Structure Semantic Indexing
JSON	JavaScript Object Notation
MIS	Method Invocation Sequence
AST	Abstract Syntax Tree

Chapter 1

Introduction

Source code reuse positively increases the productivity of the development team, and overall quality of the resulting software. Currently, the amount of open source code available on the Internet is enormous. For example, Sourceforge.net [1], the world's most popular website for open Source software management, hosts about 179,518 projects and has over two million registered users and a large number of anonymous users. As a result of the enormous amounts of open source code available on the Internet, several code search engines (CSE) such as Google code Search [2], Krugle [3], Koders [4], and Codase [5] were developed to efficiently search for relevant code examples. However, obtaining the most relevant source code remains to be challenging. This chapter gives an introduction to the research while describing the problem domain and the developer's challenges when trying code reuse.

This chapter gives an introduction to the research background while describing the problem domain and the challenge faced by the developers when they are trying to reuse the source code. Section 1.1describes the problems and challenges faced by developers. Section 1.2 describe the process of code search and its challenges in small software companies. Section 1.3 describes a possible solution for the challenges faced by developers. Finally section 1.4 describes the contribution of this research.

1.1 Source code Reuse

Today, the competition among the Software companies are very high, because of big number of companies, huge amount of intelligent staff, and the latest technologies. When software architecture enter to development process, the developer cannot start from scratch, they need to import a sample source code and the libraries in their

projects. Open source code available on the Internet has become a common platform for sharing source code. Programmers often prefer to reuse the design of code examples or adapt code examples from existing open source projects rather than discovering usage patterns by digging into documents. Currently, the amount of open source code available on the Internet is enormous. As we denoted earlier, Sourceforge.net [1], the world's most popular website for open source software development, hosts about 179,518 projects and has over two million registered users and a large number of anonymous users. GitHub [6], hosts about more than 5 million open Source projects. As well as GoogleCode [2] also having a massive amount of projects. Research in software engineering has shown that software reuse positively affects the competitiveness of an organization: the productivity of the development team is increased, the time-to-market is reduced, and the overall quality of the resulting software is improved [7]. Today's code repositories on the Internet provide a large number of reusable software libraries with a variety of functionality. The study results indicate that third-party code reuse plays a central role in modern software development and that reuse of software libraries is the predominant form of reuse. It shows that most of today's software systems consist to a considerable fraction of code reused from software libraries [7]. Therefore, code reuse is very important task in software development.

Software reuse offers a considerable advantage in time and monetary costs. The advantage of code reuse in terms of time cost can be analyzed as follows. If a developer writes a program from scratch, he needs to test every piece of written code, identify errors and correct them. If the targeted task is not achieved, he would need to repeat the entire process from the beginning. Additionally the correctness of the code depends entirely on the developer. When reusing related source code the developer does not have to face the above described scenario. The advantage of code reuse in terms of monetary cost can be analyzed as follows. If a developer writes a program from scratch, he needs extra days or month, in which Situation the company has to pay additional salary to staff, but the company cannot get extra money from a client, so

company faces money lose in particular projects. As well as the company needs to spend an amount of money to physical resource, for an example if they use software reuse process, consider a particular project would be finished in 30 days, if they do not follow software reuse process, it would take 60 days to finish, in this situation the company needs to pay following additional bills, internet connection bill, current bill, food bill, and water bill as well as if the building is rent, the company has to pay additionally one month rent. So software reuse reduces these money costs.

In addition to cost reduction described in the above paragraph, software reuse offers benefits on following aspects. In terms of the quality of the software can be analyzed as follows. Most of the time the developers are uploading the finalized project into the GitHub [6] or SourceForge [1], they are uploading the projects after checking all errors, but few errors may be included. If they use the projects in their own project, the quality of their project will be increased. In terms of developer's knowledge will be analyzed as follows. When the developer uses external projects in their own projects, he needs to go through the Source coding, while he will study the different coding style, different naming convention style, special programming technique; for an example, for the same task, different developers will write the code differently as well as number of lines of code will be different, they try to reduce the number of lines of code, so when they go through the code, they will learn the coding technique, so their knowledge will be increased. When we consider the company market will be analyzed as follows. When the developer does the source code reuse task, he will finish the projects within short period with very high-efficiency product, so the client of the company will be very happy, and trust on the company will be increased, so again the clients will not go to the other place for the needs, directly they will give all Software problem to the company. As well as when other clients here the company product and short period process, they also will contact the company to their work, in this way their market will be increased.

1.2 Code Search Engine (CSE)

Code Search Engines (CSE) is a software system that is designed to search source code on the World Wide Web. Google code search [2] (is a free beta product from Google which debuted in Google Labs on October 5, 2006, allowing web users to search for open-source code on the Internet. Features included the ability to search using operators, namely language:, package:, license: and file)[8], Ohloh [9]: (Ohloh code says it is one of the largest and more comprehensive code search engines with more than 10+ billion lines of code indexed and updated FOSS software directories. Ohloh [9] is the upgraded face of Koders.com and is also freely available and freely editable by its community. It indexes all text files for search and has syntax highlighting support for 43 programming languages.)[10], Krugle [3] (is an open Source search portal which taps into open Source search repositories like Apache, JavaDocs, and SourceForge [1] among others. You can search for code in C++, Java, Perl, Python, SQL, Ruby, XML, HTML etc. It is powered by OpenSearch. Krugle [3] also has an advanced search feature that can help you narrow down to the right APIs, libraries, sample code or documentation. From the results page, you can browse to the project developed with the code.)[10].

Likewise, few other code search engines, such as Koders [4], and Codase [5] are developed to search for relevant code examples (i.e., Source files containing a search term). These CSEs accept queries such as the names of classes or methods of Application Programming Interfaces (API) as a keyword which needs to represent the whole meaning of the context and search in CVS or SVN repositories of available open Source projects on hosts such as Sourceforge [1] and GitHub [6]. Today's code search engines, for example, Google, Koders [4], and Krugle [3], offer only a keyword-based search and file-level retrieval, and generally, have limited utility in looking for appropriate code fragments for a particular application. The second problem is that the identified code rarely meets the user's requirements. The functions with similar names do not guarantee that the functionality, computation or the output is similar or has been

implemented correctly. It might also pose security or privacy risks, and can be too complex to be understood or reused. The efficiency and design of implementation can also be problematic. The methods might take slightly different parameters or return slightly different values. Moreover, determining these differences is up to the programmer and requires understanding the retrieved code, a difficult task, especially if there are hundreds of poorly-documented search results.

The third problem is that even when code can be found that meets one's requirements, that code still has to be modified and adapted by the programmer. Overall, programmers often feel that the effort required here is more than the effort required in writing the code in the first place.

Developers are hesitant to use code Search Engine to get source code because of an above mentioned problem with CSE. When code searching process, developers are depending on the expert in a particular context or, they are developing from scratch, and wasting time and money.

1.3 Problem Statement

In this research, our goal is to propose a software solution to developers who are facing challenges in code reuse process. Our research is mainly focused only on small companies. In software companies, the typical software development process is as follows. First, the software architect designs the software architecture of the system, and visualize it using a class diagram, an object diagram. Then the design will be sent to the developer team. When the software architecture enters to the implementation phase, the developer cannot start the coding from scratch, they need to include few sample code or libraries in their own project. For this purpose, if the companies are very big in economical level or human resource level, they will maintain an expert in the particular area to advice to all staff [7].

Experienced workers are very important to companies, in the sense that someone with experience has higher productivity [7]. When an experienced engineer or other qualified employee leaves a company, it takes with her/him a corpus of knowledge, which from the company point of view is a knowledge loss. In a high competitive market environment, where companies strive to be alive, this problem is crucial. When the developers search for the source code on the internet, the results they get includes a mix of relevant and irrelevant results. The problem is that they need to identify the relevant project in the pool of projects which is much simpler for an experienced developer as they have built up a sense of the domain and software used through time [28]. Therefore an experienced developer can make a better judgment on the most appropriate code or library than a junior developer.

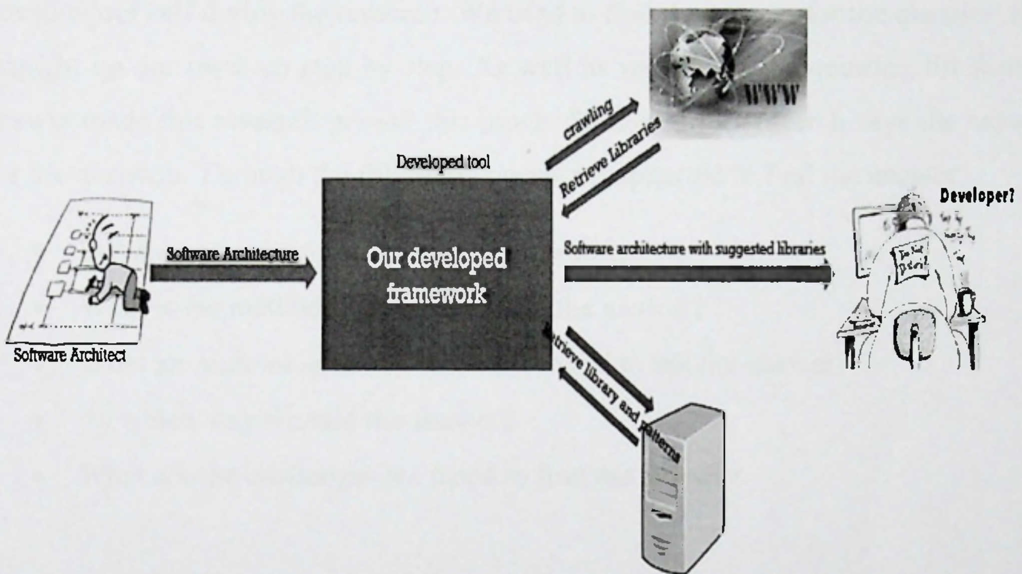


Figure 1.1: Overview of our System [67].

If they have the automated tool to do these expert's work itself, the challenges which were discussed earlier will be vanished. So our research goal is, when a software architecture enters to the development process in XML file format, automatically extract specific information and download related Source code from the internet, and

then run an analyzing and matching process to select the most relevant source code. So we ensure the impact of this research will be higher on all small and big software industry even though we focus only on small and medium companies.

1.4 Research Question and Objectives

Research question

How are the software developers getting suitable sample code and libraries in the development process?

Above mentioned question motivated us to start this research. Frequently we asked the question our self during the research. We tried to find the answer for the question that brought up our research step by step. As well as we delved the question till further answer made this research growth this much. So the whole research says the answer for the question. Through the following question supported to find the answer;

- Is there any existing answer for the question?
- What is the methodology used to find the answer?
- What are technologies and tools were used to tell the answer?
- In which way we said the answer?
- What are the challenges we faced to find the answer?

Objectives

Our goal is to develop a tool to facilitate the software engineers' challenges by automatically optimizing and suggesting some libraries either from their own company database or, from Internet by crawling, and apply appropriate design patterns. To accomplish this goal, we planned the following objectives.

The research is to:

- Analyze the challenges, which includes suggesting libraries and design pattern for a software architecture.
- Identify the challenges, which are faced by developers, when a software architecture get developed.
- Develop a tool to solve the challenges analyzed in this research.
- Evaluate the performance of the tool with some real software architecture.

1.5 Contribution

To achieve the main goals of this research, we have developed an algorithm and a framework to overcome above discussed problems. Our main contribution are as follows:

- Extracting information from Software architecture (class diagram) which is in a XML file format. For this process we used a XML parser (DOM parser).

- Download related project from the internet using the information extracted from the XML file. For this process we developed three types of crawler (GitHub crawler, SourceForge crawler, and GoogleCode crawler) using Java JSOUP API.

- Find all .java files in the downloaded projects and parse these files in Abstract Syntax Tree.

- Extract all identifiers (method names, class names, and attribute names) and leading comments for methods.

- Extracting real words from the identifiers using our algorithm as well as identifying action verb from comments.

- Matching these words with the information from class diagram, rating them and select the most relevant java class depend on the rating to be suggested to developer.

Our experimental evaluation shows that, we can automate the process, the developer does in the implementation phase, and that the automation impacts on the costs which is discussed in Chapter 1, 1.1.

1.6 Challenges

In this research, we tried to automate all processes without human interaction, so we faced lot of challenges. Few of the challenges are listed below,

- When we download the project, our developed framework will download all related projects using keyword extracted from the class diagram. Identifying most relevant project from such a large amount of downloaded project is difficult.

- Identifiers are selected by developers based on their personal preference. Therefor the identifiers can be abbreviated the words or two or more words connected to make concatenate identifiers. In such situations, identifying the original meaning from the identifiers are very difficult.

- The developers do not follow the naming convention when they connect two or more words, such as Camel Case naming convention or the explicit separator (_, /, -, etc). Finding real words from these identifiers is difficult.

- Sometime they shrink the words in very bad manner and connect without the naming convention. Finding real words from these types of identifiers is difficult too.

1.7 Organization of the Thesis

The reminder of this thesis is organize as follows. Chapter 2 presents the background of this research includes types of code searching, and types of code Search Engine, GitHub API, Crawler, and Java JSOUP API, NLP techniques, APIs and tools used in this research, Identifiers and comments analyzing, and related researches.

Chapter 3 describes research methodology, the purpose of each module in our developed framework and our developed algorithm.

Chapter 4 describes the implementation of each module that we have implemented in our framework.

Chapter 5 presents evaluation and result of our proposed approach.

Chapter 6 summarizes our work and suggest future works.

Chapter 2

Literature Review

One of the first thing is a programmer should do when writing new code is to find existing, working code with the same functionality, and reuse as much of that code as possible. With a large amount of open source code available and the fact that most applications are not completely novel, one could imagine that a significant amount of the code that is being written today has been written before in few software application, and much of it is available in an open-source repository. This clearly demonstrate the need for code search. The emphasis here is on enabling reuse, avoiding writing what has been written before, making effective use of open Source software, speeding up development, and producing higher quality software systems. But the challenge is identifying the most relevant sample code from a huge amount of code.

This chapter describes an overview of the problem context through the literature of current state of the art; while discussing every related work with our research work, and how our work differs from their work. More well as some think is better we used than other related work. This chapter includes five section, Section 2.1 includes types of code searching and types of code search engine. Section 2.2 includes GitHub API, Crawler, and Java JSOUP API. Section 2.3 describes the NLP techniques, APIs, and tools used in this research. Section 2.4 describes Identifiers and comments analyzing. Finally in section 2.5, we analyze our research with related work.

2.1 Code searching

Millions of lines of code are becoming available online. In many cases, this code samples are carefully designed, implemented, tested and therefore represents readiness

UNIVERSITY OF MO
LIBRARY
SRI LANKA

for reusability. Lately, more and more companies, especially Small and Medium Enterprises (SMEs), are reusing open source code to develop their own software. Source code repositories such as SourceForge [1], GoogleCode [2] etc., serve as component pools providing plenty of alternatives [11]. An Eclipse plugin (CodeGenie) which is based on Test-driven code search (TDCS) [12] method for source code searching and reusing from open source project on the internet. Test driven development has a test case which includes inputs and expected output. Before implementing the problem we need to write the test program (Using Junit framework in java) which includes input and desired output, when running the code we will get an error message. Then implement the actual program, and run the test code, this time we will get pass message. So, the particular input results in the expected output. This approach was used by them in code search to get the expected output. For example, if we need to search a function to convert Roman number to Arabic number (String roman (int arabic)), Roman, and Arabic will be used as keywords to search and then the result will be matched with method signature. This approach is called test-case queries.

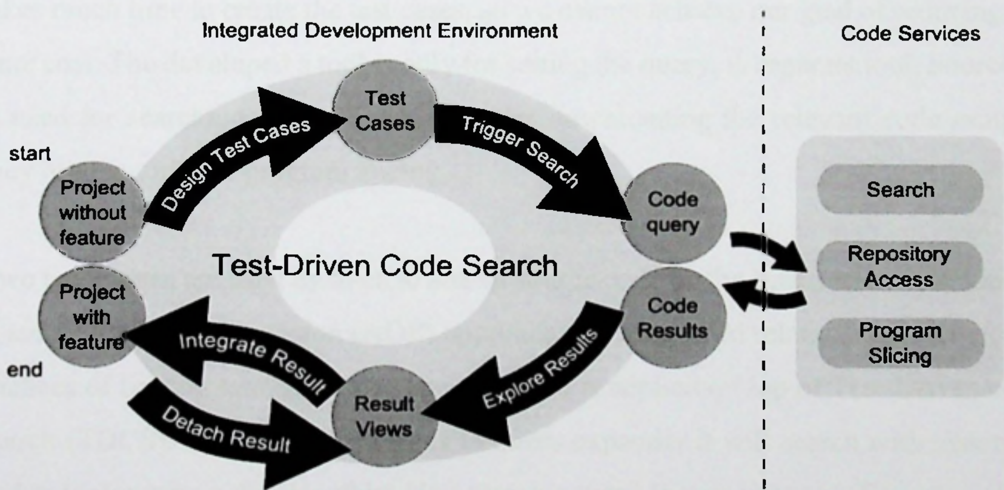


Figure 2.1: TDCS Process [12].

Once the desired code is found in a code search task, it can often be difficult to manually extract what is necessary, to integrate in a local context. In order to address this issue, the automated program slicing used by them. For searching phase they used Sourcerer [12] to search the code. CodeGenie [12] sends the query to Sourcerer which, in turn, returns the code results. The keywords are initially formed by the terms coming from the method name and the class name. Sourcerer is an infrastructure for large-scale analysis and indexing of source code Sourcerer crawls the Internet looking for source code from various sources such as open source code repositories, public websites and version control systems. The source code obtained from the Internet is analyzed, parsed, and stored in the system in various forms. After developing the plugin, they gave their application to 34 senior Computer Science students to evaluate the performance of CodeGenie against a well-known code search engine [12].

Drawbacks of CodeGenie are, we need to write the test program before implementing the problem. First, we need to know the result of the process, because the source code will be downloaded if and only if the expected output is satisfied for every input. It takes much time to create the test cases, so we cannot achieve our goal of reducing the time cost. The developed a tool is only for setting the query. A separate tool, Sourcerer is used for searching source code, and after downloading the relevant code sample, they used automated program slicing.

Two techniques are used by them to search source code on the Internet. First technique is automatic query expansion (AQE) approach that uses word relations to increase the chances of finding relevant code. The approach is applied on top of Test-Driven code search (TDCS). When a query comes to query expander it will search with synonym and antonym using the WordNet [13] (using natural language term). For example if the query had the term getName the generated terms will be 'get' and 'name'; zipFile would generate the terms zip and file. Code candidates must contain these keywords in their method names in order to be matched. The second technique is interface-driven

code search with the test-driven code search. In the interface-driven code search, when the code candidates possess an interface similar to the one designed for the desired function they selected for test-driven code search, where test cases are run against matched candidates to verify their suitability in the local context.

CodeGenie is used to develop this approach, it relies on Sourcerer, a source code infrastructure to support TDCS. The current implementation of Sourcerer receives queries to execute the interface-driven searches. First of all, the users have to develop JUnit test cases for the desired function to search in CodeGenie. Then the tool extracts the interface of the desired function's entry point method and formulates queries that contain three parts:

(1) Return type of the method.

(2) Keywords present in the entry point method name.

(3) Parameter types of the method [14].

In this research also, TDCS used by them, so we faced the same problem as we discussed earlier.

Recent source code search techniques such as keyword-based search, information about program structure, and the test-driven code search are used by the programmer when they search a source code from open source code repository. Cristina V. Lopes, Donald Bren, et al. [14], said, that in terms of keyword search technique the developer faces following challenges. The first challenge is the difficulty in finding the appropriate keywords. But in our case, we extract the keywords from software architecture. Another problem is that keyword-search typically yield many unevaluated results. Therefore the programmers have to read each instance of returned source code, attempt to understand what it does, and then determine if it meets their requirements. But we analyzed the identifier from each Java files to meet the requirements. In the second approach of their research, the program structure used by

them such as, method signature and loop structure. Its applicability is relatively limited. And when they search the program structure may be irrelevant to the search.

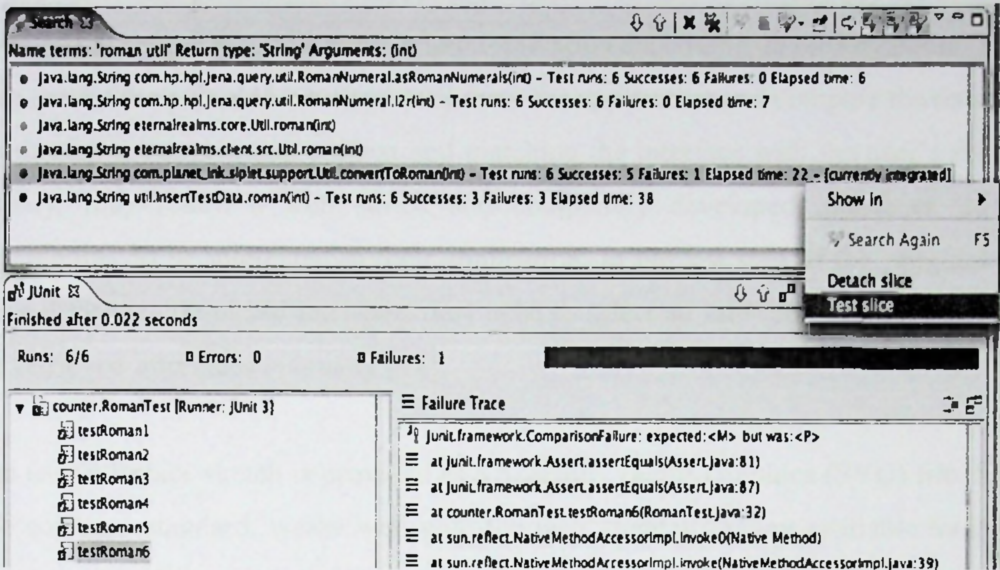


Figure 2.2: CodeGenie Search View [12].

In the test-case approach, test-cases are difficult to define when the solution is not well defined, and the test-cases can be difficult to write. In many cases, the amount of code required for testing can exceed the amount of code being retrieved. Semantics-based search is a more effective method for searching source code. The programmer need to state the information such that, what they want the identified code to do functionally, where it has to fit in, and what constraints (e.g. performance, error handling, security, privacy, synchronization) they want to impose on it. The tool S[6] proposed by Steven P. Reiss and et al. [15], enables few transformation in search such as converting a class-based implementation into a method-based one (or vice versa), generalizing or specializing parameter and return types, changing the way errors are reported, and adding or removing logging or debugging statements.

Steven P. Reiss and et al. [15], used code search for searching user interface from the open Source repository. They start with a simple sketch of the desired interface along with a set of keywords describing the application context. Then they use the existing code search engine to search appropriate application depending on the keywords. Then they extract the runnable interface code from the application and compare the returned result with the application context and matching the interface with the user's sketch. Finally, they return a well suited and completely developed interfaces to the programmer to be interacted in their application. In the last step, if the programmer has a different completed interface, they need to select an interface by going through the retrieved interfaces manually [15].

