

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



TH3497

004 "IT"
004 (013)

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

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