The user interface sketch is provided as a Scalable Vector Graphics (SVG) file. SVG is a common standard, works well with the web, there are many available tools for creating and editing such diagrams. Apache provides a suite of Java-based tools for SVG. Steven P. Reiss and et al. [15], used a modified version of the S^6 search engine to search user interfaces on the Internet. They transform the user's sketch into a hierarchical component's description for the search. The description includes the components that should be in the user interface and the relationships among the components. Then the tool is evaluated by providing an address book example, where, they take test on different keywords, different search engines, and measure the amount of time for search and the accuracy of given the results [15]. In this solution, the user needs to sketch the interface first, which will take considerable amount of time.

Adrian Kuhn, Florian S. Gysin et al. [16], presents a prototype named JBender that increases the relevance of code search result with trust-ability information. With keyword search, the developer cannot get a trustable project from open Source repository among a huge amount of open source project. In order to get the trustable projects the trustability metrics for code search results that use collaborative filtering of both user votes and cross-project activity of developers were used. JBender creates

a searchable index over the code base and provides a code search over it. Its novelty, however, lies in the underlying metadata which is linked to the projects in the searchable code base upon finding results from the latter JBender can supply the meta-information stored for the result's originating project. To collect meta-information the Ohloh [9] project used by them which is a social networking platform for open source software projects where additional information regarding the projects can be specified [10].

Ohloh [9] provides user-contributed information on both open Source projects and their developers. Normally Ohloh [9] gives Description of original project, project homepage, rating of the project, list of current repositories (type, URL, last time of update, etc.), licenses of files in the project (exact type of license, number of files), employed programming languages (percentage of total, lines of code, comment ratio, etc.), the project's users and developers who worked on the project (experience, commits per project, etc.). Ohloh [9] website provides its own measurement of developer "karma", called kudo-rank. Kudo-ranks are based on a mix of user votes for projects and of user votes for developers, called kudos. Then they use that information they calculate trustability metrics. A project which gets higher trustability metrics value is a more trustable project [16].

This research provides a solution to search trustable projects, but to improve actual relevancy of results we need to analyze the projects deeper at code level.

Sushil K Bajracharya et al. [17], states that keywords based information retrieval is not 100% suitable for code search as sometimes it will fail to perform well in retrieving API usage examples from code repositories. Instead of keywords search, they proposed Structural Semantic Indexing (SSI) [17], a technique to associate words with the Source code entities based on similarities of the API usage. The classes and methods, which have similar functionalities will use the same API, this is semantically related. A retrieval system implemented by them based on the Sourcerer. All the source codes will be indexed and imported into SourcererDB. When we search for a source

code, a retrieval scheme takes a keyword query and returns a ranked list of code entities as a search result. This ranked list of entities is called hits and each entry in the list is called a hit. The search tool will extract a related code snippet for each hit using this information. From the extracted code snippet, assessing the relevancy of each code snippet, and finally selecting a set of metrics to compare the effectiveness of the retrieval schemes. The evaluation methodology for the proposed solution consists of building a corpus to test the retrieval schemes, creating a set of candidate queries, executing the queries using all retrieval schemes to generate code snippets, assessing the relevancy of each code snippet, and finally selecting a set of metrics to compare the effectiveness of the retrieval schemes. For the evaluation, they have used features, such as corpus, ranked results, relevancy judgment, and performance metrics. Their goal with SSI was to be able to build an effective code retrieval scheme that uses no documents other than source code. Large source code repositories such as Sourceforge [1] and GitHub [6] serve both as motivation and target of their approach [17]. But our developed framework can make the keyword search 100% suitable for code search due to analysis on the code level.

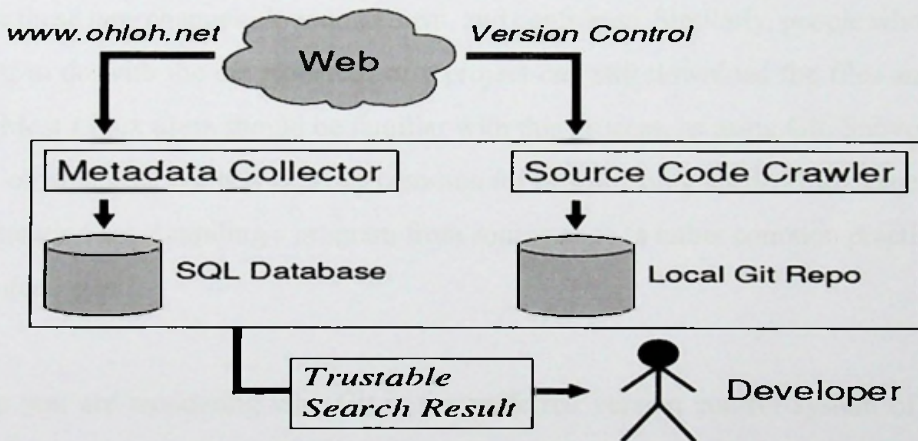


Figure 2.3: Architecture of the JBender Prototype [16].

2.2 GitHub API Integration

In this research, we considered three types of source code repositories such as Sourceforge [1], GoogleCode [2], and GitHub [6]. We develop the same type of crawler for Sourceforge [1] and GoogleCode [2], but for GitHub [6] we have developed a different implementation with GitHub API.

2.2.1 GitHub API

Git is a "version control system," its mean, when developers are creating something (an application or the document), they are making constant changes to the code and releasing new versions, up to and after the first official (non-beta) release. Version control systems keep these revisions straight, and store the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute. Similarly, people who have nothing to do with the development of a project can still download the files and use them. Most Linux users should be familiar with this process, as using Git, Subversion, or few other similar method is pretty common for downloading needed files, especially in preparation for compiling a program from source code (a rather common practice for Linux geeks)[18].

In case you are wondering why Git is the preferred version control system of most developers, it has multiple advantages over the other systems available (CVS, SVN, MERCURIAL, BAZZAR, and MONOTONE) [63], including a more efficient way to store file changes and ensuring file integrity. If you're interested in knowing the details, check out this page to read a thorough explanation on how Git works. Also if you want to store your project in Git version control system, you can use Git is a command-line

T H3497

tool, but the center around which all things involving Git revolve effectively, the Hub of Git, is GitHub.com, where developers can store their projects and network with likeminded people. The Git version control system is not only for programming projects but also it is or all documents (text file, project file, excel sheet, and) and the file can be stored as well as can be updated in desire time [18].

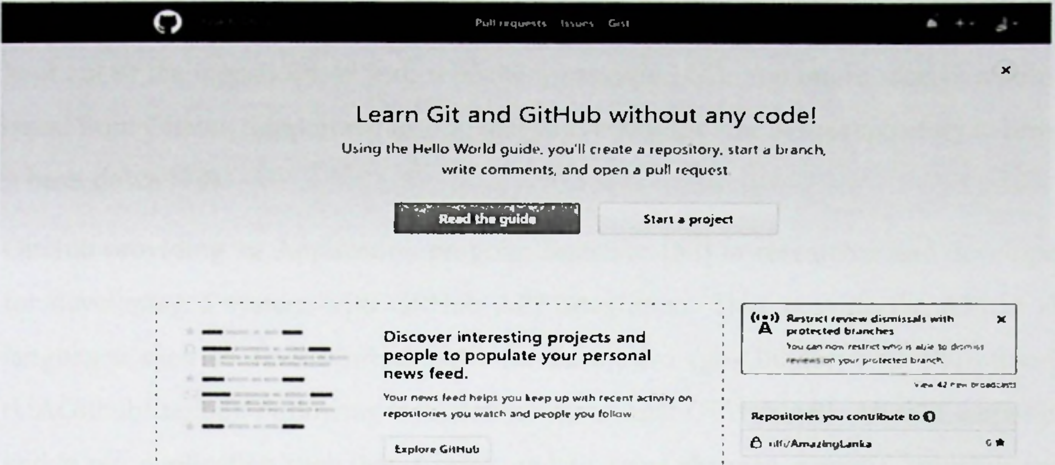


Figure 2.4: Overview of the GitHub page [6].

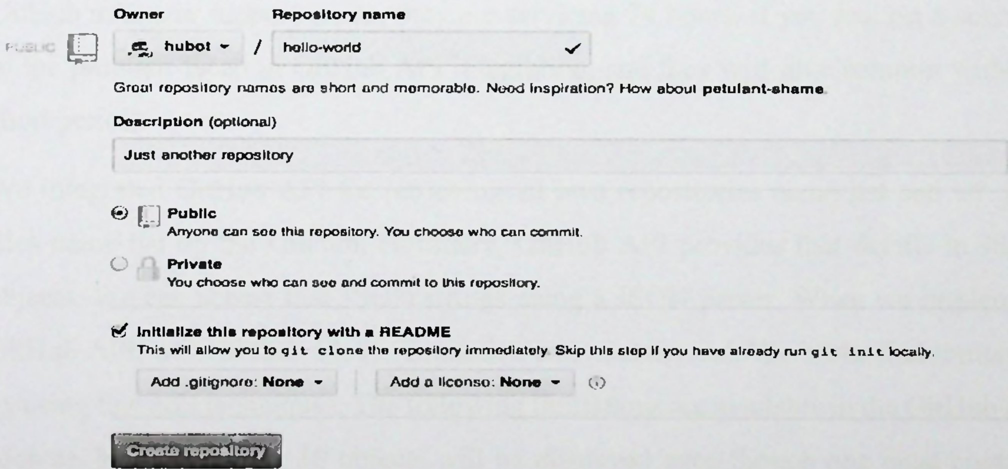


Figure 2.5: Repository in the GitHub [64].

Where repository is a location where all the files for a particular project are stored, usually abbreviated to “repo.” Each project will have its own repo, and can be accessed by a unique URL. GitHub doesn't have the set disk quotas. They try to provide abundant storage for all Git repositories, within reason. Keeping repositories small ensures that their servers are fast and downloads are quick for our users. They recommend repositories be kept under 1GB each. This limit is easy to stay within if large files are kept out of the repository. If your repository exceeds 1GB, you might receive a polite email from GitHub Support requesting that you reduce the size of the repository to bring it back down [18].

GitHub providing an Application program Interface [58] to researcher and developer for developing a system with GitHub API integration. They provide the API in all languages such as Java (GitHub API for Java), Go (go-Github), and Objective-C (UAGithubEngine). Following tasks can be done using GitHub API. All GIT activities within our application such that, Create, update, local changes, commit, branches and tag, etc. As well as we can have all repository details such as, repository name by language and by an author, and class names. When doing an integration of GitHub API, the developers and researchers will face a lot of technical problems. For these issues, GitHub maintain support teams, they are servicing 24 hours, if you request a solution to the problem faced in GitHub API integration, and they will give solution within a short period.

We integrated GitHub API for retrieving all java repositories name list and all .java files name list on the GitHub. Normally, GitHub API provides that details in JSON objects. We can access that JSON strings using a JSON parser. When we implement GitHub API, we will face a lot of limitation and access level. We broke that limitation by using few neat techniques. The following limitations are available in the GitHub API such as, by default only 30 objects will be displayed even though one page contains more than million repositories name, so we can access only 30 objects. But we can extend to 100 JSON object according to Figure 2.7. After doing this extension also we

cannot access all JSON object. Then we used particular date in the query. We can access the projects which are created on particular date. Even though the number of project created on particular date were more than 5000. Then we shrank the query into particular time on particular date, so using one loop we can access all projects name list which are created every hour in every day. For this we used calendar API in java.

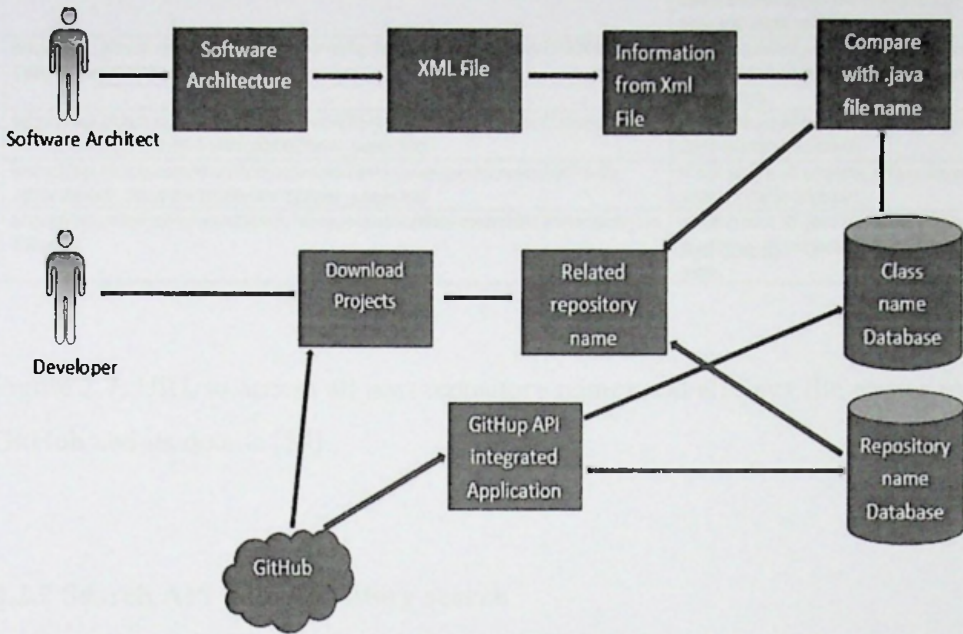


Figure 2.6: GitHub API implemented System overview

Even we used each hour on a particular date, the number of the project was more than 100, and then we split the query into seconds. The number of projects created in a second was less than 100. So we can access all the projects detail which are created in each second in each hour on a particular date. Another very big limitation is, we can make only 10 queries within one minute. It means they allow our crawler to hit their web page only ten times in a minute. Then we used a Java threat to wait one minute after making 10 queries. GitHub API provides so many facilities, in which we used

only repository search for all java repository name in GitHub, and code search for all .java file name list in a particular project.

URL using GitHub API	Details
https://api.github.com/search/repositories?q=language:java	It will return only 30 JSON objects in a page and number of projects in GitHub as well as details of the projects (author, created date, etc.)
https://api.github.com/search/repositories?q=language:java&per_page=100	It will return 100 JSON objects in a page and some details about the projects. But we access only 100 JSON objects
https://api.github.com/search/repositories?q=language:java+created:>2016-01-16&per_page=100	It will return all projects, which are created before 2016-01-16, now also we cannot access all projects.
https://api.github.com/search/repositories?q=language:java+created:2016-01-16T00:00:00Z..2016-01-16T01:00:00Z&per_page=100	It will return all projects, which are created on 2016-10-16 at 01:00:00.
https://api.github.com/search/repositories?q=language:java+created:2015-09-19T00:00:00Z..2015-09-19T00:00:01Z&per_page=100	It will return all projects, which are created on 2016-10-16 at 00:00:01.
https://api.github.com/search/code?q=repo:elastic/elasticsearch%20extension:java&start=0	It will return all .java file names within elasticsearch projects in GitHub, and on first page.

Figure 2.7: URL to access all java repository names and all .java file names in GitHub and its details [59].

2.2.2 Search API and repository search

The Search API is optimized to help you find the specific item you're looking for (e.g., a specific user, a specific file in a repository, etc.). Think of it the way you think of performing a search on Google. It's designed to help you find the one result you're looking for (or maybe the few results you're looking for). Just like searching on Google, you sometimes want to see a few pages of search results so that you can find the item that best meets your needs. To satisfy that need, the GitHub Search API [59] provides up to 1,000 results for each search. Unless another sort option is provided as a query parameter, results are sorted by best match, as indicated by the *Score* field for each item returned. This is a computed value representing the relevance of an item relative to the other items in the result set. Multiple factors are combined to boost the most relevant item to the top of the result list. Find repositories via various criteria,

this method returns up to 100 results per page [19]. Table 1 and figure 2.1 describe parameter need to use to get all Java repositories name on the GitHub.

Name	Type	Description
q	string	The search keywords, as well as any qualifiers.
sort	string	The sort field. One of stars, forks, or updated. Default: results are sorted by best match.
order	string	The sort order if sort parameter is provided. One of asc or desc. Default: desc

Table 2.1: The parameters used in Search API Query [59].

To keep the Search API fast for everyone, they limit how long the individual query can run. For queries that exceed the time limit, the API returns the matches that were already found prior to the timeout, and the response has the incomplete results property set to true. Reaching a timeout does not necessarily mean that search results are incomplete. More results might have been found, but also might not. The q search term can also contain the combination of the supported repository search qualifiers. Suppose you want to search for popular Tetris repositories written in Assembly. Your query might look like this, it means you make a request with the keyword *Tetris*. ***https://api.github.com/search/repositories?q=tetris+language:assembly&sort=stars &order=desc***. In above request, we're searching for repositories with the word Tetris in the name, the description, or the README. We're limiting the results to only find repositories where the primary language is Assembly. We're sorting by stars in descending order so that the most popular repositories appear first in the search results. In this way, we can break the limitations which we discussed in section 2.2.1. But using these parameters and qualifiers we cannot break the number of request in minutes [19].

Table 2.2: The search qualifiers used in query [59].

Search qualifiers	Description
in	Qualifies which fields are searched. With this qualifier you can restrict the search to just the repository name, description, readme, or the combination of these.
size	Finds repositories that match a certain size (in kilobytes).
forks	Filters repositories based on the number of forks.
created or pushed	Filters repositories based on date of creation, or when they were last updated.
user or repo	Limits searches to a specific user or repository.
language	Searches repositories based on the language they're written in.
stars	Searches repositories based on the number of stars.

All API access is over HTTPS and accessed from the https://api.github.com. All data is sent and received as JSON. When you fetch a list of resources, the response includes a subset of the attributes for that resource. This is the "summary" representation of the resource. (Few attributes are computationally expensive for the API to provide. For performance reasons, the summary representation excludes these attributes. To obtain these attributes, fetch the "detailed" representation. When you fetch an individual resource, the response typically includes all attributes for that resource. This is the "detailed" representation of the resource. (Note that authorization sometimes

influences the amount of detail included in the representation.). There are three possible types of client errors on API calls that receive request bodies [19]:

1. Sending invalid JSON will result in a 400 Bad Request response.

```
HTTP/1.1 400 Bad Request
Content-Length: 35

{"message":"Problems parsing JSON"}
```

Figure 2.8: Client errors 1[59].

2. Sending the wrong type of JSON values will result in a 400 Bad Request response.

```
HTTP/1.1 400 Bad Request
Content-Length: 40

{"message":"Body should be a JSON object"}
```

Figure 2.9: Client errors 2[59].

3. Sending invalid fields will result in a 422 Unprocessable Entity response.

```
HTTP/1.1 422 Unprocessable Entity
Content-Length: 149

{
  "message": "Validation Failed",
  "errors": [
    {
      "resource": "Issue",
      "field": "title",
      "code": "missing_field"
    }
  ]
}
```

Figure 2.10: Client errors 3 [59].

All error objects have resource and field properties so that your client can tell what the problem is. There's also an error code to let you know what is wrong with the field. These are the possible validation error codes. Resources may also send custom

validation errors (where the code is custom). Custom errors will always have a message field describing the error, and most errors will also include a documentation_url field pointing to few content that might help you resolve the error.

Missing : This means a resource does not exist.

missing_field : This means a required field on a resource has not been set.

Invalid : This means the formatting of a field is invalid. The documentation for that resource should be able to give you more specific information.

already_exists : This means another resource has the same value as this field. This can happen in resources that must have few unique key (such as Label names).

The GitHub API provides a vast wealth of information for developers to consume. Most of the time, you might even find that you're asking for too much information, and in order to keep their servers happy, the API will automatically paginate the requested items. To start with, it's important to know a few facts about receiving paginated items: Different API calls respond with different defaults. For example, a call to list GitHub's public repositories provides paginated items in sets of 30, whereas a call to the GitHub Search API provides items in sets of 100. You can specify how many items to receive (up to a maximum of 100); but, for technical reasons, not every endpoint behaves the same. For example, events won't let you set a maximum for items to receive. Be sure to read the documentation on how to handle paginated results for specific endpoints. With the ?per_page parameter. Note that for technical reasons not all endpoints respect the ?per_page parameter few will respect that, https://api.github.com/user/repos?page=2&per_page=100, Note that page numbering is 1-based and that omitting the ?page parameter will return the first page [19].

2.2.3 Crawler, and Java JSOUP API

Web crawling is the process by which we gather pages from the Web, in order to index them and support a search engine. The objective of crawling is to quickly and efficiently gather as many useful web pages as possible, together with the link structure that interconnects them [20]. A crawler is a program that visits Web sites and reads their pages and other information in order to create entries for a search engine index. The major search engines on the Web all have such a program, which is also known as a "spider" or a "bot." Crawlers are typically programmed to visit sites that have been submitted by their owners as new or updated. Entire sites or specific pages can be selectively visited and indexed. Crawlers apparently gained the name because they crawl through a site a page at a time, following the links to other pages on the site until all pages have been read [21].

Following features a crawler should provide, *Distributed*: The crawler should have the ability to execute in a distributed fashion across multiple machines. *Scalable*: The crawler architecture should permit scaling up the crawl rate by adding extra machines and bandwidth. *Performance and efficiency*: The crawl system should make efficient use of various system resources including processor, storage and network bandwidth. *Quality*: Given that a significant fraction of all web pages are of poor utility for serving user query needs, the crawler should be biased towards fetching "useful" pages first. *Freshness*: In many applications, the crawler should operate in continuous mode: it should obtain fresh copies of previously fetched pages. A search engine crawler, for instance, can thus ensure that the search engine's index contains a fairly current representation of each indexed web page. For such continuous crawling, a crawler should be able to crawl a page with a frequency that approximates the rate of change of that page. *Extensible*: Crawlers should be designed to be extensible in many ways to cope with new data formats, new fetch protocols, and so on. This demands that the crawler architecture be modular [20].

Crawling is a basic operation of the hypertext crawler (whether for the Web, an intranet or other hypertext document collection) is as follows. The crawler begins with one or more URLs that constitute a seed set. It picks a URL from this seed set, then fetches the web page at that URL. The fetched page is then parsed, to extract both the text and the links from the page (each of which points to another URL). The extracted text is fed to a text indexer. The extracted links (URLs) are then added to a URL frontier, which at all times consists of URLs whose corresponding pages have yet to be fetched by the crawler. Initially, the URL frontier contains the seed set; as pages are fetched, the corresponding URLs are deleted from the URL frontier. The entire process may be viewed as traversing the web graph [20].

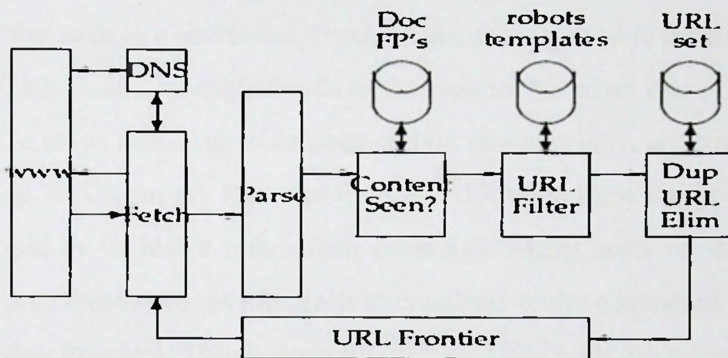


Figure 2.11: The basic crawler architecture [20].

The simple scheme outlined above for crawling demands several modules that fit together as shown in Figure 2.11.

1. The URL frontier, containing URLs yet to be fetched in the current crawl (in the case of continuous crawling, a URL may have been fetched previously but is back in the frontier for re-fetching).

2. A DNS resolution module that determines the web server from which to fetch the page specified by a URL.

3. A fetch module that uses the http protocol to retrieve the web page at a URL.

4. A parsing module that extracts the text and set of links from a fetched web page.

5. A duplicate elimination module that determines whether an extracted link is already in the URL frontier or has recently been fetched [20].

A crawler thread begins by taking a URL from the frontier and fetching the web page at that URL, generally using the http protocol. The fetched page is then written into a temporary store, where a number of operations are performed on it. Next, the page is parsed and the text as well as the links in it are extracted. In addition, each extracted link goes through a series of tests to determine whether the link should be added to the URL frontier. First, the thread tests whether a web page with the same content has already been seen at another URL. The simplest implementation for this would use a simple fingerprint such as a checksum. Next, a URL filter is used to determine whether the extracted URL should be excluded from the frontier based on one of several tests. For instance, the crawl may seek to exclude certain domains (say, all .com URLs) – in this case the test would simply filter out the URL if it were from the .com domain. A similar test could be inclusive rather than exclusive. Many hosts on the Web place certain portions of their websites off-limits to crawling, under a standard known as the Robots Exclusion Protocol. This is done by placing a file with the name robots.txt at the root of the URL hierarchy at the site. Here is an example robots.txt file that specifies that no robot should visit the URL whose position in the file hierarchy starts with /yoursite/temp/, except for the robot called "search-engine" [20].

*User-agent: **

Disallow: /yoursite/temp/

User-agent: search-engine

Disallow:

The robots.txt file must be fetched from a website in order to test whether the URL under consideration passes the robot restrictions, and can therefore be added to the URL frontier. Next, a URL should be normalized in the following sense: often the HTML encoding of a link from a web page p indicates the target of that link relative to the page p. Threads in a crawler could run under different processes, each at a different node of a distributed crawling system. Such distribution is essential for scaling; it can also be of use in a geographically distributed crawler system where each node crawls hosts "near" it. Partitioning the hosts being crawled amongst the crawler nodes can be done by a hash function, or by few more specifically tailored policy [20].

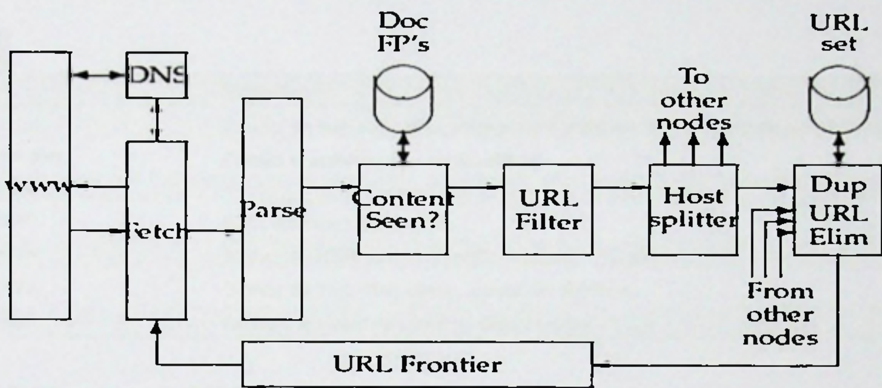


Figure 2.12: Distributing the basic crawl architecture [20].

JSOUP is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. JSOUP implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do [22].

- scrape and parse HTML from a URL, file, or string
- find and extract data, using DOM traversal or CSS selectors
- manipulate the HTML elements, attributes, and text

- clean user-submitted content against a safe white-list, to prevent XSS attacks
- output tidy HTML

JSOUP is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; JSOUP will create a sensible parse tree [22].

Overview Package Class Use Tree Deprecated Index Help

Prev Next Frames No Frames

jsoup Java HTML Parser 1.10.2 API

jsoup Java HTML parser that makes sense of real-world HTML soup

See Description

Packages

Package	Description
org.jsoup	Contains the main Jsoup class, which provides convenient static access to the jsoup functionality
org.jsoup.examples	Contains example programs and use of jsoup
org.jsoup.helper	
org.jsoup.nodes	HTML document structure nodes
org.jsoup.parser	Contains the HTML parser, tag specifications, and HTML tokeniser
org.jsoup.safety	Contains the jsoup HTML cleaner, and whitelist definitions
org.jsoup.select	Packages to support the CSS-style element selector

Figure 2.13: Overview of JSOUP package [22].

2.3 NLP techniques, APIs and tools

The NLP techniques and the Machine learning techniques are used in this research, such as N-gram algorithm, and Dynamic Time Warping speech recognition technique. As well as the implemented software tools are used such as Stanford SpellChecker, WordNet, and Stanford POS tagger.

2.3.1 N-gram Technique

An N-gram is an N-character slice of a longer string. Although in the literature the term can include the notion of the co-occurring set of characters in a string (e.g., an N-gram made up of the first and third character of a word) [23]. N-grams of texts are extensively used in text mining and natural language processing tasks. They are basically a set of co-occurring words within a given window and when computing the n-grams you typically move one word forward (although you can move X words forward in more advanced scenarios). For example, for the sentence "*The cow jumps over the moon*". If N=2 (known as bigrams), then the n-grams would be: *the cow, cow jumps, jumps over, over the, the moon* so you have 5 n-grams in this case. Notice that we moved from the->cow to cow->jumps to jumps->over, etc, essentially moving one word forward to generate the next bigram [24].

If N=3, the n-grams would be: the cow jumps cow jumps over jumps over the over the moon so you have 4 n-grams in this case. When N=1, this is referred to as unigrams and this is essentially the individual words in a sentence. When N=2, this is called bigrams and when N=3 this is called trigrams. When N>3 this is usually referred to as four grams or five grams and so on [24].

In this way, we use this algorithm to get chunks of a word, using the k-gram technique, for an example, when we split a word into grams, append $ to the beginning and end of the string in order to help with matching beginning-of-word and ending-of-word situations. (We will use the dollar character ("$") to represent blanks.) Thus, the word "TEXT" would be composed of the following N-grams: bi-grams: *T, TE, EX, XT, T*, tri-grams: *$TE, TEX, EXT, XT_, T$$,* quad-grams: $TEX, TEXT, EXT$, XT$$, T$$$ In general, a string of length k, padded with blanks, will have k+1 bi-grams, k+1tri-grams, k+1 quad-grams, and so on.

2.3.2 Dynamic Time Warping

Dynamic time warping (DTW) is a well-known technique to find an optimal alignment between two given (time-dependent) sequences under certain restrictions. Intuitively, the sequences are warped in a nonlinear fashion to match each other. Originally, DTW has been used to compare different speech patterns in automatic speech recognition. In fields such as data mining and information retrieval, DTW has been successfully applied to automatically cope with time deformations and different speeds associated with time-dependent data [7].

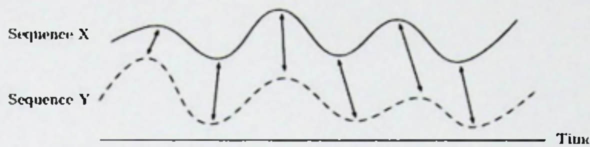


Figure 2.14: Time alignment of two time-dependent sequences [7].

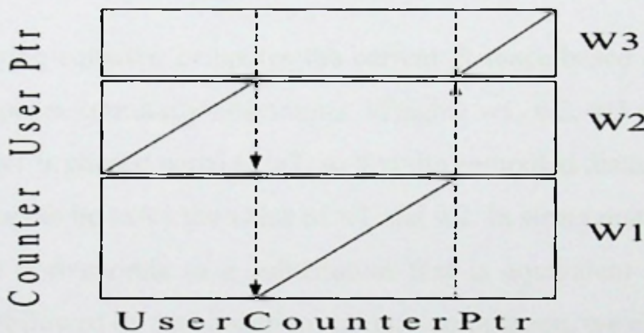


Figure 2.15: Composite identifier recognition [7].

As shown in Figure 2.14, given two signals, two vectors of features, i.e., coefficients extracted from two utterances, the algorithm determines an alignment between the two vectors of features. Let x1, x2. . . xN be the input feature vector extracted from an unknown utterance (x axis) and yl, y2, . . . , yM a feature vector in a dictionary of

signals (y axis) a DTW algorithm performs a warping of the time axis x and y to find the optimal match between the two vectors and, thus, the closest match of the unknown input utterance with prerecorded dictionary entries [7]. To compute the optimal matching given two time series (or strings) x1, x2. . . xN and y1, y2, . . . , yM, the DTW distance D(N,M) is recursively computed. Let d (xi, yj) be a local distance chosen depending on the problem at hand.In speech recognition, such a distance is often the Euclidean distance between feature vectors; for strings, it can be the ordinal distance of characters or just the comparison of the two characters at position i and j. Then, for the intermediate point [7]

D (i, j):

c (i, j) = d(xi, yj)

D (i, j) = min [w1 · D (i − 1, j) + c (i, j), //insertion

w2 · D (i, j − 1) + c (i, j) // deletion

D (i − 1, j − 1) + w3c (i, j)] // match

The recurring equation computes the current distance based on previous values and thus it imposes continuity constraints. Weights w1, w2, w3 are problem-dependent, typically w1 is chosen equal to w2, so that the computed distance is symmetric; w3 is often chosen to be twice the value of w1 and w2. In string matching, if xi differs from yj, then it corresponds to a substitution that is equivalent to the deletion of one character followed by one insertion. In our computation, we choose w1 = w2 = 1 and w3 = 2.The computation is done as follows. It uses a grid built by putting an identifier on the x-axis and the dictionary words on the y-axis, as shown in Figure 2.15. It starts on the bottom-left side of the grid and is performed by computing a distance D (i, j) based on d(xi, yj) for each cell of the grid, i.e., by comparing the corresponding elements on the x and y axis (which are portions of the signal in speech recognition, while they are characters in our application) and finding the local path of minimum cost, i.e., (i − 1, j), (i, j − 1), or (i − 1, j − 1). The computation proceeds by columns

(or rows); once the cost matrix D(i, j) has been filled, the cell D(N,M) contains the minimum alignment cost, i.e., the minimum distance between x1, x2, . . . , xN and y1, y2, . . . , yM. Backtracking from (N, M) down to (0, 0) recovers the warping path corresponding to the optimal alignment of x1, x2. . . xN and y1, y2, . . . , yM [7].

Given a dictionary containing the words rotate and shape, the identifier rotateShape would best match with the words rotate and shape. The identifier splitting problem can thus be brought back to the problem of determining the sequence of R words q (1) . . . q(R), where q (i) represents a word in a given dictionary, such that the distance between the input identifier and the sequence of word is minimized. Figure 2.15 graphically represents this problem, where the x axis represents an identifier (composed of one or more words) and on y axis are entries of a reference dictionary. In Figure 2.15, the dictionary contains three words Counter, User, and Ptr, while the input identifier is UserCounterPtr [7].

2.3.3 Stanford SpellChecker

Stanford University people has developed a spell checker to check the spelling for wrong spelling words. They used N-gram technique to implement this spell checker. If we want to implement the spell checker in our project, we can download the JAR file of the spell

Checker, and then integrate the JAR file into our own project. When we use Stanford spell checker integrated application, by default we will get 10 related words including the correct word. As well as they are maintaining a website to do this spell checking process without the implementation, Figure 2.16 show the online version of Stanford SpellChecker.

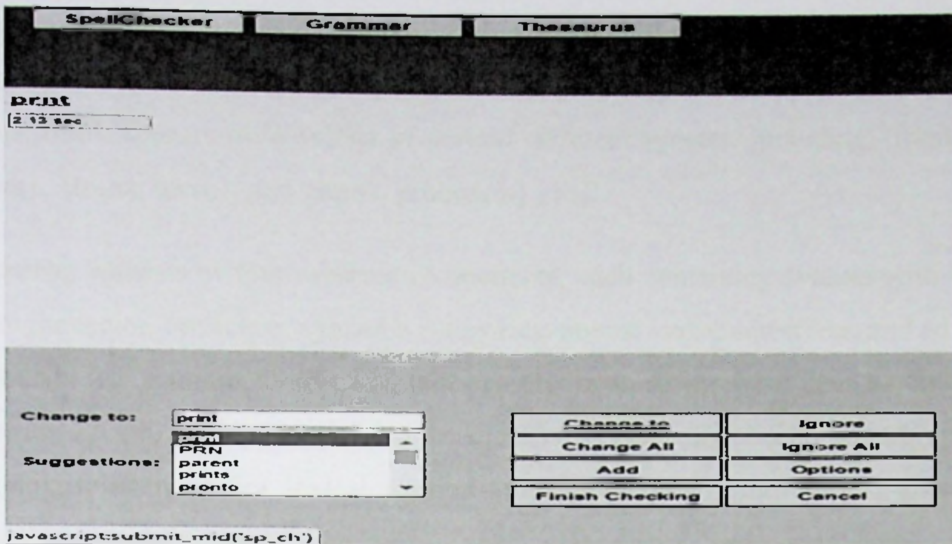


Figure 2.16: Online Version of Stanford SpellChecker [66].

2.3.4 WordNet

WordNet, a manually constructed electronic lexical database for English, was conceived in 1986 at Princeton University, where it continues to be developed. Experiments by researchers in Artificial Intelligence (Collins and Quillian, 1968, inter alia) probing human semantic memory inspired the psycholinguist George A [13].

WordNet is a large semantic network interlinking words and groups of words by means of lexical and conceptual relations represented by labeled arcs. WordNet's building blocks are synonym sets (synsets), unordered sets of cognitively synonymous words and phrases (Cruse, 1986). Each member of a given synset expresses the same concept, though not all synset members are interchangeable in all contexts. Examples are {car, automobile}, {hit, strike}, and {big, large}. All synsets further contain a brief definition, and most include one or more sentences illustrating the synonyms' usage. A domain label (sports, medicine, and biology) marks many synsets. Joint membership

of words in a given synset illustrates the phenomenon of synonymy. Membership of a word in multiple synonyms reflects that word's polysemy, or multiplicity of meaning. Thus, trunk appears in WordNet in several different synsets, including {trunk, tree trunk}, {trunk, torso}, and {trunk, proboscis} [13].

WordNet consists of four separate components, each containing synsets with words from the major, openclass, syntactic categories: nouns, verbs, adjectives, and adverbs. WordNet 2.1 contains almost 118 000 synsets, comprising more than 81,000 noun synsets, 13 600 verb synsets, 19 000 adjective synsets, and 3 600 adverb synsets. Synonymy is the major lexical relation among individual word forms; another is antonymy, as between the pairs {wet} and {dry} and {rise} and {fall}. Morphosemantic relations link words from all four parts of speech that are both morphologically and semantically related (Fellbaum and Miller, 2003). For example, the semantically related senses of interrogation, interrogator, interrogate, and interrogative are interlinked. Conceptual-semantic relations link not just single word forms but entire synsets [13].Concepts expressed by nouns are densely interconnected by the hyponymy relation (or hyperonymy, or subsumption, or the ISA relation), which links specific concepts to more general ones. For example, the synset {mailbox, letterbox} is a hyponym, or subordinate, of {box}, which in turn is a hyponym of {container}. {Mailbox, letter box} is a hypernym, or superordinate, of {pillar box}, which denotes a specific type of mailbox. Hyponymy builds hierarchical 'trees' with increasingly specific 'leaf' concepts growing from an abstract 'root.' All noun synsets ultimately descend from {entity} [13].

Another major relation among noun synsets is meronymy, which links synsets denoting parts, components, or members to synsets denoting the whole. Thus, {finger} is a meronym of {hand}, which in turn is a meronym of {arm}, and so forth. Meronymy in WordNet actually encompasses three distinct partwhole relations. One holds among proper parts or components, such as {leg} and {table}. Another links substances that are constituents of other substances: {oxygen} is a part of {water} and

{air}. Members such as {tree} and {parent} are parts of groups such as {forest} and {family} [13].

Verbs are organized by a several entailment relations (Fellbaum and Miller, 1990; Fellbaum, 1998b). The most prevalent is troponymy, which relates synset pairs such that one expresses a particular manner of the other (e.g., {whisper}-{talk} and {punch}-{strike}). Like hyponymy, troponymy builds hierarchies of several levels of specificity. Other relations are backward entailment (divorce-marry), presupposition (buy-pay), and cause (show-see).

WordNet distinguishes descriptive and relational adjectives. Descriptive adjectives are organized into direct antonym pairs, such as wet-dry and long-short. Each member of a direct antonym pair is associated with a number of 'semantically similar' adjectives. Damp and drenched are semantically similar to wet, and arid to dry. These concepts are said to be indirect antonyms of the direct antonym of their central members, i.e., drenched is an indirect antonym of dry, and arid is an indirect antonym of wet [13].

Two important components of WordNet's design are inheritance and reversibility. Inheritance applies to hierarchy-building relations. If {mailbox, letter box} is enCoded as a hyponym of {box}, and {box} as a hyponym of {container}, then {mailbox, letterbox} is automatically recorded as a hyponym of {container}, via the principle of inheritance. Similarly, if {finger} is a part of {arm}, and {hand} is part of {arm}, then {finger} is necessarily a part of {arm}, too. Many concepts are assigned to both types of hierarchy. Relations are enCoded in WordNet only once between a given pair of synsets or words. The pointer gets automatically reversed, so if {tree} is manually enCoded as a meronym of {forest}, then {forest} will automatically become a holonym of {tree}. And if {mailbox} is manually enCoded as hyponym of {box}, then {box} will automatically become a hypernym of {mailbox}. The lexical (word-word) relations are bidirectional, too [13].

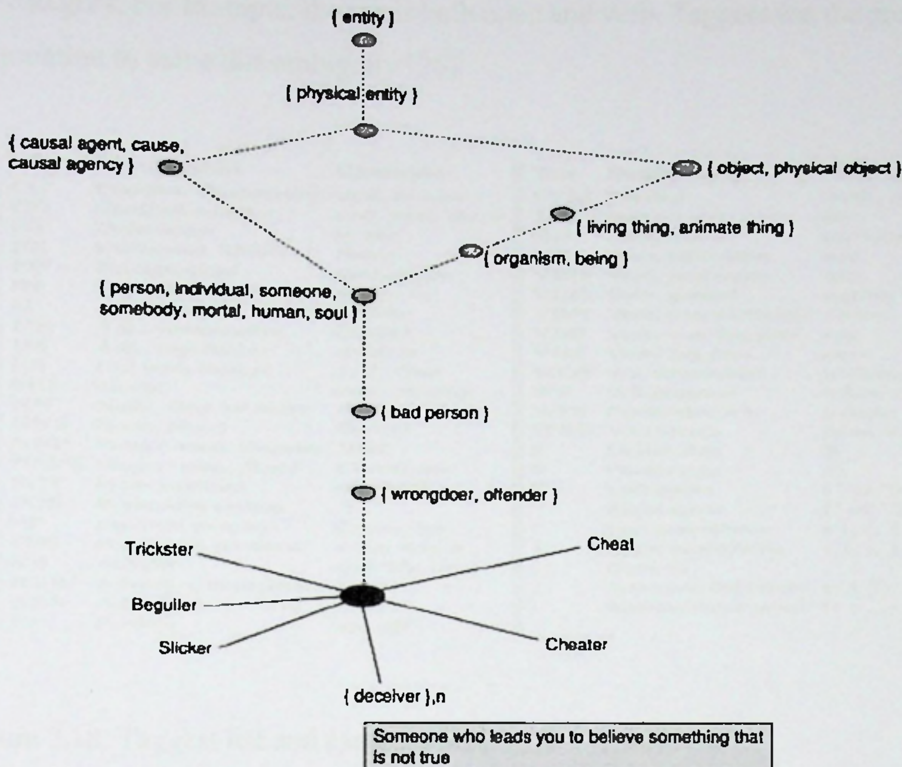


Figure 2.17: A WordNet Noun Tree [13].

2.3.5 Stanford POS tagger

The process of assigning one of the parts of speech to the given word is called Parts Of Speech tagging (POS). It is commonly referred to as POS tagging. Parts of speech include nouns, verbs, adverbs, adjectives, pronouns, conjunction and their sub-categories [10]. A Part-Of-Speech Tagger (POS Tagger) is a piece of software that reads the text in the language and assigns parts of speech to each word (and another token), such as noun, verb, adjective, etc., although generally computational applications use more fine-grained POS tags like 'noun-plural' [25]. Taggers use several kinds of information: dictionaries, lexicons, rules, and so on. Dictionaries have category or categories of a particular word. That is a word may belong to more than

one category. For example, the run is both noun and verb. Taggers use the probabilistic information to solve this ambiguity [26].

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	$
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	``	Left quote	(' or ")
POS	Possessive ending	*'s*	''	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... — -)
RP	Particle	*up, off*			

Figure 2.18: Taggest list and example [65].

There are mainly two type of taggers: rule-based and stochastic. Rule-based taggers use hand-written rules to distinguish the tag ambiguity. Stochastic taggers are either HMM based, choosing the tag sequence which maximizes the product of word likelihood and tag sequence probability, or cue-based, using decision trees or maximum entropy models to combine probabilistic features [26]. Ideally a typical tagger should be robust, efficient, accurate, tunable and reusable. In reality taggers either definitely identify the tag for the given word or make the best guess based on the available information. As the natural language is complex it is sometimes difficult for the taggers to make accurate decisions about tags. So occasional errors in tagging is not taken as a major roadblock to research [26]. Tagset is the set of tags from which the tagger is supposed to choose to attach to the relevant word. Every tagger will be given a standard tagset. The tagset may be coarse such as N (Noun), V(Verb), ADJ(Adjective), ADV(Adverb), PREP(Preposition), CONJ(Conjunction) or fine-grained such as NNOM(Noun-Nominative), NSOC(Noun-Sociative), VFIN(Verb

Finite),VNFIN(Verb Nonfinite) and so on [26]. Most of the taggers use only fine grained tagset. Architecture of POS [26]:

1. Tokenization: The given text is divided into tokens so that they can be used for further analysis. The tokens may be words, punctuation marks, and utterance boundaries.

2. Ambiguity look-up: This is to use lexicon and a guessor for unknown words. While lexicon provides list of word forms and their likely parts of speech, guessors analyze unknown tokens. Compiler or interpreter, lexicon and guessor make what is known as lexical analyzer.

3. Ambiguity Resolution: This is also called disambiguation. Disambiguation is based on information about word such as the probability of the word. For example, power is more likely used as noun than as verb. Disambiguation is also based on contextual information or word/tag sequences. For example, the model might prefer noun analyses over verb analyses if the preceding word is a preposition or article. Disambiguation is the most difficult problem in tagging.

2.4 Identifiers and comments analyzing

Normally all method signature consist of a method name, which often is an action word or it contains an action word, for an example update ( ), delete ( ), addTask ( ). As well as all programmer writing the comments to describe the method before defining a method. The leading comments also have action word which describes the task of the method. Sometimes the action words in both comment and method signature are semantically similar in computer science context, but not in typical natural language documents. Matthew J. Howard and et al [27] proposed a technique to automatically mine these semantically similar words from the leading comments and method signature. First, they identified the leading descriptive comments through

analyzing that whether the comments describe the method or not. Then extracting the action words in analyzed comments. They identified the set of potential verbs (i.e., actions) and then determined these verbs that are followed by a word that could be a noun. These verb-noun sequences become a set of potential main actions documented by the descriptive comment. They started by running Stanford's Log-Linear Part-of-Speech Tagger on the current comment phrase under analysis [27].

Then they checked if any of the non-verb tagged words in the phrase could be a verb in the sense by checking if it could be a possible verb according to WordNet. Sometimes the 'unregister' and 'deregister', are not found in WordNet, but 'register' is a potential verb as indicated by the WordNet. Then the prefix and suffix removed by them, and then run the remaining partial word through WordNet to see if it can be a verb. If so, they tag it as a verb. Then the Samurai (their earlier researched tool) used by them to efficiently break a method into its component words. Then, WordNet is implemented to quickly find possible word forms for the component pieces produced by Samurai. POS tagger then assigns all possible POS tags – from a set of 14 including singular nouns, base verbs, adjectives, past participles, etc. Then they analyzed the comment-code word pair using a typical algorithmic NLP approach [28].

Madhuri.R et al. [29], developed a generic life-cycle model that can be used to improve the software quality by exploiting CSEs. The model that combines code searching through CSEs and mining common patterns of API usages from gathered code examples. The model can be used to assist three main software development tasks:

(1) To learn about an API usage by automatically inferring programming rules (from the mined patterns).

(2) To use mined patterns to detect defects in a program under analysis.

(3) To infer a fix that needs to be applied for a detected defect.

One issue with a larger number of code bases is that mining a larger number of code bases is often not scalable. To address this issue, they propose a life-cycle model based on code searching and mining.

So their approach has two major part such as searching and mining, searching part contains two sub part such that query construction and duplicate elimination. To search an API we need to set a well-formulated query, otherwise, CSEs will give the higher number of irrelevant code example. After getting all result, that result set may have the duplicate result, it means same source code can have a number of copies. On the one hand, it says the code example is widely used and therefore the code example can be trusted more than these code examples that do not have duplicate or multiple versions. So it can bias the results of mining approaches that try to mine common patterns. So using duplicate elimination they filtered out duplicate code. Then in the mining part, they extract few pattern (it contains detail about API).

Normally the developer enters a query into a source code search tool. Depending on the relevance of the results, the developer will reformulate the query and search again. This process continues until the user is satisfied with the results. This novel approach that provides automated support to the developer both in formulating queries and discriminating between relevant and irrelevant search results. In this research, they presented a novel approach to automatically extract natural language phrases from the source code identifiers and method signature (method names, argument, etc) [28].

A tool PARSEWeb [30], is to download the source code interact with Google Code Search Engine (GCSE). Also, after downloading the source code the tool will store the source code into the local code repository, and do the analysis to extract Method Invocation Sequence (MIS). Then the tool will suggest few MIS to the programmer. To search in GCSE, they used an approach that "Source-Destination". Normally the programmers know what type of object that they need, but do not know how to get that object with a specific method sequence. Their approach takes queries of the form

"Source object type-Destination object type" as input, and suggests relevant method-invocation sequences that can serve as solutions that yield the destination object from the Source object given in the query.

Their tool contains a module called query splitter which is used for the situation their approach accepts the query of the form "Source-Destination" and tries to suggest solutions. If no possible MISs are found, their approach tries to split the query by Source and destination separately and do the search again and get the result and do the analysis. As well as their tool contains a module called Sequence Postprocessor, which is used for MIS clustering. After the MIS extraction finish, the Sequence postprocessor will do the clustering. Clustering of MISs helps to identify distinct possible MISs and also reduces the total number of MISs. This reduction of the number of results can help programmers to quickly identify the desired MIS for the given query. To further assist programmers, the sequence postprocessor also sorts the clustered results [30].

Documentation of a software system is not updated due to time pressure and need to reduce the costs. Consequently, the only up-to-date Source of information is the source code and therefore identifiers and comments are key means to support developers during their understanding and maintenance activities. Identifiers can highly affect the source code understandability and maintainability. The substring in a compound identifier as a term, while an entry in a dictionary (e.g., the English dictionary) will be referred to as a word. A term may or may not be a dictionary word. A term carries a single meaning in the context where it is used, while a word may have multiple meanings (upper ontologies like WordNet associate multiple meanings to words). Identifiers are often composed of terms reflecting domain concepts [11], referred to as *"hard words"*.

Hard words are usually concatenated to form the compound identifiers, using the Camel Case naming convention, *e.g.*, *drawRectangle*, or underscore, *e.g.*, *draw_rectangle*. Sometimes, no Camel-Case convention or another separator (*e.g.*

Underscore) is used. Also, acronyms and abbreviations may be part of the identifier, *e.g.*, *drawrect* or *pntrapplicationgid*. The component words *draw, application*, the abbreviations *rect, pntr*, and the acronym *gid* (*i.e.*, group identifier) are referred to as *"soft-words"*. Often programmers build new identifiers by applying a set of transformation rules to words, such as dropping all vowels (*e.g.*, *pointer* becomes *pntr*), or dropping one or more characters (*e.g.*, *pntr* becomes *ptr*) [7].

WordNet (which contains around 90,000 entries) or dictionaries used by spell checkers, such as a-spell (which contains around 35,000 English words in a typical configuration). Each dictionary word may be associated with a set of known abbreviations in a way similar to a thesaurus. For example, the *pointer* entry in the dictionary can be associated to abbreviations *pntr, ptr* found as terms composing identifiers. Thus, if *pntr* is matched, the algorithm can expand it into the dictionary term *pointer*. The overall idea is to identify near-optimal matching between substrings in identifiers and words in the dictionary, using an approach inspired by speech recognition. They take input x1, x2, x3… xn, and put it in x axis, y1, y2, y3….yn from the dictionary, and put it in y axis [7].

Then they find the optimal match between the two vectors and, thus, the closest match of the unknown input utterance with prerecorded dictionary entries. Using distance measure they find shortest distance word, here distance is often the Euclidean distance between feature vectors. They use the transformation rules to match the word, the available transformation rules are the following: *Delete all vowels:* All vowels contained in the dictionary word are deleted, *e.g.*, *pointer* → *pntr*; *Delete Suffix:* suffixes—such as *ing*, *tion*, *ed*, *ment*, *able*—are removed from the word, *e.g.*, *improvement* → *improve*; *Keeping the first* n *characters only*: the word is transformed by keeping the first n characters only, *e.g.*, *rectangle* → *rect* for n = 4 *Delete a random vowel:* one randomly chosen vowel from the word is deleted, *e.g.*, *number* → *numbr*;

Delete a random character: i.e., one randomly-chosen character is omitted, *e.g.*, *pntr* → *ptr* [7].

Developers spend the majority (80%) of their time to maintaining the source code, both industrial and open source developers often submit their code for review prior to check-in [31]. Unfortunately, programmers are often unaware of coding conventions because inferring them requires a global view, one that aggregates the many local decisions programmers make and identifies emergent consensus on style. NATURALIZE [31], a framework that solves the coding convention inference problem for local conventions, offering suggestions to increase the stylistic consistency of a code-base. NATURALIZE can also be applied to infer rules for existing rule-based formatters. NATURALIZE is descriptive, not prescriptive: it learns what programmers actually do. When a code-base does not reflect consensus on a convention, NATURALIZE recommends nothing, because it has not learned the thing with sufficient confidence to make recommendations. As NATURALIZE detects identifiers that violate code conventions and assists in renaming. It is the first tool they are aware of that uses NLP techniques to aid refactoring. The techniques that underlie NATURALIZE are language independent and require only identifying identifiers, keywords, and operators, a much easier task than specifying grammatical structure [31].

2.5 Analyzing our research with related researches

Few amount of research are slightly related to our research but not fully. As well as the researchers did not start with a Software architecture, instead of start with a Software architecture, they start with the source code searching. In source code searching, few of researchers are suggesting the keyword searching as well as few of them are giving negative feedback to the keyword search. However, we recommend for the keyword search because we start our task from a Software Architecture. The Development process will start from a Software Architecture, so we started with a

Software Architecture, as well as we can get few information as keywords from the Software Architecture. The source code searching process should be in the development process because of that we will have keywords in development process after designing process, so keyword search method is suitable search method for source code searching. In several research, the researchers are telling the reason of rejecting keyword code searching is getting few irrelevant code with relevant code when they do the source code search. But we have developed a solution via our developed framework to overcome the issue.

Most of the research are starting with code search and finishing with downloading the relevant projects or source codes. Most of the research are not analyzing the result after downloading process, but in our research, a big part of the research are analyzing the results. And few of the research are starting from source code analyzing, in the research the researchers did not include code searching. Nioosha Madani, Latifa Guerrouj, and et al. [7], used Dynamic Time Warping technique to recognize the words from the Source code, as we discussed the technique in section 2.3.2, DTW is a speech recognition technique can be used to identify the real words from the composite identifiers and we can use this technique to identify the words from the abbreviated word terms. We followed their approach in our research but using DTW, it took more time to identify the real words from the identifiers. As well as it did not identify the real word for some abbreviated term derived from the identifiers, then we used N-gram technique and Stanford SpellChecker to identify the real words from the identifiers.

We used the GitHub API which is not used in the related research to our research. As we discussed in section 2.2.1, the GitHub provides an API to researchers and programmer as open source, to integrate the API into their own project. Using the API, we can do all GIT activities as well as we can do a few additional things, such as, if we want to have all repositories names in the GitHub, all particular programming language repositories (such as all java projects name, and all C++ projects names), and all file names (such as all .java file names, and all .c file names) we can have that.

Using the GitHub API, we can reduce the time to download unwanted projects which mean, it will check the repository names with the keywords and then download the projects. In this way, we can reduce the number of unwanted projects in the downloading process, so we will reduce the time to analyze the downloaded projects to identify the most relevant project among them. Even though we integrated GitHub API, it will download most relevant projects and slightly relevant projects. We need to analyze further to identify the most relevant projects.

Chapter 3

Research methodology and Proposed Framework Architecture

3.1 Research Methodology

As it is indicated in the title, this section includes the research methodology of the dissertation. In order to satisfy the objectives of the dissertation, a qualitative research was held. The main characteristic of qualitative research is that literature revive. As we discussed in the section 2.5, there were no actual related research to our research but partial work of our research had done by some researchers. We launched the research 2017 may 05 officially, but we started the literature revive before two month to the launching of the research.

After finishing the literature revive we designed our research and we submitted the proposal of our research. We spent around six month for literature revive and design process. After the designing process, we started to develop an algorithm to find the answer of research question on 2015 October 10. We spent two month to develop the algorithm. And then, we started the implementation part of our framework on 2015 January 01. And then we spent around ten month for implementation and initial testing. After the implementation, we collected some sample data (sample projects) from Git version control system.

Finally we started the evaluation of our developed framework with the collected sample data. We did the evaluation manually, even though our algorithm will do the proposed work automatically, we needed to check whether the algorithm worked properly, that is what we had done the evaluation manually. We started the evaluation on 2016 November and we finished the evaluation 2017 February 11. In the evaluation part, we targeted five projects from GitHub, and we checked whether the developed

framework identify the targeted five projects, finally we conclude that the developed framework was working successfully.

Designing and implementing the framework were the very problematic and time consuming tasks, in which we spent considerable time on the GitHub integration and analyzing the downloaded projects from the forges. Our framework consist of 10 module to carry out the tasks such as, Extract the information from the Software Architecture (class diagram), Download the projects (crawl the projects) depend on the information from the class diagram, Decompress the downloaded projects in the compressed format, parse all the Java classes into the Abstract Syntax Tree to access the identifiers and the comments, identify the meaningful words from the identifiers, identify the action words from the comments, analyze the words and the information of the class diagram, and then rating the relevant .java files.

As we mentioned earlier the GitHub integration consist of few sub-modules such as the project name dumber, and the class name dumber. Figure 2.6 and Figure 3.1 describe the modules of our framework.

This chapter provides a depth details of our design, developed module and their process as well as how they interact with each other. Next chapter will detail on the implementation of the each module and used existing tools. Section 3.1, describes the XML parser and its usages. Section 3.2 describes the GitHub API integration and its usages. Section 3.3, describes three types of crawlers we developed to download the relevant project from the internet depend on the information from the class diagram and the de-compressor to extract the downloaded projects. Section 3.4 describes the Abstract Syntax Tree and its usage in this research. Section 3.5 explains types of identifiers, and the splitter for the composite identifiers to extract the string terms or the words separately from the identifiers. Section 3.6, describes the developed spell-checker, the word-finder and their usage in our research. Section 3.7 describes analyzing the comments in the source code and extracting the action verb from the source code. Section 3.8 describes the final module of our developed framework that

is matching and rating module. Finally, Section 3.9 summarizes our all developed module and their process as well as their interaction.

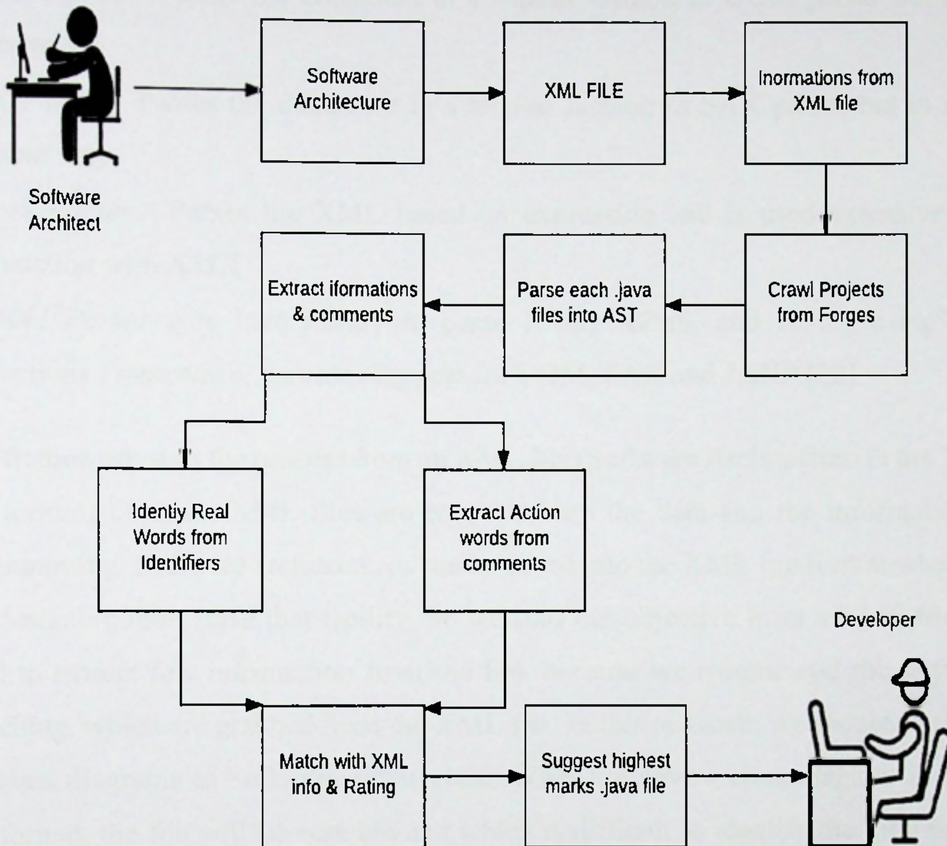


Figure 3.1: Overview of our System

3.2 XML Parser

XML Parser provides a way how to access or modify data present in an XML document. Java provides multiple options to parse the XML document. Following are various types of parsers which are commonly used to parse the XML documents [32].

- *Dom Parser* - Parses the document by loading the complete contents of the document and creating its complete hierarchical tree in memory.
- *SAX Parser* - Parses the document on event-based triggers. Does not load the complete document into the memory.
- *JDOM Parser* - Parses the document in a similar fashion to DOM parser but in an easier way.
- *StAX Parser* - Parses the document in a similar fashion to SAX parser but in more efficient way.
- *XPath Parser* - Parses the XML based on expression and is used extensively in conjunction with XSLT.
- *DOM4J Parser* - A Java library to parse XML, XPath, and XSLT using Java Collections Framework, provides support for DOM, SAX and JAXP [32].

Our framework start the process from an XML file (Software Architecture in the XML file format) because, XML files are easy to share the data and the information in programming. Software architectures can be saved into the XML file format where all the designing tools have that facility. So we start our objective from a XML file, we need to extract few information from the file, because we recommend the keyword searching, which are grabbed from the XML file. In this research, we focused only on the class diagrams as Software architecture. When we have a class diagram in XML file format, the file will be very big and which is difficult to identify the information of the class diagram. Because of that we decided to use one of the XML parser we mentioned above. If we parse the XML file into the parser, it will represent all the information in a hierarchical order. Root of the class diagram represents the class name, first child level represents the method names, and the other child levels represent the attributes. In this way we can extract all the class names, method names, and attribute names separately and easily.

---

* *https://www.tutorialspoint.com/java_xml/java_xml_parsers.htm* [53]

3.3 GitHub API Integration

This module explains about the integration of the GitHub API to achieve the objective described in section 2.2. This module includes two sub-modules such as the Java projects name dumber, and the Java class names dumber. In this research we focused only on the Java projects, therefore, we developed these sub-modules. Each module deal with the GitHub web-pages to make two repositories.

3.3.1 Java project names dumber

GIT version control system contains a billion numbers of projects written in JAVA, C++, C#, VB.NET, and few other programming languages. When we try to retrieve the projects written in a particular programming language will be impossible without the proper information about the projects such as author details of the projects, created date of the projects, and the name of the projects. According to our GitHub API implemented System Overview in Figure 2.6, we extract the information from the XML file (Software Architecture (class diagram)), and then we need to download few projects depend on the information. If we try to download the projects using the information (class names, method names, and attribute names) as keywords in the keyword searching technique, we will download a huge amount of projects. Even though we intended to download the relevant projects, we have to download more irrelevant projects. It will be a very big task to find the relevant projects among the huge number of irrelevant projects. Also, it is a time-consuming task, analyzing the irrelevant projects and we have to pay for internet data usage to download these projects.

Sometime, we will suggest a few amount of irrelevant projects while analyzing the downloaded project. If we suggest the irrelevant projects the entire development process will be delayed, then the developer need to speed up the development,

therefore, the speed up will affect the quality of the product. Therefore we can prevent the problems discussed above by downloading the predefined projects where we can avoid to download the unwanted projects. We need to have all the projects name in the GIT version control system to predefine the projects. Therefore, we developed the Java project names dumber module with the GitHub API to predefine the projects are to be downloaded. We can dumb all the Java project names using this module, then analyze the project names and the information retrieved from the XML file (class diagram), and then we will get the relevant project names which are to be downloaded.

3.3.2 Java class names dumber

As the GIT version control system contain a billion numbers of projects, each project contains a lot of source code files. For an example, each Java projects contain hundreds of .java files and big projects may contain thousands of .java files. If we need to predefine the projects are to be downloaded, we cannot predefine without analyzing the project repository with the information from the class diagram. We will dumb only all the Java project and exclude the details of the .java file of the projects using project names dumber, so we need to extract few more information about the projects. Therefore we developed this Java class names dumber module with the GitHub API to dumb all .java file names belongs to all Java projects available in the GIT version control system.

With the help of the GitHub API, this module will make a repository to store all the .java file names into our local machine. When a class diagram enters to the implementation phase, information of all the classes in the class diagram will be extracted using our XML parser module. On the one hand, we will have the information from the class diagram (the class names, the method names, and the attributes names) and on the other hand, we will have all the .java file names using this module, and then we will identify the relevant class names from the created repository

using this module by analyzing the both information. While we try to get all the .java file names we will get few more information by the API such as project names of the file names, author of the project, repository created date in GitHub, the number of commits, and few other information. We will find the class names and related project names after the analyzing process. And then, we will download the predefined project list. We will attain the following advantages by the usage of this module such as, we give a solution to the problem we discussed in section 3.2.1, we can save our valuable time instead of wasting with the irrelevant projects, and we will reduce the money cost by preventing the download of the irrelevant projects via saving the internet data.

3.4 Crawlers and Decompressor

We discussed about the crawler in section 2.2.3. As we discussed, we developed three types of crawlers to fetch the URL of the projects to be downloaded using the keywords from the class diagram. In this research, we focused only on three types of forges such as SourceForge [1], GitHub [6], and GoogleCode [2]. These three types of forges are different with each other, therefore, we developed separate three types of crawlers for three types of forges. We used the JSOUP Java API to develop these crawlers. While we use these crawlers, we need to give a seed URL, the crawler will start the crawling process from the URL and go ahead. We have given www.Sourceforge.com, www.github.com, www.googleCode.com as seed URLs to our crawlers.

Web pages are connected together, one web page is linked to another page using hyperlink and anchor text. For an example, if we want to include another web page in our web page we can't copy and paste all the data from that web pages into our web page. The following fragment of HTML code from a web page shows a hyperlink pointing to the home page of the Journal of the ACM: Journal of the ACM. In this case, the link points to the page http://www.acm.org/jacm/ and the anchor text is Journal of the ACM. Clearly, in this

example, the anchor is descriptive of the target page. But then the target page (http://www.acm.org/jacm/) itself contains the same description as well as considerable additional information on the journal [20].

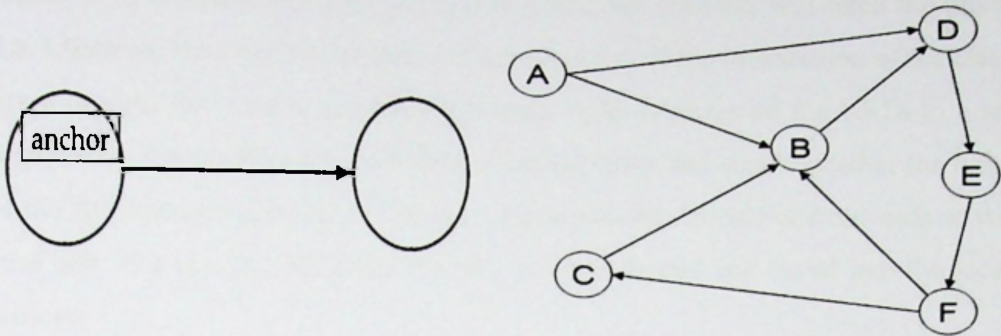


Figure 3.2: Web graph joined by a link [20].

We can view the static Web consisting of static HTML pages together with the hyperlinks between them as a directed graph in which each web page is a node and each hyperlink a directed edge [20]. Figure 3.2 shows two nodes A and B from the web graph, each corresponding to a web page, with a hyperlink from A to B. We refer to the set of all such nodes and directed edges as the web graph. Figure 3.2 also shows that (as is the case with most links on web pages) there is few text surrounding the origin of the hyperlink on page A. This text is generally encapsulated in the href attribute of the <a> (for anchor) tag that encodes the hyperlink in the HTML code of page A, and is referred to as anchor text. As one might suspect, this directed graph is not strongly connected: there are pairs of pages such that one cannot proceed from one page of the pair to the other by following hyperlinks. We refer to the hyperlinks into a page as in-links and these out of a page as out-links. The number of in-links to a page (also known as its in-degree) has averaged from roughly 8 to 15, in a range of studies. We similarly define the out-degree of a web page to be the number of links out of it [20].

First of all, our XML parser will extract the information (class names, method names, and attribute names) from the class diagram, and then the crawlers will use this information as the keywords. The crawlers will start the crawling process with the seed URLs and the keywords if anything in the web pages matches with the information extracted from the class diagram, and if it is a link, the crawlers will fetch the link's URLs. Likewise, the crawlers continue the process for all the information of the class diagram or until the crawler reaches dead-ends. After fetching all the URLs to a set data structure, the module will analyze the fetched links and check whether the links are a file (it means project file will be in compressed file format) or dead-ends or the normal link. If it is a file link, that file will be downloaded and saved into the local repository.

After the crawling process, we will have a few amount of relevant projects. All forges are giving the download facility for the project in compressed file format. So all the downloaded project will be in compressed file format, as it is in the compressed format, we cannot do any operation on it. Our approach is in source code level, so we need to deal with all the .java files, all .java files and few other configuration files are combined together inside the compressed file. For that purpose, we developed decompressor module. This module extracts the compressed project in the format such as .zip, .rar, .gz, .7z, and etc.

3.5 Abstract Syntax Tree

The representation of the source code as a tree of nodes representing constants or variables (leaves) and operators or statements (inner nodes). Also called a "parse tree". An Abstract Syntax Tree is often the output of a parser (or the "parse stage" of a compiler) and forms the input to semantic analysis and code generation (this assumes a phased compiler; many compilers interleave the phases in order to conserve memory) [33].

Note that the Abstract Syntax Tree reveals the lexical/syntactical structure of the program text - what blocks and statements are lexically contained within in what. This may - or may not - be related to the semantic structure of the program. For example, in most OO languages with inheritance, the inheritance hierarchy is not revealed by examining the arrangement of the AST [33].

Abstract syntax trees are data structures widely used in compilers, due to their property of representing the structure of program code. An AST is usually the result of the syntax analysis phase of a compiler. It often serves as an intermediate representation of the program through several stages that the compiler requires, and has a strong impact on the final output of the compiler. Following code can be represented in an Abstract Syntax Tree, Figure 3.3 shows the Abstract Syntax Tree [33].

Our approach is in source code level, after de-compressing all the downloaded project, we will have a huge amount of projects, and so we need to identify the most relevant projects from the project pool. We will get following information from the class diagram such as class name, method names, and attribute names. So we need to access all the identifiers from the downloaded projects to compare with the information from the class diagram. If the identifiers and the information are matched then an amount of marks will be given to the respective .java file. If the total mark of the .java file is over the threshold marks, we will suggest the .java file to the developer as relevant source code. Following paragraph is a quote of our research.

"Program identifiers are a fundamental Source of information to understand a software system. Because programmers choose program names to express the concepts of the domain of their software. (Methods, classes, fields). [34]"

Also, we extract few action word from the comment because the task of a class or a method will be described by the developers through the leading comments of the class or the methods. The leading comments of the methods will explain the whole process of the methods. It will be difficult to understand the code in a situation, when another

developers read the code or the same developer try to edit the code after a long time. They will understand the context of the code when they read the comments. Also most of the methods are doing the action (delete, add, send, and etc). Therefore all the developers are selecting an action word as the method identifiers. Also, they use an action word in the comments to describe the methods. We developed a sub-module to extract the comments from the source code using the Abstract Syntax Tree.

while $b \neq 0$

if $a > b$

$a := a - b$

else

$b := b - a$

return a

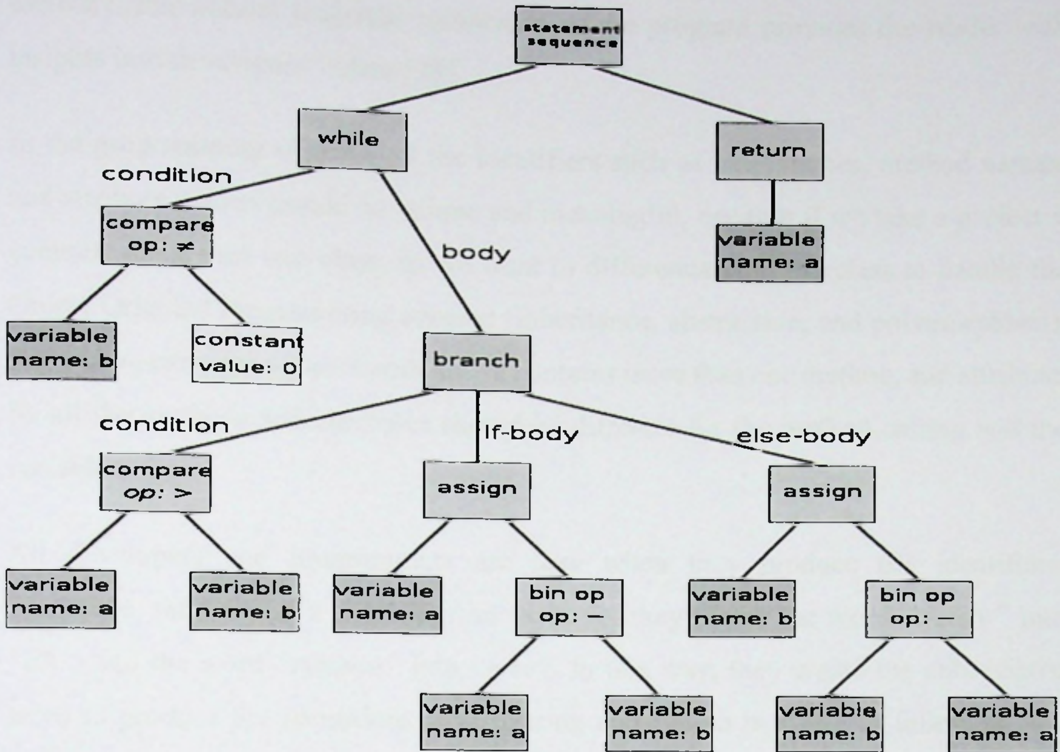


Figure 3.3: Abstract Syntax Tree example [52]

3.6 Identifier Splitter

Source code identifiers such as names of classes, interfaces, methods or functions, variables and formal parameters or arguments can be viewed as a sequence of characters (a string) consisting of one or many tokens [35]. The tokens or terms which constitutes an identifier can be a word (a word in a standard english dictionary), well-known and commonly used acronyms (such as the country USA or the company IBM and even short-forms like Mr. or max) or domain-specific abbreviations (such as str for string, pntr for pointer, rect for rectangle)[35]. Program identifiers are a fundamental Source of information to understand a software system. Because programmers choose program names to express the concepts of the domain of their

software, this natural language component of the program provides the reader with insights into developers' intent [28].

In the programming context, all the identifiers such as class names, method names, and attribute names should be unique and meaningful, because if we take a project it contains more than one class, so we want to differentiate all the class to handle the Object Oriented Programming concept (inheritance, abstraction, and polymorphism). Also, if we consider a source code file, it contains more than one method, and attribute. So all the methods and attributes should be different for the method calling and the variable using.

All developers and programmers are lazy when they produce the identifiers. Sometime, they shrink a word. For an example, they shrink the word *"delete"* into *"dlt"*, and the word *"remove"* into *"rmv"*. In this way, they create the abbreviated word to produce the identifiers. Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc. But, it is not forced to follow. So, it is known as a convention not a rule. Constants name should be in uppercase letter. E.g. *RED, YELLOW, MAX_PRIORITY* etc, and some other naming convention should be followed when naming another variable. But some of the programmers and the developers are not following these conventions.

A research shows that a significant percentage of identifiers in source-code do not contain explicit markers between tokens and this makes the task of identifier splitting as non-trivial. For example, Butler et al. [32], performed analysis of a sample dataset and found that approximately 15% of the identifiers in the sample dataset are non-trivial to tokenize because of lack of explicit markers (all lowercase and all upper-case letters without the underscore or special symbol to separate concepts) and presence of abbreviations, non-dictionary terms and unconventional usage of capitalization and digits [32]. The developers need to follow the Naming Convention, such as Camel case Naming convention when they connect more than two abbreviated terms to create an

identifier, for an example, *isDeccending*, *isPrimaryKey*, *displayingEvents*, *createTempTable* or the explicit splitter (_, $, number, etc), for an example, *remove_file, add$row, search6key*. It will difficult to identify the dictionary words from these types of identifiers. If we want to get the meaningful words from the identifiers we must split the composite words. Then only we can identify the meaningful words from the split terms.

For this purpose, we developed a module to split composite identifiers into separate string terms or words. If they follow a naming convention or the explicit separator in identifiers, our identifier splitter is enough to split. But, if they do not follow the naming convention to create the identifiers, it will be a significant challenge to split the identifiers. We have developed an algorithm using the NLP technique for this challenging process. The algorithm will be discussed in the following sections. We will parse all the identifiers into this Splitter module after extracting all the identifiers from the class diagram. It will split the good identifiers which follow any naming convention into the string terms. Otherwise, it will give the same identifiers as a result which is bad identifier does not follow any naming convention. String terms may be a meaningful dictionary words or just a string token. If it is a meaningful dictionary word we will directly take the word into matching process. Otherwise, the token will be sent to the next process to identify the meaningful word from the token.

3.7 Spell Checker and Word Finder

As our approach is based on the identifiers, we need to analyze the identifiers to get the meaningful dictionary words from the identifiers. We will get three types of the result after the usage of the identifier splitter module. The first type may be the meaningful English dictionary word, the second type may be a string token (we need to identify the meaningful dictionary word of the tokens). And the third types may be the bad identifiers which do not follow the naming convention. E.g. *nummessg*

("number" and "message" words are connected together), *wrdcntr* ("word" and "counter" words are connected together). It is very difficult to identify the meaningful dictionary word from the third types of the result.

3.7.1 Spell checker for Good Identifiers

Good identifiers which are following any naming convention whether the Camel Case naming convention (*rmvFle, dteRow, and sndMssg*) or the explicit separator (*sleect_name, fnd$value, and get8rslt*). We parse these type of identifier into our splitter, it will split the identifiers where the Camel Case or explicit separator is started in the identifiers. This process will return two types of results such as meaningful dictionary words and string tokens. As we discussed earlier, the programmers are lazy, they shrink the meaningful words into the string tokens and then they connect more than two string token together to produce an identifier (E.g *mkeReqst, strReader*). If it is a dictionary word, the word will be sent into matching process, for an example, *removeFile* will be separated *"remove"* and *"file"* (both are dictionary words, so these words will be sent to the comparing process to match the words with the information of the class diagram).

If it is not a meaningful dictionary word, it means a string token such as *rmvFle* will be split into *rmv* and *fle,* both words are not dictionary words. As a human, we will realize and identify the word *"remove"* to the token *"rmv"* and the word *"file"* to the token *"fle"* using our intelligence. But the computer is an electronic machine it cannot think. So we need to give this power artificially to the computer using machine learning techniques or Natural Language Processing techniques. First, we used a machine learning technique that was Dynamic Time Warping (DTW) and we found few drawbacks. And then, we used an NLP technique (N-gram) to overcome the drawbacks while the usage of the DTW technique. Using the N-gram NLP technique, the machine identifies the meaningful dictionary word to the string token. The machine could not

identify the exact word when it tries to identify the dictionary word, rather than it will suggest the related words including the exact word.

3.7.2 Word Finder and Spell checker for Bad Identifiers

Bad identifiers which are not following any naming convention, but it may contain more than two word or one word in abbreviated form. Developers shrink the words into abbreviated term and connect more than two abbreviated terms together to produce an identifiers without following the naming convention. For an example, when we take above mentioned example *rmvfle,* it contain two abbreviated words, as a human we can understand these two abbreviated words and its original dictionary word, but computer cannot understand because it does not have thinking power. As well as there are no any algorithm or technique to find these meaningful dictionary words. But Nioosha Madani, et al. [7], developed an algorithm based on a modified version of the Dynamic Time Warping (DTW) (is a machine learning speech recognition algorithm, read the section 2.3.2 for further details) algorithm proposed by Ney for connected speech recognition [36] (i.e., for recognizing sequences of words in a speech signal) and on the Levenshtein string edit-distance [37]. We followed their algorithm, sometime, their algorithm gives wrong word as a result to a particular string token. Also it could not identify the meaningful dictionary word to a particular string token. As well as the time consuming was very high. The time cost is very less when compare with the usage of the DTW technique with the NLP technique.

If the algorithm lead to find the irrelevant dictionary word for a particular string token, it leads to select the irrelevant source code as a relevant source code. And then the developers will use the source code as a pillar to their project. They will realize that the source code is irrelevant in the middle of the development, so it will be a very big time waste. For that reason, we developed an algorithm to overcome above mentioned challenges using N-gram technique (Read section 2.3.1 to understand the N-gram

technique). We developed a Spell-Checker and a dictionary to identify the meaningful words from the string token extracted from the identifiers. We used the Stanford SpellChecker as a pillar to develop our SpellChecker. Our SpellChecker gives 10 words for each string token. So usage of Our SpellChecker increases the number words to be checked with the information of the class diagram. We will discuss deeply about this 10 word list in Evaluation chapter. The algorithm is following, as well as see the Figure 3.4 for further explanation.

- If the identifiers is a good identifiers (it follows whether the camel case or the explicit separator in naming convention) split the identifier using our splitter.

- If the split terms are meaningful dictionary word, send it into comparing process.

- Split terms are string token, send it into our spell-checker and identify the meaningful words, and then send it into comparing process.

- If the identifier is a bad identifiers, and the token length is > 2, use 2-gram to make the chunk and check the spelling.

- If the SpellChecker identify the word, send the word into comparing process, remove that chunk and repeat the work.

- If the Spellchecker do not identify the word, increase the N by one in the N-gram technique and repeat the process until the length of the words is equal to N.

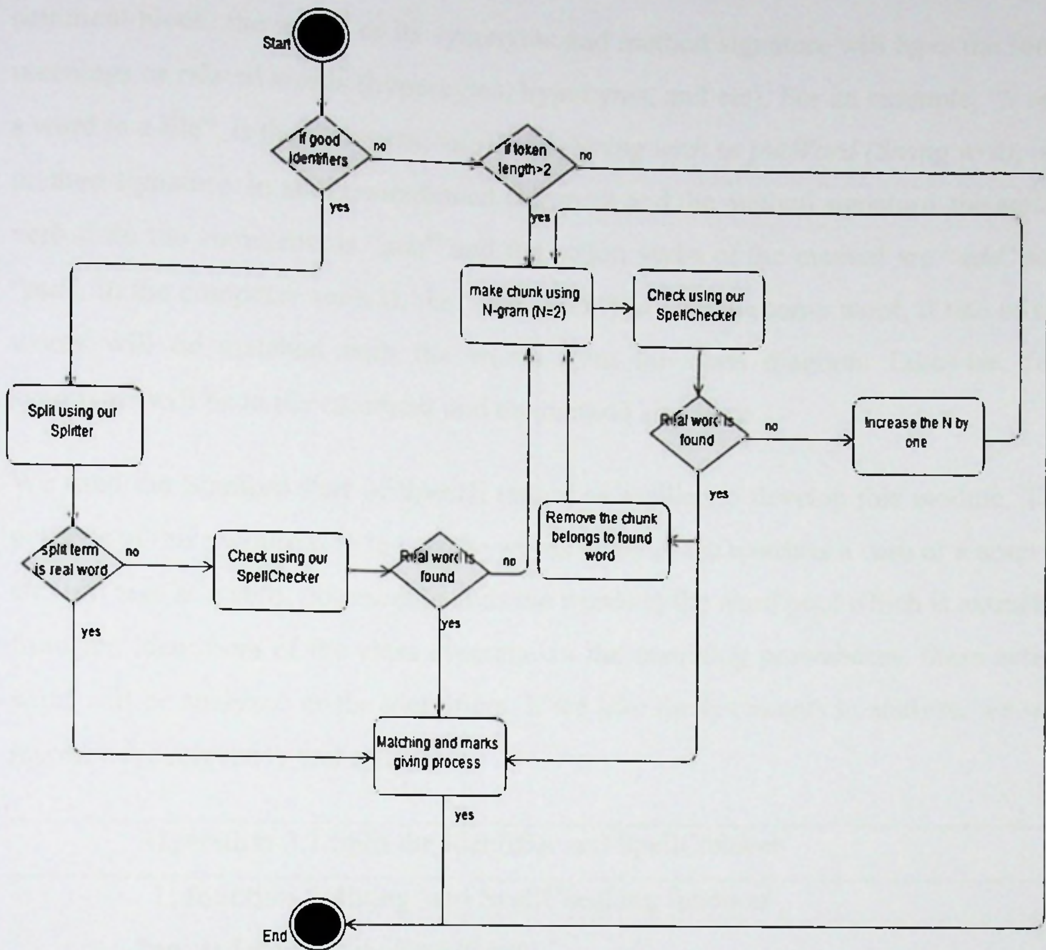


Figure 3.4: Overview of our proposed algorithm

3.8 Action word extractor for comments

Action verb used in the descriptive leading comment for a method is semantically similar to the action verb used in the signature of the documented method. Descriptive leading comment to be a comment block placed before a method signature that provides the reader with the overall summary of a method's actions. That is, a descriptive comment describes the intent of the code succinctly [27]. So comments will explain the task of a method or class. If we identify the action verb from a

comment block, the words or its synonyms and method signature will have the same meanings or related words (hypernyms, hyponyms, and etc). For an example, "// add a word to a file", is the comment, *addWord (String wrd)* or *putWord (String wrd)*, is a method signature. In above-mentioned comment and the method signature, the action verb from the comments is "*add*" and the action verbs of the method are "*add*" and "*put*". In the computer context, the "*add*" and "*put*" are the same word, if one of the words will be matched with the words from the class diagram. Likewise, few synonyms will be in the comment and the method signature.

We used the Stanford Part of Speech tagger as a pillar to develop this module. The purpose of this module is to tag all the words whether the words is a verb or a noun or etc. If it tags as a verb, this module adds the words to the word pool which is extracted from the identifiers of the class diagram. In the matching prowzbcess, these action word will be analyzed as the identifiers. If we take the comments to analyze, we will increase the relevancy and reliable.

Algorithm 3.1 Split the Identifier and SpellChecker

1: **function** Splitting and SpellChecking function

Input: *Source code identifier(x)*

2: *if x = = goodIdentifiers*

3: *String token = Split(x)*

4: *If token = = dictionary Word*

5: **return** *dictionary Word*

6: *else parse token into SpellChecker*

7: **return** *word*

8: *else* **return** *badIdentifiers*

9: **end function**

10: **function** Chunking the bad identifiers and Comparing

Input: *bad identiiers*

Input: *information from class diagram(y)*

Input: *Mi (initial marks)*

11: *chunking the bad identifiers using N-gram*

12: *check the chunk with SpellChecker*

13: *if chunk = = dictionary Word*

14: **return** *dictionary Word*

15: *remove the chunk*

16: *repeat the process*

17: *else increase the N by one*

18: *repeat the process*

19: *if y matched with dictionary Word*

20: $M = Mi + 100 / N$

21: *if marks >= 50%*

22: **return** class

19: **end function**

3.9 Matching and marks giving

After completing all the process in the algorithm, we will have two types of information. On the one hand, we will have the information which is extracted from the class diagram, and on the other hand, dictionary words which belong to all the identifiers and all the string token extracted from the identifiers. The problem is we need to identify the most relevant .java file through a comparing process of above mentioned two hands of the information. The developed module will take one word from the identifiers extracted from the .java files which are downloaded from the internet, and then compare with the information from the class diagram. If the two

words are matched, an amount of marks will be given to the .java file belongs to the compared words. Following formula is used to give the marks to the matched words.

$$M = Mi + (100 / N) \qquad (1)$$

In (1), M is denoted the marks is to be given for the .java files, Mi is denoted the initial marks for every loop iteration. N is denoted the number of identifiers extracted from the .java files which are downloaded from the internet and *100* for convert the marks into the percentage. Using the above equation, we calculate the marks for every matching word, after finishing all the calculation, if the mark is greater than 50%, the .java file will be selected to suggest as most relevant source code. Likewise, every .java file from the downloaded directory will be analyzed and suggested.

3.10 Summary of all module and process

As we discussed earlier, our developed framework starts with an XML file, all the software Architecture can be saved in the XML file format using the designing software tool. We focused only on the class diagram in this research. So if the developer needs to use our system they need to enter the class diagram in XML format. When we save a class diagram in the XML file format, it will consist of several lines of text, it will difficult to extract the information about the class diagram. We developed an XML parser module to parse the XML file and extract the information (class name, method name, and attribute name) from the file. As we discussed earlier, there are several forges, having a billions number of Open Source software repository such as SourceForge [1], GitHub [6], and GoogleCode [2]. We focused only on these three types of forges to download the projects. We need to download the projects from the forges depend on the information extracted from the class diagram. We developed three types of the crawler to fetch the projects from the three types of forges related to the information of the class diagram.

Depend on the information of the class diagram, our three types of the crawler will download a huge amount of related projects. The group of downloaded projects include the most relevant projects along with more irrelevant projects. In this research, we focused only on the Java projects. We developed two modules using the GitHub API to reduce the amount of the irrelevant projects. The first module is Java Class Name dumber, it dumbs all the .java file name from the GitHub forge, the second module is Java Repository Name dumber, it dumbs all the Java project name from the GitHub forge. These two modules will take the information from the class diagram one by one, and then compare with the Java Class Name list of the GitHub, if the class name matches with the information, it will retrieve the respective projects name from the Java Repository Name list. Finally, download the project from the GitHub forge. In this way, we can reduce the number of the irrelevant projects to be downloaded. Just we reduce the number of the irrelevant projects from GitHub, we cannot prevent completely. Also, a more irrelevant project will be downloaded from the SourceForge and the GoogleCode. So the crawler module downloads a huge amount of the project, next step is finding the most relevant projects from the group of the downloaded project.

We cannot check the relevancy in project level, we need to check the relevancy in the source code level. Our next module is the De-compressor, because, all the downloaded projects will be in the compressed format. We cannot do any operation on the projects as the projects in the compressed format. So we need to extract all the project from the compressed projects, our developed De-compressor module extracts all the downloaded project. After extracting all the projects, each project will include an amount of .java file. The next step is that we need to access all the identifiers from the Java source code to analyze the relevancy. For that purpose, we developed a module which represents all the source code into the Abstract Syntax Tree (AST), it will parse all the .java file into AST. And then, we can access all the identifiers (class name, method name, and attribute name). As we discussed earlier, extracted identifiers will

be in two types, one is the good identifier which follows any naming convention, and another one is the bad identifier which does not follow any naming convention. All the programmer are lazy, therefore, they do not use the full meaningful word. They shrink the words into the abbreviated term and connect more than two abbreviated terms to produce an identifier.

Identifying the meaningful word from the two types of the identifiers is the next big challenge. For the composite identifiers, we developed a splitter which splits the identifiers which are following any naming convention such as the camel case naming convention or the explicit separator (the symbol /, _, -, & and numbers 1-9). Our developed splitter will split the identifiers where the camel case or the explicit separator is started. After splitting the good identifiers, we may have an amount of the meaningful English words and the words will be sent to the comparing process, as well as we may get an amount of the string tokens. We developed the Spell-Checker to identify the meaningful words from the string tokens. It will identify the meaningful words from the string token using a dictionary. And then, we send the identified word to the comparing process. The next big challenge is identifying the meaningful words form the bad identifiers, developer shrinks the words into the abbreviated terms and then they connect two or more abbreviated term together to produce an identifier without following any naming convention. It is a very difficult process to identify the meaningful words from the bad identifiers.

We developed an algorithm to solve the above-mentioned problem by using the N-gram NLP technique. Refer the Algorithm 3.1 for further understanding. The algorithm will identify the meaningful words from the bad identifiers. Additionally, we developed a module using the Stanford Part of Speech tagger to analyze the comments in the Java source code. The module will identify the action verbs from the comments, and then it will send the action verb to the comparing process. Finally, we developed a module to compare the information extracted from the class diagram and words from the Java source code from the downloaded projects. If the word matches with the

information of the class diagram, an amount of marks will be given to the .java file. If the total mark is greater than 50%, the .java file will be selected as a relevant source code and suggested to the developers.

Chapter 4

Implementation

A major part of our research was implementation, a lot of the task needed to be implemented because this is a very big research and a lot of things wanted to test with the coding. We did analyze in source code level to identify the relevant source code. There are no any actual related research or system exist but some part of our research were related to a few amount of research but not fully related. The GitHub API integration took considerable time, because that was a new thing in Git forge. As well as we implemented the DTW technique and then we found some error and time cost, that is the reason, we shifted the implementation to the N-gram technique to solve the problem we faced in the DTW usage. We used some other APIs and software tool, all will be discussed in this chapter.

This chapter provides the details of the implementation of the developed module of our developed framework as well as several API usages and several tools used to develop our framework. Section 3.1 describes the programming language used to develop our framework and the reason to select the language, used IDE, and then implementation of XML parser and the API's module used to develop the parser. Section 3.2 describes the GitHub API integration and the challenges faced in the implementation. In section 3.3, we describes the implementation of three types of crawler we developed to download the projects from the internet depend on the information from the class diagram and the de-compressor to extract the downloaded projects. Section 3.4 describes the implementation of Abstract Syntax Tree. Section 3.5 explains the implementation of splitter for the composite identifiers to extract the meaningful words from the identifiers. In section 3.6, we describe the implementation of the spell-checker and the word-finder. Section 3.7 describes the implementation of the action verb Extractor from the comments. In section 3.8, we describe the

implementation of the final module of our developed framework that is the matching and rating module.

4.1 Implementation of XML Parser

We used Java programming language for all our implementation because we used a lot of Java existing APIs as well as Java programming language is a convenient language to develop the client server application. We developed three types of crawler using the JSOUP Java library. Usage of the JSOUP library makes the development of the crawlers easy. Even though the GitHub API available in several languages, a good documentations are available only for the Java API. We used the HTTPClient API to make the HTTP request, the implementation of the HTTP request was very easy and no any other configuration needed. We used Java programming language because of some other advantages than using other programming languages. We used the Eclipse as IDE (Integrated Development Environment), usually an application that combines the text editor, the debugger and other essential tools to enable developers to write applications. Eclipse IDE is "an open Source, robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools and rich client applications". Although the Eclipse Platform is written in Java programming language, it supports plug-ins that allow developers to develop and test the code written in other languages.

Our first implemented module is XML Parser as our suggested framework start with a Software Architecture (We focused only on the class diagram in our research) in the XML file format, our framework needs to read the XML file and then extract the information(class names, method names, and attribute names) from the file. It is important to understand how the XML stores and describes the information in it before to implement an application to read an XML file. The XML is a mark-up language and

it uses custom tags to describe the data it is storing. Tags can be nested, creating hierarchies to represent more complex data types.

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
        <book>
            <author>raja</author>
            <page>50</page>
           <color>black</color>
        </book>
        <book>
            <author>kamal</author>
           <page>100</page>
           <color>red</color>
        </book>
        <book>
           <author>kumar</author>
           <page>150</page>
           <color>green</color>
        </book>
</library>
```

Consider the above example of the XML code snippet, It can be seen that between opening and closing book tags (<book> and </book>) there are more tags storing more specific data. This means that a book entity has a few properties, each defined by its own tags. There can be multiple entities within the same file, as the example shows.

Likewise, if we convert a class diagram to an XML file, we will have a very big file including around thousand lines of code. It was difficult to handle that file. For that purpose, we developed our own XML Parser using the DOM (Document Object Model) parser [38], which is included in the javax.xml package. The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document [38]. The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure,

Figure 4.1 shows an example. It has two main classes such as, DocumentBuilder: Defines the API to obtain the DOM Document instances from an XML document. DocumwntBuilderFactory: Defines a factory API that enables the applications to obtain a parser that produces the DOM object trees from the XML documents. As well as we used the document interface, which represents the entire HTML or XML document. Conceptually, it is the root of the document tree and provides the primary access to the document's data. So by using these API, we developed an XML extractor to extract information from the XML file.

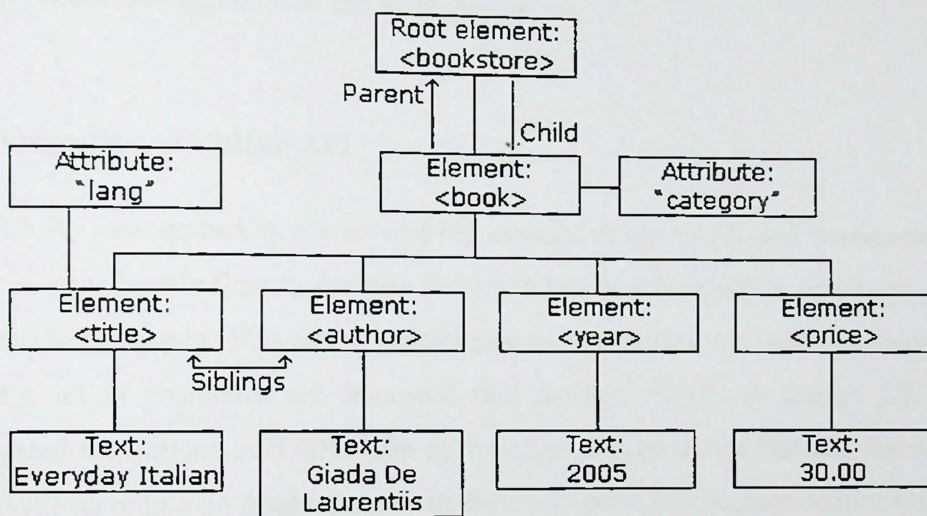


Figure 4.1: Example of XML Parser [20].

The code snippet below shows how the task of reading and accessing data from an XML file. Step-by-step explanation follows.

```
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
                    DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
                    Document doc = dBuilder.parse(XmlFile);
                    doc.getDocumentElement().normalize();
                    doc.getDocumentElement().getAttribute("name");
    FileWriter("ProjectName:"+doc.getDocumentElement().getAttribute("name"));
    if (eElement.getAttribute("modelType").equals("Class"))
```

```
eElement.getAttribute("name");
```

First of all, the dbFactory object is created of DocumentBuilderFactory class as described above. And then the dBuilder and doc objects of the DocumentBuilder and Document classes are created using the dbFactory object. Then we parse the XML file into the doc object to represent the XML file into a tree format. Finally, we check the modelType, if it is equal to the Class, that is the name of the class, if it is equal to Operation, then they are names of methods, as well as if the modelType is Attribute they are attributes. In this way, we access all the identifiers from the class diagram by parsing the class diagram into the XML parser.

4.2 Integration of GitHub API

GitHub API integration is our second big module in our developed framework. We spent a considerable time to develop this module using the GitHub API because this API is a new thing in GIT version control system. When we developed this module we faced a lot of problems, we discussed that problem deeply in section 2.2.1. We integrated the GitHub API with help of the support team of the GitHub, the support team will be online 24 hours to serve to the developers and the researchers. Once we ask a help via mailing list, they will respond and give a solution within few minutes. We published a research paper on this section you can refer the paper [39]. As we discussed earlier, this module has two sub-module, called Java project names dumber and Java class names dumber. Following two section will describe that two submodule's implementation deeply.

4.2.1 Implementation of Java project names dumber

When we use the GitHub API to access the repository names and the file names, the API will give that details in JSON format, so we need to use a JSON parser to extract

the information from the JSON object. The JavaScript Object Notation (JSON) is a text-based data interchange format, an alternative to the XML. JSON Object structure can be either as key-value pairs or ordered list of values. An example of JSON data is following:

```
{"Employees":[
{
"Name":"ABC",
"Designation":"Manager",
"Pay":"Rs. 60000/-",
"PhoneNumbers":[{
"LandLine" : "11-2xxxx99",
"Mobile" : "11xxxxxx11"
}]
},
{
"Name":"XYZ",
"Designation":"Sr. Manager",
"Pay":"Rs. 70000/-",
"PhoneNumbers":[{
"LandLine" : "11-2xxxx32",
"Mobile" : "66xxxxxx66"
}]
}
]}
```

In the JSON, [ ] represents Array. {} represents the Object. If we want to access the name list of employees, we use "Name" as the key to access its values. Likewise, in the GitHub respond for our request, all projects details are organized in the JSON format using the key-value pairs. As we discussed earlier, we want to have all the repository names (In our research we focused only on the Java repository names) in our local machine. And then we access all Java the file names belongs to that projects, which will be explained in next section. Following code snippet is used to access all the Java repository names in the GitHub. Step-by-step explanation follows.

```
String jsonString = callURL(URL);
JSONObject outerObject = new JSONObject(jsonString);
```

```
JSONArray jsonArray = outerObject.getJSONArray("items");
for (int i = 0, size = jsonArray.length(); i < size; i++)
        JSONObject objectInArray = jsonArray.getJSONObject(i);
        objectInArray.get("name")

public String callURL(String myURL) {
    StringBuilder sb = new StringBuilder();
            URLConnection urlConn = null;
    InputStreamReader in = null;
    URL url = new URL(myURL);
            urlConn = url.openConnection();
            if (urlConn != null)
                    urlConn.setReadTimeout(60 * 1000);
    if (urlConn != null && urlConn.getInputStream() !=
    null) {
            in = new
    InputStreamReader(urlConn.getInputStream(),
    Charset.defaultCharset());
    BufferedReader bufferedReader = new
    BufferedReader(in);
            if (bufferedReader != null) {
                    int cp;
                    while ((cp = bufferedReader.read()) != -1) {
                            sb.append((char) cp);
                    }
                    bufferedReader.close();
            }
            }
    in.close();
    return sb.toString();
    }
```

In the first line, we call the callURL(String myURL) method to parse an URL, an example of an URL is https://api.github.com/search/repositories?q=language:java+ created:. We will add few other details with this URL such that, the date after "created:" word in the URL, a particular time, the page number, and the number of objects per page. We parse the improved URL into the method as a parameter, and then the URLConnection and URL objects are created using the URL to make a request. Consider

the line if (urlConn != null), it means the API allow only 10 requests per minutes, so after the ten request connection will be null, that is what we stop the request for one minute using urlConn.setReadTimeout(60 * 1000); and after the one minute again we make another 10 request. Likewise, we make the request and take rest to get all repository names in the JSON data format, and then we will use a JSON parser to access that data.

A JSONObject is an unordered collection of name/value pairs. Its external form is a string wrapped in curly braces with colons between the names, the values, and the commas between the values and the names. An internal form is an object having to get () and opt () methods for accessing the values by the name and put () methods for adding or replacing the values by the name [40]. The values can be any of the following types: Boolean, JSONArray, JSONObject, Number, String, or the JSONObject.NULL object. A JSONObject constructor can be used to convert an external form of JSON text into an internal form whose values can be retrieved with the get () and opt () methods, to convert the values into a JSON text using the put () and toString () methods. A get () method returns a value if one can be found, and throws an exception if one cannot be found. And opt () method returns a default value instead of throwing an exception, and so is useful for obtaining the optional values [40].

A JSONArray is an ordered sequence of values. Its external text form is a string wrapped in square brackets with commas separating the values. The internal form is an object having get () and opt () methods for accessing the values by index, and opt () methods for adding or replacing values [41]. The values can be the following types: Boolean, JSONArray, JSONObject, Number, String, or the JSONObject.NULL object. The constructor can convert a JSON text into a Java object. The toString method do the operation vice versa. A get () method returns a value if one can be found, and throws an exception if one cannot be found. A opt () method returns a default value instead of throwing an exception, and so is useful for obtaining optional values [41].

Finally, through an iteration (loop), all the data such as the project names, the author's names, the created dates and few other details will be accessed from the JSON file. Likewise, all the project name will be dumbed using the above code snippet. As we discussed in the section 2.2.1, getting all the project names at once was very difficult because of the restriction of their API that is we could not access more than 1000 result. And then we used a neat trick to fetch more than 1000 results. We split up our search into segments, by the date when the repositories were created. For example, we first search for repositories that were created in the first week of October 2013, then second week, then September, and so on. Because we would be restricting the search to a narrow period, we will probably get less than 1000 results, and would therefore be able to get all of them. In case we notice that more than 1000 results are returned for a period, we had to narrow the period even more, so that we could collect all results.

4.2.1 Implementation of Java class names dumber

We had made a local repository containing all the Java project names in the GitHub forge using above discussed module (Java repository dumber). As we discussed earlier, each project could have hundreds of .java files, few project may have more than 100 .java files. The purpose of this module is read all project names one by one, and access all .java file names belongs to the particular project, and then make a local repository to hold all .java file names in the GitHub forge. When we take a keyword from the class diagram, first we will check with the java file name, if it matches and then we retrieve the project name from the project names repository, finally we download the project from the GitHub forge. Consider the following code snippet and explanation is following the code snippet.

```
String str1 = "https://api.github.com/search/Code?q=repo:";
String str2 = projectName;
String str3 = "+extension:java";
try {
```

```
String jsonString = callURL(str1+str2+str3);
        numCount++;
        if(( numCount % 10 ) == 0){
Thread.sleep(60000);
}
```

First three lines are describing three string to make a URL, string str2 hold the project names from the project names repository, and we connect these three string together to make a full URL. Finally we parse the URL into the callURL(), (String jsonString = callURL(str1+str2+str3);) method we discussed in the section 4.2.1. In the next line, we increase the count using numCount++; to check the number of requests is reached 10. If it reaches 10, we used a Java thread to take rest for one minute and again make the request. In this way the callURL() method will return a jsonString, we parse this string into the JSONObject (see the code snippet in section 4.2.1) and check the number of JSON objects to do pagination, following code will explain this problem.

```
int id = Integer.parseInt(outerObject1.get("total_count").toString());
                    if(id>100){
                    int pagenum = id/100;
                    for (int i = 1; i <= pagenum+1; i++)
        jsonString=callURL("https://api.github.com/search/Code?q=repo:"+line+"+exte
nsion:java&page="+i+"&per_page=100");
```

As we discussed earlier, only 100 objects will be displayed in a page, if the number of objects is more than 100, we need to make another page to access rest of the objects. That is what we are getting the id from the JSONObject and in the next line, we check number of objects, if it is more than 100, we divide the total number of objects by 100 to get the number of pages to be made. Next line of code describes a for-loop to make the number of pages depend on the divided value. Finally, we parse this value into the URL with starting "i" value from one and we increase the "i" value by one to reach all pages. Again we parse this URL into the callURL() method, get the jsonString, and again parse this string into the JSONObject to access all .java file name. If the id value is not more than 100 we parse the jsonString into the JSONObject. In this module and the

above module, we followed same code to do some another process, JSONObject is explained in the previous section, and if you want details about that code refer the previous section. In this way, we made a repository to hold all the .java file names in the GitHub forge. When you use the GitHub API, you will get so many problems, in that situation you can contact the support team of the GitHub, they will help to solve all the problem regarding the API usage.

4.3 Implementation of Crawler and Decompressor

We developed three types of the crawler to fetch the related projects to the class diagram depend on the information retrieved from the class diagram. We used the JSOUP Java API to implement this module. Our developed crawlers take the information (class name, method names, and attribute names) one by one as a keyword, take the forges URL as seed URL (www.Sourceforge.com, www.googleCode.com, and www.github.com) and it will fetch related projects through a few iteration. Refer the section 2.2.3 for further information about the crawler. These three types of crawlers are different because the related three types of forges and its websites are different. But we used same JSOUP library to develop these types of crawlers. Consider the following code and explanation.

```
Document doc = Jsoup.connect("https://Code.google.com/archive/p/oqs/");
                    Elements links1 = doc.select("p");
                    for (Element el : links1)
                                        System.out.println(el);
```

Java JSOUP library follows the Document Object Model (DOM) to fetch data, so we create doc object to hold all the data of the website. And then we parse the seed URL into the connect () method. The Elements object gets the data from doc object by giving the HTML tag into the select () method. Finally we will get all the data using an iteration. Following code snippet will illustrate the GoogleCode Crawler.

```
Document doc =
Jsoup.connect("https://Code.google.com/hosting/search?q="+term+"+label%3AJava
&start="+i);
```

We parse the GoogleCode search URL with the term string which is extracted from the class diagram. We parse all the term extracted from the class diagram one by one. Normally the Google code forge arrange all the project names in a table format. So we analyze the table value in the website to get the related project name.

```
for (Element tr : doc.select("#serp table tbody tr")) {
                    Element el = tr.child(0).child(0);
                    String projectName = el.attr("href").split("/")[29];
                    String SourceCodeUrl = "https://Code.google.com/p/" +
projectName + "/Source/browse/";
                    links.add(new String(SourceCodeUrl));

                    }
```

Then, we get all the element of the tables on the website, and then we get the child node of the tr element of the table, that will be a link. Finally we get the link using the attr("href") methods, and then split the link where the "/" symbol in the link to get the project names from the link and save the project name as link in a set using the 5th line of the code in the above-mentioned code snippet. Often the project name will be on the link, otherwise it will route another link to reach the project file. So we need to analyze the link using the same code again.

```
for(String link: zipLinks){
          Document doc  = Jsoup.connect(link);
           Elements linksOnPage = doc.select("a[href]");
                for (Element tr : linksOnPage) {
                    if(tr.absUrl("href").endsWith(".zip"))
                         links.add(tr.absUrl("href"));
                    if(tr.absUrl("href").contains("downloads")){
                         Document doc1  = Jsoup.connect(tr.absUrl("href"));
                         Elements linksOnPage1 = doc1.select("a[href]");
                         for(Element tr1 : linksOnPage1){
                              if(tr1.absUrl("href").endsWith(".zip")){
```

```
                links.add(tr1.absUrl("href"));
            }
        }
    }
}
```

We take each link one by one from the set we created using above link, and then we parse the link into crawler again to check the link route to the other link, if it routes to the other link we check the link whether it includes the ".zip" string using the tr.absUrl("href").endsWith(".zip") code snippet. If it contains the .zip word we add the link to another set which is the project file location. Otherwise, we check whether the link has "download" word, it means the link has project file location link. Again we check the link using same code and we add all link which has ".zip" string to the set. Finally, we download the zipped project file using HTTP request and HTTPGet method. Likewise, other two types of crawler will work, but it is implemented little bit different manner. But the basic idea is same.

As we discussed earlier, all the downloaded projects will be in zipped format. We cannot do any operation as the project file is in zipped format. Because our approach is based on source code level, so we need to extract these project automatically to get all .java file. For this purpose, we developed a De-compressor module to extract all downloaded project. But all downloaded projects will be in different zipping format such as .zip, .tar.gz, .tgz, .tar.bz2, .tar.bzip2, .tbz2, .tar.lzma, .tlzma, and .rar, .tar. We developed three classes to decompress all these types of file format such that tarDecompressor.java, zipDecompressor.java, and rarDecompressor.java. The .zip files will be decompressed by zipDecompressor.java, the .rar files will be decompressed by rarDecompressor.java, and rest of all other types of file format will be decompressed using tarDecompressor.java.

4.4 Abstract Syntax Tree and Identifier Splitter

Our proposed approach based on source code identifiers, we extract all compressed project using previous modules. All extracted projects will have a huge amount of .java files. Our next task was access all the identifiers (class name, method names, and attributes names) and the comments from these .java files. Because our developed approach based on analyzing the identifiers and check the relevance of the downloaded project to the class diagram. So only way to access the identifiers from the source code is to represent all the Java code into Abstract Syntax Tree.

4.4.1 Implementation of Abstract Syntax Tree

All source code can be represented in a tree format is called Abstract Syntax Tree. If you want detail about the AST you can refer the section 3.4. In this research, we used Eclipse JDT Java API to represent all Java source code into AST. The AST will represent the class name in the root node of the tree, the methods will be in the child nodes of the class name node, and the attributes and the method's parameters will be in the child nodes of the methods node. Following code snippet explain the AST representation and extract the identifiers.

```
ASTParser parser = ASTParser.newParser(AST.JLS3);
parser.setSource(str.toCharArray());
parser.setKind(ASTParser.K_COMPILATION_UNIT);
final CompilationUnit cu = (CompilationUnit) parser.createAST(null);
cu.accept(new ASTVisitor() {
        public boolean visit(VariableDeclarationFragment node) {
                SimpleName name = node.getName();
                names.add(name.getIdentifier().toString());
                return false; // do not continue

        }
        public boolean visit(MethodDeclaration node) {
                SimpleName name1 = node.getName();
            names.add(name1.getIdentifier().toString()+"()");
```

```
            return super.visit(node);
        }
        });
        return names;
         }
```

First three line of the code snippet explains, creating an ASTParser object, parsing the .java file path and making an Abstract Syntax Tree. If we represent the source code in a tree format, we can visit all the node of the tree using the ASTVisitor and access all the identifiers. Using the visit () Boolean type method we can access all the identifiers. If we want to retrieve all the attributes, we parse the VariableDeclarationFragment type node as a parameter to the visit () method, and then using the getIdentifier() method we can access the attributes. Likewise, if we want to access methods names, we parse MethodDeclaration type node as a parameter to the visit () method and using the same getIdentifier() method we can access the method names in the source code.

We access the leading comments of a method in the source code to extract the action verb from the comments. The coding of comments extraction from source code is somewhat different from the methods and attributes extraction. Consider the following code snippet and following explanation.

```
public CommentVisitor(CompilationUnit cu, String Source) {
        super();
        this.cu = cu;
        this.Source = Source;
}
public boolean visit(LineComment node) {
        int start = node.getStartPosition();
        int end = start + node.getLength();
        String comment = Source.substring(start, end);
        FileWriter(comment.replaceAll("\\s+", " ").replaceAll("/", ""));
        return true;
}
public boolean visit(BlockComment node) {
        int start = node.getStartPosition();
```

```
int end = start + node.getLength();
String comment = Source.substring(start, end);
FileWriter(comment.replaceAll("\\s+", " ").replaceAll("/*", ""));
return true;}
```

As we discussed earlier, if we retrieve the comments from the source code represented in AST, we create a CommentVisitor object and using the same visit () Boolean method, we retrieve the comments. Inside the method, we create two int type variables such as start and end to get the starting and the ending position of the comments. There are two types of comments, one is line comments which will start from a double forward slash. And another type is block comments which will start from a single slash and following * symbol. We used the Stanford Part of Speech tagger to identify whether the extracted word is an action verb or not.

4.4.2 Implementation of Identifier Splitter

The previous module will be used to extract all the identifiers (Class names, method names, and attribute names) from the Java source code. As we discussed in the section 3.5 all the developers are lazy, so they are not writing the full meaningful words. Instead of to use the full meaningful word, they are shrinking the words and then they are connecting these two or more abbreviated words together to create an identifier. If the programmers are ready to follow any naming convention when they producing composite identifier, normally they will follow Camel-Case naming convention (*rmvFle, dteRow, and sndMssg*) or they will use few explicit separator (*sleect_name, fnd$value, and get8rslt*). If they follow these types of naming convention to produce an identifiers, we need to split these identifiers to get the full meaningful words. This is what we developed this module using the following code snippet.

```
String str2="";
for (String w : identifiers.split("(?<!(^|[A-Z]))(?=[A-Z])|(?<!^)(?=[A-Z][a-z])"))
{ if (identifiers==w){
            str2=w;
```

```
}else
        splitedNames.add(w);
  }String str[]  = {"_","-","1","2","3","4","5","6","7","8","9","0"};
for (String s:str){
        if (str2.contains(s)){
                String st[]=StringUtils.split(str2, s, 0);
                for (String s1: st){
                splitedNames.add(s1);
                }}else
                unSplitedNames.add(str2);
}}
```

Most of the developers are following the Camel-Case naming convention (if they would like to follow any naming convention) and a few developers are using an explicit separator to make the composite identifiers. First, we check whether the identifiers are following the Camel-Case naming convention using the second line of code of the above code snippet, if it follows the Camel-Case naming convention we split the identifiers where the capital letter start, and store the term into a set using the splitedNames.add(w); code, else the word will be returned as it is. The second type of naming convention is using the explicit separator, the 7th line of the above code snippet is used to split the identifier using the explicit separator where the separator is started. Then the split term will be added to the set using the splitedNames.add(s1); code. Otherwise, the identifiers will be added to another new set using the unSplitedNames.add(str2); code.

4.5 Implementation of Spell Checker and Word Finder

We will have string terms and composite identifiers which are not followed any naming convention after using the above module. As we discussed earlier the identifiers can be divided into two types. One is Good identifiers (can be split using our splitter) and another one is Bad identifiers (cannot be split using our splitter). Our splitter will give the string term from the Good identifiers which should be checked

with a dictionary to identify the meaningful dictionary word. For spell checking purpose, we used Stanford Spell-Checker as a pillar to develop our own Spell-Checker. Following two section will describe the implementation of two sub-module to identify the meaningful dictionary word from Good identifiers and Bad identifiers.

4.5.1 Implementation of Spell Checker for Good Identifiers

As we discussed earlier, a Good identifier is a one which follows any naming convention such as Camel-Case or the explicit separator. Our splitter will split these types of identifiers into meaningful dictionary words and string tokens. If it is a meaningful dictionary word, that will be stored into the splChecked, otherwise, we use our spell checker module to get the meaningful dictionary word which is related to the string tokens. Consider the following code snippet and the following step by step explanation.

```
CamelSplitter cs = new CamelSplitter();
cs.idfyIdentifiers(clssPath);
SplitName = cs.splitedNames;
UnSplit = cs.unSplitedNames;
Set<String> splChecked = new HashSet<>();

for (String term1: SplitName){
        String term2 = term1.replaceAll("[0-
    9]","").replaceAll("[^\\w\\s]","").replaceAll("_", "").replaceAll("()", "");
        if(dictionary.contains(term2)){
                splChecked.add(term2);
        }else
        {
        KGramSpellingCorrector sp = new KGramSpellingCorrector();
        List<String> top10 = sp.corrections(term2.toLowerCase());
        for(String str: top10){
                splChecked.add(str);
        }}}
```

In the first line, we create a splitter object and we get two types of the identifiers using the cs.idfyIdentifiers(clssPath); and then we store these identifiers into two set SplitName and UnSplit. All string token and meaningful English dictionary words from good identifiers will be in the SplitName set. Our next task is identifying the meaningful dictionary word from the string token and send that word into comparing process using the if(dictionary.contains(term2)) code. If it is a string token we use our spell checker. Our developed spell checker using the N-gram technique to identify the meaningful English dictionary word. The spell checker will give 10 most relevant words to a string token. The 12th and the 13th line of the code are used to get the word list. Finally, we store these word into the splChecked set and send it to the comparing process to check the relevance with the information from the class diagram.

4.5.2 Spell Checker and Word Finder for Bad Identifiers

The previous module identifies the good identifiers and identify the meaningful English dictionary words from these identifiers. As well as it will make a set which holds all the bad identifiers which cannot be split by our developed splitter. However, we need to identify the meaningful English dictionary words from the bad identifiers as these identifiers include more than two words in the abbreviated form without following the naming convention. We developed an algorithm to do above-mentioned process. We used N-gram technique to implement the algorithm, as we mentioned earlier we used Dynamic time warping is a speech recognition technique to identify the meaningful words from the bad identifiers, but it took more time to identify the words and it suggested wrong word list for particular string token. You can refer the section 3.6.2 to get a full explanation of the developed algorithm. Following code snippet will explain the process of bad identifiers analyzing.

```
for(String word: UnSplit){
```

```
		word = word.replaceAll("[0-9]","").replaceAll("[^\\w\\s]","").replaceAll("_", "").replaceAll("()", "");
	for(int i=2; i<word.length(); i++){
	List<String> chunks = getChunks(word,i);
	for(String chunk:chunks){
		KGramSpellingCorrector sp = new KGramSpellingCorrector();
		List<String> top1 = sp.corrections(chunk.toLowerCase());
		for(String str: top1){
			splChecked.add(str);
		}}}}
```

We have stored all the bad identifiers into a set UnSplit. We get all the identifiers one by one using a loop and remove all the following from the identifiers, such as numbers, space, () (from method signature) and _ symbol. Then we split the identifiers into the chunk using the getChunks() method by parsing the identifiers and an integer (i) value which will start from 2 because one letter words are very rare in the computer programming field. This process will split the word into two letters chunk and check the chunk in the English dictionary using our developed spell checker. If the chunk is the meaningful English dictionary word, our spell checker will give 10 top related word list, and then we remove the chunk and continue the process. As well as we store the word list into splChecked and send the word set into the comparing process. If all the chunks are not an English dictionary word we will increase the (i) value by one and repeat the above-mentioned process. We will repeat the same process until identifying the related words to the identifiers. After identifying all the meaningful English dictionary word from the bad identifiers, we start the comparison process with the following module.

4.6 Implementation of Matching and Rating

This is our final developed module, to compare the identified words from the identifiers with the information from the class diagram. We take the words one by one from the identified word list and compare with the information of the class diagram. If the word matches with the information, we give an amount of marks to the downloaded

class and repeat the process to all the words. We used the WordNet [13] to get all synonyms word to a particular word. For an example, if the word is "delete", but the class diagram's information contains the word "remove", both words are same in the computer context (but both are synonyms in the English grammar). That is what we implemented the WordNet [13] to get all the synonyms and repeat the matching process. Following code snippet will explain the implementation of this module.

```
for (String elmntXMLCls : XMLClsDtlsLst) {
    for (String elmntSPL : splChecked) {
        String[] str_class = elmntXMLCls.split(",");
        for (String idntfr_in_class : str_class) {
            if (elmntSPL.equalsIgnoreCase(idntfr_in_class)) {
                marks += 100 / str_class.length;
            } else {
            WordNet wn = new WordNet();
            Set<String> synonyms = wn.wordNetSynonyms(elmntSPL);
            for (String syn : synonyms) {
                if (syn.equalsIgnoreCase(idntfr_in_class)) {
                    marks += 100 / str_class.length;
                }
            }}}}}
            System.out.println("Marks of the class is:"+ marks);
            if (marks >= 50) {
            System.out.println("class path");
            }
```

We used two loops to get the words and the class diagram's information one by one from the two set XMLClsDtlsLst and splChecked. And then, we check the both retrieved words by ignoring case. If these two words are matched, we divide the 100% by the number of identifiers in the class diagram and add the divided percentage value with initial marks (initial marks is zero) using the marks += 100/str_class.length; code statement. If both words are not matched, we get all the synonyms from the WordNet [13] using the WordNet wn = new WordNet (); and Set<String> synonyms = wn.wordNetSynonyms(elmntSPL); statement. To implement the WordNet [13], we used the JAWS Java API and its Data set. And then, again repeat the matching process

with all the retrieved synonyms from the WordNet [13]. Finally, we check the total marks of the class. If the total marks are more than 50% we suggest the class to the developers.

Chapter 5

Evaluation

We have done the experimental evaluation to show the effectiveness of our developed framework. Our research domain is source code searching and source code analyzing. Source code analyzing done by analyzing the identifiers of the source code (Names of classes, interfaces, methods or functions, variables and formal parameters or arguments can be viewed as a sequence of characters (a string) consisting of one or many tokens). We asked few question ourself to evaluate our developed System. We designed our evaluation to answer the following five research questions,

1) How well does our System download relevant projects?

2) How well does our System identify the real words from good identifiers?

3) How well does our System identify the real words from bad identifiers?

4) How accurate is the automatic extraction of the action verb from a comment?

The subjects in our evaluation were identifiers from 5 open Source Java programs across multiple domains and different developers from the GitHub repositories (for testing purpose we focused only on the Git forge). In total, the projects were comprised of 3985 lines of code from 34 Source files. In this dataset, there were 135 methods, with methods documented by the leading comments, and there were 309 attributes. Our goal of the evaluation was that the 5 targeted projects should be in the downloaded project group. As well as these projects wanted to be selected as the related projects to the class diagram through getting more than 50 percentage of marks by all the class belongs to the projects.

Our evaluation had several tasks and we checked our developed module with above-mentioned projects and its elements. We ran each of the four phases, (1) Downloading

the relevant project depends on the information from the class diagram, 2) Identify the words from the identifiers which are following naming convention, 3) Identify the words from the identifiers which are not following the naming convention, 4) Extracting the action verbs from the comments) on the entire set of 5 Java projects. Each phase's result was different, we could not show in one common result, that is what we explain the results of the four phase separately. Now we will discuss the result of each of the four phases.

This chapter provides the details of the approach we took to evaluate our developed framework. We explain the performance by asking the questions that we denoted previously. In section 5.1, we describe the performance of our developed crawlers (it means Downloading the relevant project depend on the information from the class diagram). Section 5.2 describes the performance of our Spell-Checker and Word-Finder for the Good identifiers. In section 5.3, we describe the performance of our Spell-Checker and Word-Finder for the Bad identifiers. Section 5.4 describes the performance of extracting the action verbs from the leading comments. Finally, section 3.5 explains the overall performance of our developed framework.

5.1 Performance of crawlers

To check the performance of all the module of our developed framework, we targeted 5 projects from the GitHub forge. For the evaluation, we selected only the GitHub crawler and the GitHub projects (aprendendo-vraptor, BST, BusMan, listview2, and PrintWB). First of all, we drew 5 class diagrams for all targeted 5 projects and we saved the diagrams in the XML file format. And then, we enter the class diagrams into our developed XMLExtractor module to extract the elements (class name, method names, and attribute names) of the class diagrams. Finally, we parse the extracted information as the keywords to our developed GitHub crawler. The crawler downloaded 734 projects including our targeted 5 projects. Our developed crawler

downloaded all our targeted projects, so the accuracy of our developed crawler for this test case was 100%. Through this evaluation we got the answer for our first research question *"How well does our System download relevant projects?"* we mentioned earlier.

Table 5.1: Result of downloading relevant projects

Targeted Projects names	**Keywords**	**Downloaded Projects name**	**The Project is there**
Vrapper	-Controller -CORS -Option ·Request	-ABR_controller -accountant-app -aprendendo-vraptor -vraptor multimodule	Yes
PrintWB	-Devicelist -Create -pairedlist -Intent	-IntentFilter -IcePointG -PrintWB	Yes
BusMan	-RiderMessagesTest - MimeType - ManifestActivity	-SSCPL -BusMan -droidel	Yes
BST	-node -BST	-bstrlin -AlgoDS -BST	Yes
listview2	-computer -mainActivity	-assembler-simulator -Code-dot-org -listview2	Yes

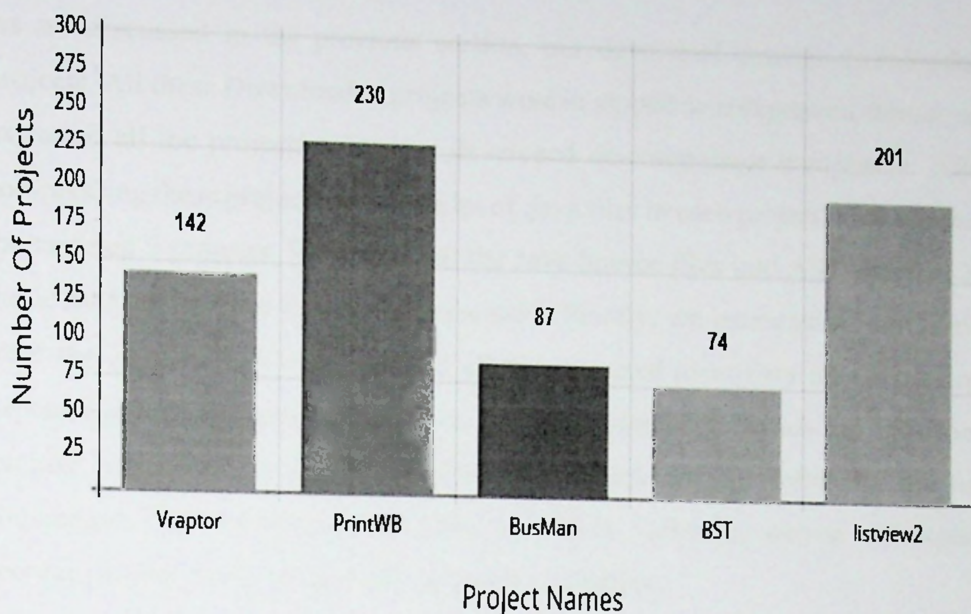


Figure 5.1: Result of downloading relevant projects

In Table 5.1 and Figure 5.1, we have shown the result of the crawler module in our developed framework for the evaluation test. If you take a look in Figure 5.1, the listview2 projects have only two .java file and few identifiers, but our crawler downloaded 201 (highest amount of projects) projects. Because the keywords given to the crawler is "computer" and "MainActivity", these words are commonly used words in the IT field, so the keywords occur in several projects. That is what, our developed crawler downloaded the highest amount of projects for the keywords. The crawlers downloaded 230 projects for the "PrintWB" project because it includes 15 .java file, so it includes a big number of identifiers. The .java files are more, so the identifiers are more, that is what the keywords are more. So the crawler downloaded that much of projects. However, the accuracy of our crawler is 100% because it downloaded the targeted 5 projects.

5.2 Performance of Spell Checker and word finder

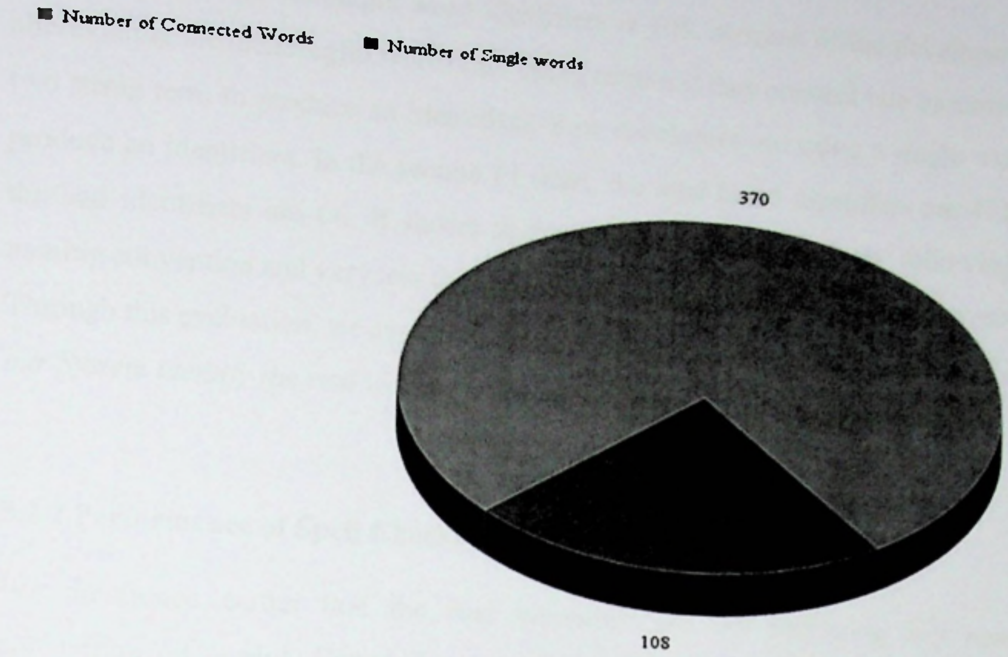
As we discussed in the previous section, our developed crawler downloaded 734 projects. All these Downloaded projects were in zipped or compressed format, and we extracted all the projects using our developed de-compressor component. After de-compressing these projects, we had a lot of .java files in each project. First we analyzed the targeted 5 projects. We parsed all the Java Source files into AST parser to access the identifiers and the comments separately. Finally, we extracted all the identifiers from the .java files. We categorized all the extracted identifiers into two categories depending on their nature, such that, the good identifiers which are following the naming convention, and the bad identifiers which are not following the naming convention. First, we consider the Good identifiers, following section will explain the accuracy of our Spell-checker for the good identifiers.

5.2.1 Performance of Spell Checker for Good Identifiers

Usage of Abstract Syntax Tree gave a big amount of identifiers. As we mentioned earlier, we categorized all these identifiers into two types, in this module we check the good identifiers which are following the naming convention. Altogether we extracted 478 good and bad identifiers. In which, 472 good identifiers and 06 bad identifiers, it shows that most of the developers are following the naming convention. And then we used our developed splitter to split the good identifiers where the naming convention occurs. Sometimes developers are abbreviating the words to produce the identifiers and sometimes not. If they have not abbreviated the word we can send the identified words into matching process. But if they have abbreviated the words to produce the identifiers we need to use our developed Spell-Checker to identify the meaningful English dictionary words.

Table 5.2: Sample result of Identifiers

Identifiers	**Connected Word sample**	**Single word sample**	**Sample Good Identifiers**	**Sample Bad Identifiers**
telefone	mModeLabel	Allowed	riderText	isAtivo
addToView()	writeTag()	Rider	newLocale	todoAtivos
allowed	isAtivo()	Size	btnSend	todos()
mModeLabel	BUILD_TYPE	bo	listView	timeregexp
				mywebview
				Htmlcontent()



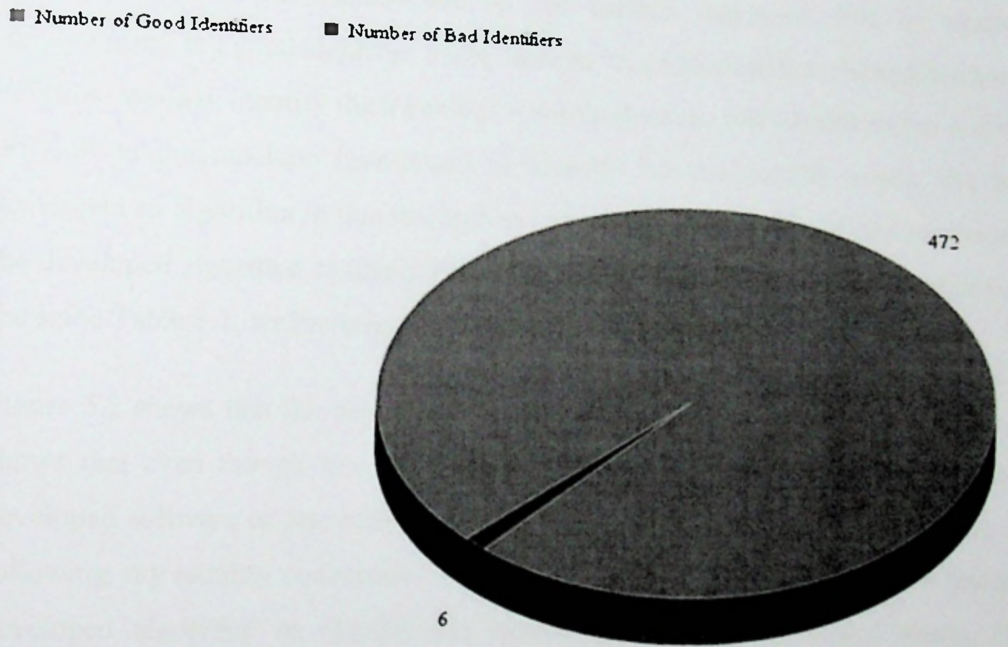


Figure 5.2: Result of Connected and Single Words in Identifiers

In Table 5.2, we have shown a sample of the composite identifiers and the good identifiers. In Figure 5.2, in the first PI chart, the number of composite identifiers are

370 and the number of single word identifiers is 108, so most of the developers are abbreviating the meaningful words into string term and they connect two or more than two string term to produce an identifiers. Few developers are using a single word to produce an identifiers. In the second PI chart, the total Good identifiers are 472 and the bad identifiers are 06. It shows that most of the developers are following the naming convention and very less developers are not following the naming convention. Through this evaluation, we answered to the second research question *"How well does our System identify the real words from good identifiers?"* we mentioned earlier.

5.2.2 Performance of Spell Checker for Good Identifiers

We mentioned earlier that the Bad identifiers are not following any naming convention; it means Camel-Case or the explicit separator. For an example, "addNumber" is a good identifier while "adnum" is a bad identifier, through the human intuition, we may identify the meaningful words from the bad identifiers but it is very difficult to the machine (computer) to identify the meaningful words. We have developed an algorithm in this research to overcome the problem. Therefore, we used the developed algorithm to identify the meaningful words from the bad identifiers. In the same Table 5.2, we have shown a sample of Bad identifiers.

Figure 5.2 shows that the percentage of all 6 bad identifiers out of all identifiers. It shows that even though few developers do not follow any naming convention in developed software or currently developing software but most of the developers are following any naming convention. Finally, we parsed all the bad identifiers into our developed algorithm to identify the meaningful English dictionary words. Our algorithm uses the N-gram technique, and the WordNet to identify the meaningful English words. Table 5.3 shows that a sample of the result of identifying meaningful English words from the good and bad identifiers. Through this evaluation, we

answered to the third research question *"How well does our System identify the real words from bad identifiers?"* we mentioned earlier.

5.3 Performance of Extraction of action verb from comments

If we need to understand a class or method, we can read the each line of codes and realize the logic of the methods or task of the methods. Otherwise, we can read the leading comments of the methods or the class to understand the task. Comments will explain the whole task of a class or a method. Most of the methods are doing an action to complete a task, so it includes an action verb. Add, remove, call, connect, delete, send, and show are the example of the action verbs. Likewise, all the leading comments of methods are having an action verbs in it (sometimes, it will be a synonyms of the action verbs of the methods), so our assumption is that we can get the relevant words to the task of the methods through analyzing the comments that is the action verbs. The action verbs will describe the task of methods, so we can use these action verbs to the comparing and rating process.

So we extracted all the leading comments using the AST parser, then we extracted all the action verb from the comments using the Stanford Part of Speech Tagger. And then analyzed the action verbs as the identifiers with the information of the class diagram. We accessed 34 .java files from all the targeted 5 projects, we extracted all the leading comments of the methods from these .java files, and finally, we extracted 126 action words from the extracted comments. We parse these action verbs as the identifiers to compare with the information of the class diagram. Most probably the action verbs from the comments were synonyms of the action verbs used in the method signature or directly to the methods name.

UNIVERSITY OF MORATUWA LIBRARY

Table 5.3: Sample result of splitting and identifying the real words

Identifiers	Split terms	non split terms	Our algorithm
addToView()	Add, To, view		
rider			rider
writeTag()	Write, Tag		
BUILD_TYPE	BULID, TYPE		
Mywebview		Mywebview	my, we, web, via, view
phraseLocale	Phrase, Locale		
lstview		lstview	last, list, lost, stove, via
recordNewRider	ReCode, New, Rider		

Table 5.4: Sample result of rating the project

Projects	**Targeted Projects**	**Average marks of all .java files of the projects (threshold marks 50%)**	**Status Of selection**
aprendendo-vraptor	Yes	100%	Selected
DrivingController	No	47%	Rejected
eboard -app	No	23%	Rejected
BusMan	Yes	100%	Selected
PrintWB	Yes	100%	Selected
jersey-header-cache	No	24%	Rejected

Chapter 6

Conclusions

6.1 Conclusion

The whole work of this research does not exist as a related work or the relevant Software tool, but part of our developed framework are available with so many drawbacks but not full work that we have done. First of all this thesis give a brief introduction of the current situation in the software development company as well as the common problems that the developers are facing when they launch the implementation of a software architecture. Then it presents the analyzing of the all related work with our work, in that section we described the GitHub API integration to develop our framework, which is a new thing providing by the GitHub. As well as we discussed the crawlers, the JSOUP java API, the WordNet, the Stanford Spell Checker, and the Stanford Part of Speech Tagger.

Next, we described the design of our framework, in that section we describe all the developed module to develop our framework. All together we have developed ten module, such as XML Parser, Java project names dumber, Java class names dumber, Crawler and De-compressor, Abstract Syntax Tree, Identifier Splitter, Spell Checker for good identifiers, Spell Checker and Word Finder for bad identifiers, Action verb Extractor, and Matching and rating module. We discussed the purpose of all modules and what are the tools and techniques used as pillars to develop the modules. We used w3c DOM parser, GitHub API, JSOUP API, JUNRAR API, LZMA API, ECLIPSE JDT API, and JAWS API as pillars to develop this framework.

The implementation part was a very important part in our research because we had several implemented module, each module depends on another module through an output of a module was an input to another module. Likewise, if and only if we implement the one module we can use another module. We used the Java language to

implement our framework because most of our planned pillars were as Java API. As well as we developed a client-server application, we can easily do the web application's work in Java so using the HTTP request and respond. In beginning, we used Dynamic Time Warping to split the identifier for the bad identifiers. But we faced two types of bug, one was the highest time cost to identify the meaningful English words from the bad identifiers than our current approach. And another one, the DTW suggest a wrong word list for a particular identifier. So we shifted to the N-gram algorithm instead of the DTW algorithm, we avoided these problem faced in DTW algorithm usage through the usage of the N-gram algorithm. Even though the usage of the WordNet to get synonyms is taking sometime but if we compare with the DTW, the time is less.

As we discussed earlier, the GitHub API is a new component in the GIT forge, using GitHub API, we can do all GIT activity, we can have all the repository names classified by all the programming language, and we can have all the name of the program Source file. We integrated the GitHub API to having all the repository names and all the .java file names. But the integration of the GitHub API was very difficult because that was a new component in the GIT forge. The GitHub maintain a support team to help the developers and the researchers when the API integration. They know that the developers and researchers will face a lot of problems when they try to integrate the GitHub API. In which situation, they can make a request for a help from the support teams, at the moment they will respond to the request. 24 hours the support team will be ready to help.

The Difficulties and the challenges were very high when we analyzed the identifiers. As we discussed earlier all the developers are lazy, that is what they do not use the full English words to produce the identifiers, instead of using full word, they shrink the words to create the acronym. They use more than two abbreviated terms to produce an identifier. In that situation, if they followed any naming convention we can identify the meaningful words from the identifiers. Otherwise, it will be a very difficult process

to identify the meaningful English words from the identifiers. Sometimes, they shrink the words very badly, for an example they shrink the word "length" into "ln", as a human it will difficult to identify the words from the token, but to the machine it is impossible. That is what we developed an algorithm to overcome the difficulties.

Although it was a challenging process we have developed a framework to address the research problem that we discussed in the introduction section. Our developed framework has higher accuracy in all phases as we discussed earlier: 100% accuracy in downloading the related projects depends on the class diagram information with some other additional projects, as given in the Table 5.1, 734 projects was downloaded, even though 729 was unwanted projects 5 targeted projects was downloaded. However, these 729 projects were related in any of instant of the information given to download these projects from the class diagram. And finally, we took all the targeted 5 projects. From the 5 projects (vraptor contains 15 source code files, PrintWB contains 8 source code files, BusMan contains 6 source code files, listview2 contains 2 source code files, and BST contains 3 source code files) we extracted 34 .java files, and all the source code files together contain 3985 lines of source codes. We got 478 identifiers from these .java files in which 472 good identifiers and 6 bad identifiers, from the Good identifiers, totally there were 679 split terms, 131 meaningful Dictionary words, and 548 unknown terms.

Our developed SpellChecker suggested 2103 related word for 448 split terms out of 548 unknown terms. Our developed SpellChecker could not identify the meaningful English words for 100 unknown terms out of 548 unknown terms. So we treated the 100 unknown terms as bad identifiers, already we were having 6 bad identifiers so totally there were 106 bad identifiers. The developed tool could not identify approximately 6% of the good identifiers, which were abbreviated in very bad manner. We therefore, claim the tool accuracy for this level to be 94% in identifying the meaningful words from good identifiers. Our proposed algorithm suggested 304 related words for 106 bad identifiers. Our Algorithm identified only ~87% of

meaningful words from the 106 bad identifiers, because of the unknown terms are so badly abbreviated, and did not follow the Naming Convention. However, it failed to identify about 13% of the bad identifiers. However, through the overall analyzing and matching process all the .java files of the targeted 5 projects were suggested as relevant .java files. So matching and rating process module had 100% accuracy.

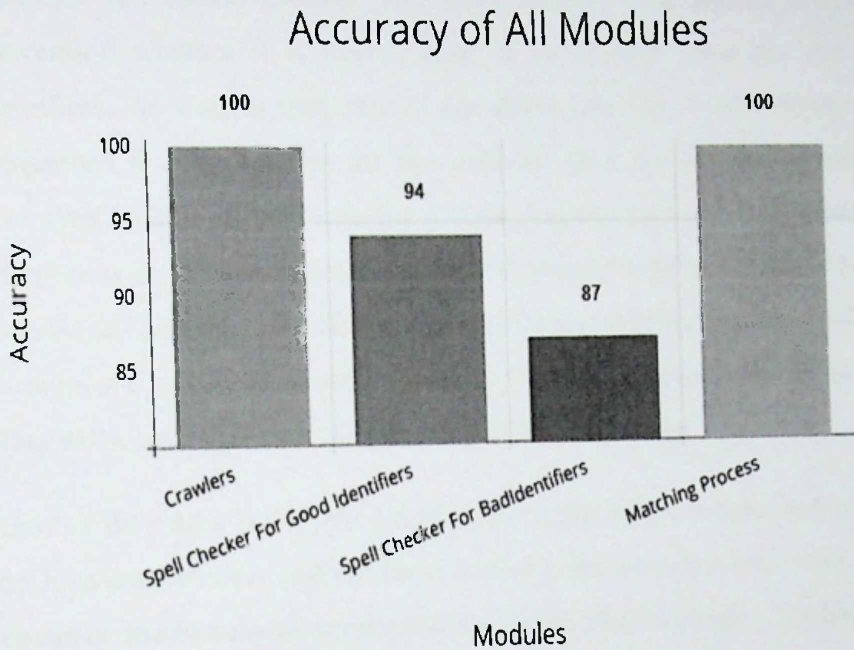


Figure 6.1: Accuracy of all Modules

6.2 Limitation and Future Work

We have few limitations in this developed framework. The section 6.2.1 describes these limitations and we have a plan to extend our study and work. The section 6.2.2 describes our future work.

6.2.1 Limitation of our work

In this research, we focused only on the class diagram as the Software Architecture. Our developed framework starts the process from a class diagram in the XML format, so the developer needs to enter the class diagram in the XML format into our framework. When the designer develops a class diagram and give a name into an identifier (attributes, method, and class names) they should follow any naming convention whether it is Camel case or an explicit separator for the composite identifiers. As well as they should not abbreviate the words. These are the current limitations to the designer for the software tool we developed and this must be improved in future. After entering a class diagram into our framework, it will do the all process itself automatically. As we discussed earlier, in this research, we have focused only on the class diagram as a software architecture, and we have completed all the work successfully using the class diagram, so we can do the same work for all other software architecture which is in XML format file.

Another limitation is that we download only the Java projects, and we analyzed only the Java source code, and we have done the above-mentioned work successfully, so we can do the same work for all other programming languages. Another very important limitation is that the size of the project file is to be downloaded into 50 megabytes. So downloading the projects which have the file size more than 50 megabytes is prevented. We check the file size in the respond to the HTTP request, if the file size is more than 50 megabytes, then we avoid the project. Because this research work targeted only on the small and the medium software companies, so very big projects are not suitable for them. As well as we limit the number of projects to be downloaded into 100 by our crawlers. It means when the crawler downloading the projects we count the number of projects if it reaches to 100, we stop the crawler, because already our crawler has downloaded the targeted projects within 100 projects. So it will download all the most relevant project within 100 projects.

6.2.2 Future work

In future development, we planned to include automatic Design pattern suggestion with our developed framework. When a software architecture gets into the development process, the developer needs to search relevant sample code or libraries, for that, we developed a framework which is capable of doing the process automatically; an extension feature would be to support the developers by giving them help to select a proper design pattern. If the software company has one expert in a particular context, the expert person will suggest the proper design pattern but we focus only on the small and the medium software company. They cannot maintain like that expert people because if they maintain like that people, they have to pay more money as salary for them. That is what they want a help to get a proper design pattern. So we plan to extend our framework to automatically select a proper design pattern in a particular situation. For this selection process, we planned to use the Case-Based Reasoning (CBR) is one of the most successful applied Artificial Intelligence technologies of recent years and the act of the developing solutions to unsolved problems based on pre-existing solutions of a similar nature.

Reference

[1]. Sourceforge web site, https://Sourceforge.net/about. [Online accessed 10-june-2015], 2015.

[2]. Google Code web site, https://Code.google.com/, [Online accessed 23-januarry-2016], 2016.

[3]. Krugle Source Code search engine home web page, http://www.krugle.com/, [Online accessed 14-April-2017], 2017.

[4]. Koders Source Code search engine web site home page, http://Code.openhub.net/, [Online accessed 14-April-2017], 2017.

[5]. Codase Source Code search engine web site home page, http://www.codase.com/, [Online accessed 14-April-2017], 2017.

[6]. GitHub web site home page, https://github.com/, [Online accessed 14-April-2017], 2017.

[7]. Nioosha Madani, Latifa Guerrouj,"Recognizing Words from Source Code Identifiers using Speech Recognition Techniques," presented at the CSMR 2010 14th European Conference at Madrid, Spain, March 2010.

[8]. Google Code search engine definition and its purpose, [online accessed 18-April-2017], https://www.revolvy.com/main/index.php?s=Google%20Code%20 Search, 2017.

[9]. Ohloh, krugle, definition, http://www.makeuseof.com/tag/open-Source-matters-6-Source-Code-search-engines-you-can-use-for-programming-projects /, [Online accessed, 18-April-2017], 2017.

[10]. Kalpana B. Khandale,Ajitkumar Pundage,C. Namrata Mahender, "Similarities in words Using Different Pos Taggers", International Conference On Recent Advances In Computer Science, Engineering And Technology, 25 June, 2016.

[11]. Apostolos Kritikos, George Kakarontzas and Ioannis Stamelos "A semi-automated process for open Source Code reuse", presented at 15th International Conference, ICSR-2016, on June 5-7, 2016.

[12]. Otavio Augusto Lazzarini Lemos, Sushil Bajracharya, Joel Ossher, Paulo Cesar Masiero and Cristina Lopes , "Applying Test-driven Code Search to the Reuse

of Auxiliary Functionality", presented at the 2009 ACM symposium on Applied Computing on 08 March 2009.

[13]. Keith Brown, "WordNet", published in the Encyclopedia of Language & Linguistics, Second Edition, published by Elsevier, 2006.

[14]. Cristina V. Lopes, Donald Bren, A. L. Lemos, Adriano C. de, Paula, Felipe C. Zanichelli, "Thesaurus-Based Automatic Query Expansion for Interface-Driven Code Search" published at 11th Working Conference on Mining Software Repositories, Hyderabad, India, on May 31 – June 1 2014.

[15]. Steven P. Reiss, Department of Computer Science, Brown University, "Specifying What to Search for", Presented at 2009. SUITE '09, ICSE Workshop on 16 May 2009.

[16]. Adrian Kuhn, Florian S. Gysin, "A Trustability Metric for Code Search based on Developer Karma", presented at 2010 ICSE Workshop on Search-driven Development on May 01 2010.

[17]. Sushil K Bajracharya, Joel Ossher, and Cristina V Lopes, "Leveraging Usage Similarity for Effective Retrieval of Examples in Code Repositories". Presented at ACM SIGSOFT international symposium on Foundations of software engineering, University of California, Irvine, CA, USA November 2010.

[18]. Git and GitHub definition and related informations, https://www.howtogeek.com /180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/ [Online accessed, 05-Januarry-2017], 2017.

[19]. Search API of the GitHub, https://developer.github.com/v3/search/, [Online accessed, 23-june-2016], 2016.

[20]. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze, "An Introduction to Information Retrieval", 2009 Cambridge University Press, Printed on April 1, 2009.

[21]. Crawler definition http://searchmicroservices.techtarget.com/definition/ crawler, [Online accessed, 13-Januarry], 2017.

[22]. JSOUP Java API, https://jsoup.org/apidocs, [Online, accessed 12-jan-2016], 2016.

[23]. N-gram definition, http://text-analytics101.rxnlp.com /2014/11/what-are-n-grams.html, [Online Accessed, 25-Januarry], 2017.

[24]. William B. Cavnar and John M. Trenkle, "N-Gram-Based Text Categorization", presented at In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 1994.

[25]. Part-of-Speech Tagging, https://nlp.stanford.edu/software/tagger.shtml, [Online accessed, 23-February-2017], 2017.

[26]. Part-of-Speech Tagger, http://language.worldofcomputing.net/postagging/ parts-of-speech-tagging.html, [Online accessed, 23-February-2017], 2017.

[27]. Matthew J. Howard, Samir Gupta, Lori Pollock, and K. Vijay-Shanker, "Automatically Mining Software-Based, Semantically-Similar Words from Comment-Code Mappings", published on 10th Working Conference on Mining Software Repositories, 2013 May 18.

[28]. Emily Hill, Lori Pollock and K. Vijay-Shanker, "Automatically Capturing Source Code Contaxt of NL-Queries for Software Maintanence and Reuse" published on Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference, May 2009.

[29]. Madhuri R. Marri, Suresh Thummalapenta, Tao Xie, "Improving Software Quality via Code Searching and Mining", Presented at 2009. SUITE '09, ICSE Workshop on 16 May 2009.

[30]. Tao Xie, Suresh Thummalapenta, "PARSEWeb: A Programmer Assistant for Reusing Open Source Code on the Web", presented at twenty-second IEEE/ACM international conference November 2007.

[31]. Miltiadis Allamanis, Christian Bird, Charles Sutton, Redmond, "Learning Natural Coding Conventions", published at 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 11 November 2014.

[32]. XML Parser, https://www.tutorialspoint.com/java_xml/java_xml_parsers.htm, [Online, accessed 10-june-2015], 2015.

[33]. Abstract Syntax Tree, http://wiki.c2.com/?AbstractSyntaxTree, [Online accessed, 03-March-2017], 2017.

[34]. Samir Gupta, Sana Malik, Lori Pollock, "Part-of- Speech Tagging of Program Identifiers for Improved Text-based Software Engineering Tools", presented Program Comprehension (ICPC), 2013 IEEE 21st International Conference on, 21 May 2013.

[35]. Ashish Sureka, "Source Code Identifier Splitting Using Yahoo Image and Web Search Engine", presented at KDD Knowledge Discovery and Data Mining, August 12, 2012, Beijing, China.

[36]. H. Ney, "The use of a one-stage dynamic programming algorithm for connected word recognition," presented at Acoustics, Speech and Signal Processing, IEEE Transactions on, vol. 32, no. 2, pp. 263–271, Apr 1984.

[37]. V. I. Levenshtein, "Binary Codes capable of correcting deletions, insertions, and reversals," Cybernetics and Control Theory, no. 10, pp. 707–710, 1966.

[38]. Document Object Model, https://www.w3.org/DOM/, [Online accessed, 5-March-2017], 2017.

[39]. P.Pirapuraj, Dr.Indika Perera, "GITHUB Application Program Interface and WordNet for Code reuse", presented at Fifth Annual Science Research Session-2016, South Eastern University of Sri Lanka.

[40]. Class JSONObject, https://docs.oracle.com/middleware/maf240/mobile/api-ref/oracle/adfmf/json/JSONObject.html, [Online accessed, 10-March-2017], 2017.

[41]. Class JSONArray, http://docs.oracle.com /middleware/maf222/mobile/api-ref/oracle/adfmf/json/JSONArray.html, [Online accessed, 10-March-2017], 2017.

[42]. Giriprasad Sridhara, Emily Hill, Lori Pollock and K. Vijay-Shanker, "Identifying Word Relations in Software:A Comparative Study of Semantic Similarity Tools," presented at the 16th IEEE International Conference, Amsterdam, Netherlands, June 2008.

[43]. Lars Heinemann, Prof. Dr. Dr. h.c. Manfred Broy Prof. Martin Robillard, McGill University, Montréal, Kanada, "Effective and Efficient Reuse with Software Libraries" presented at the 20th ACM SIGSOFT International Symposium on Foundations of Software Engineering Jully 2012.

[44]. D. Lawrie, C. Morrel, H. Feild, and D. Binkley, "What's in a name? A study of identifiers," in Proc. of the International Conference on Program Comprehension (ICPC), 2006, pp. 3–12.

[45]. Steven P. Reiss Department of Computer Science, Brown University, "Semantics-Based Code Search", presented at ICSE '09 Proceedings of the 31st International Conference on Software Engineering, may 2009.

[46]. Taweesup Apiwattanapong, Alessandro Orso, Mary Jean Harrold, "A Differencing Algorithm for Object-Oriented Programs", presented at 19th IEEE international conference on Automated software engineering, IEEE Computer Society Washington, 2004-09-20.

[47]. Jens Krinke, "Identifying Similar Code with Program Dependence Graphs", presented at Eighth Working Conference on Reverse Engineering (WCRE'01), IEEE Computer Society Washington 2001-10-02.

[48]. Martin P. Robillard, "Automatic Generation of Suggestions for Program Investigation", presented at 13th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, USA 2005-09-05.

[49]. Adrian Kuhn, "Automatic Labeling of Software Components and their Evolution using Log-Likelihood Ratio of Word Frequencies in Source Code", presented at 09 Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, Washington, DC, USA 2009-05-16.

[50]. Ashish Sureka, "Source Code Identifier Splitting Using Yahoo Image and Web Search Engine", presented at First International Workshop on Software Mining, New York, NY, USA 2012-08-12.

[51]. Oliver Hummel, Werner Janjic, Colin Atkinson, "Proposing Software Design Recommendations Based on Component Interface Intersecting", presented at 2nd International Workshop on Recommendation Systems for Software Engineering, 2010-05-04.

[52]. Abstract Syntaxt Tree, https://en.wikipedia.org/wiki/Abstract_syntax_tree, {Online, accessed 12-March-2016], 2016.

[53]. Simon Butler, Michel Wermelinger, Yijun Yu, and Helen Sharp, “Improving the tokenization of identifier names”, presented at European Conference on Object-Oriented Programming, volume 6813 of Lecture Notes in Computer Science, pages 130–154. Springer Verlag, 2011.

[54]. Girish Maskeri, Santonu Sarkar, Kenneth Heafield, "Mining Business Topics in Source Code using Latent Dirichlet Allocation", 08 Proceedings of the 1st India software engineering conference 2008-02-19.

[55]. Florian S. Gysin, "Improved Social Trustability of Code Search Results", presented at 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, 2010-05-01.

[56]. Sushil Bajracharya, Trung Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, Cristina Lopes, "Sourcerer: A Search Engine for Open Source Code

Supporting Structure-Based Search", presented at 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, 2006-10-22.

[57]. Paul W. McBurney, Collin McMillan, "Automatic Documentation Generation via Source Code Summarization of Method Context" presented at 22nd International Conference on Program Comprehension, 2014-06-02.

[58]. Github API, https://developer.github.com/v3, [Online, accessed 21-may-2016], 2016.

[59].GitHub searching API API,https://api.github.com/search/repositories?q=tetris+language:assembly&sort=stars&order=desc, [Online, accessed 27-may-2016], 2016.

[60]. Eric Enslen, Emily Hill, Lori Pollock, Lori Pollock,Lori Pollock, "Mining Source Code to Automatically Split Identifiers for Software Analysis", presented at 6th IEEE International Working Conference on Mining Software Repositories, 2009-05-16.

[61]. Oleksandr Panchenko, "Hybrid Storage for Enabling Fully-Featured Text Search and Fine-Grained Structural Search over Source Code", presented at Search-Driven Development-Users, Infrastructure, Tools and Evaluation, 2009. SUITE '09. ICSE Workshop,Vancouver, BC, Canada, 16 May 2009.

[62]. Steven P. Reiss, "Seeking the User Interface", presented at 29th ACM/IEEE international conference on Automated software engineering, 2014-09-15.

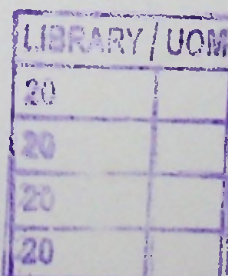
[63]. Version control systems and its related informations, https://www .smashingmagazine.com/2008/09/the-top-7-open-Source-version-control-systems/, [Online accessed, 11-Januarry-2017], 2017.

[64]. Repository in GitHub. https://github.com/new, [Online accessed 13-March-2016], 2016.

[65]. Taggest list of Part of Speech tagger. http://gitqwerty777.github.io/img/NLP /tagset.png [Online accessed 20-January], 2016

[66]. Online version of Stanford Spell Checker. http://www.spellcheck.net/how-do-you-spell/stanford, [Online accessed 02-January], 2016.

[67]. http://fostnope.files.wordpress.com/2011/12/121911_2241_agilearchit1.jpg ?w=538,http://images.clipartpanda.com/server-clipart-RTGK5rgTL.png.


LIBRARY / UOM
20
20
20
20

UNIVERSITY OF MORATUWA - SRI LANKA
RECEIVED
09 FEB 2018
AQ - SECTION