Analyzing Source Code Identifiers for Code Reuse using NLP Techniques and WordNet

P. Pirapuraj, Indika Perera
Department of Computer Science and Engineering
University of Moratuwa, Sri Lanka
pirapu@cse.mrt.ac.lk, indika@cse.mrt.ac.lk

Abstract— **Massive amount of source codes are available free and open. Reusing those open source codes in projects can reduce the project duration and cost. Even though several Code Search Engines (CSE) are available, finding the most relevant code can be challenging. In this paper we propose a framework that can be used to overcome the above said challenge. The proposed solution starts with a Software Architecture (Class Diagram) in XML format and extracts information from the XML file, and then, it fetches relevant projects using three types of crawlers from GitHub, SourceForge, and GoogleCode. Then it finds the most relevant projects among the vast amount of downloaded projects. This research considers only Java projects. All java files in every project will be represented in Abstract Syntax Tree (AST) to extract identifiers (class names, method names, and attributes name) and comments. Action words (verbs) are extracted from comments using Part of Speech technique (POS). Those identifiers and XML file information need to be analyzed for matching. If identifiers are matched, marks will be given to those identifiers, likewise marks will be added together and then if the total mark is greater than 50%, the .java file will be considered as a relevant code. Otherwise, WordNet will be used to get synonym of those identifiers and repeat the matching process using those synonyms. For connected word identifiers, camel case splitter and N-gram technique are used to separate those words. The Stanford Spellchecker is used to identify abbreviated words. The results indicate successful identification of relevant source codes.**

Keywords— Software Architecture; Class Diagram; WordNet; N-gram technique; PoS Tagging; Sourcecode identification; Code reuse

I. INTRODUCTION

As Software Source codes continue to grow and evolve, identifying relevant code to particular task within millions of lines of code becomes increasingly difficult [1]. When performing software reuse tasks, developers must first identify the relevant code fragments to be reused. To identify code relevant to the task developers typically use several code search engines (CSE) such as Google code search, Krugle, Koders, Sourcerer, and Codase. In this process, the developer enters a query into one of CSEs [2]. Depending on the relevance of the results, the user will reformulate the query and search again. This process continues until the user is satisfied with the results. In this process, the user has two important tasks: (1) query formulation and (2) determining whether the search results are relevant [3, 4].

This new development culture led to millions of free lines of code that transformed the WWW into a large pool of reusable code, which are organized as large code repositories called forges (SourceForge, Google Code, Git, etc.) [5]. For example, SourceForge hosts about 179,518 projects with two million registered users and a large number of anonymous users [5, 2]. The CSEs accept queries such as the names of classes or methods of Application Programming Interfaces (API) and search in CVS or SVN repositories of available open source projects [2], then if the organization has any expert in particular area, they will identify the most relevant source code to reuse.

Experienced workers are very important to companies, in the sense that someone with experience has higher productivity [3]. Many Software engineering research shows that Source code reuse positively affects the competitiveness of an organization: through this reuse process the productivity of the development team is increased, and the overall quality of the resulting software is improved. It shows that many of today's software systems consist of a considerable fraction of code reused from software libraries [1]. Therefore, code reuse is very important task in software development.

Keywords based code searching is the most efficient method among several code searching method as well as it is giving most desirable and most relevant source code to reuse [5]. However, the keywords based searching can give relevant result along with irrelevant results. The main challenge is to identify the most relevant result. The best way to address the challenge is analyzing source code identifiers with class diagram information for relevance. 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)[6]. Several NLP techniques and Machine Learning techniques have been used to analyze identifiers. Nioosha Madani, and Latifa Guerrouj et al. [3] used a Speech Recognition Technique to recognize words from source code identifiers. They used Dynamic time warping (DTW) Speech Recognition Technique to recognize words from Identifiers. We face some challenges when we use DTW, such that (1) it took more time to identify the real words of term from source code identifier and (2) Some time, it gives wrong words for some terms. We used Standford SpellChecker which includes some NLP techniques, and we overcome this challenge when using DTW.

978-1-5090-6491-5/17/$31.00 ©2017 IEEE

Indeed, identifiers are often composed of terms reflecting domain concepts [7], referred to as "hard words". If they follow the Camel Case naming convention, for example removeFile or abbreviated words such that pntrResult, we can separate using camel case splitter. As well as if they used an explicit separator such as underscore or a number and a symbol, we can split. If they do not follow any naming convention, for example, addwords and popraw, we cannot separate using simple methods. In this paper we proposed an algorithm using N-gram techniques to split abbreviated and concatenated words in identifiers. The main contributions of this paper are the following:

1. Keyword search and download files from Sourceforge and GoogleCode forges depend on the information from XML file. Previous work of this research explains for GitHub to download and analyze the result [8].

2. New algorithm for analyzing identifiers and splitting the connected words in identifiers.

3. Result of using our framework to existing projects in GitHub.

II. PROPOSED SYSTEM

Our proposed approach consists of five major components, 1) Extracting information from XML file, 2) Crawl some projects from Sourceforge and GoogleCode depend on information from XML file, 3) parse all .java files into Abstract Syntax Tree and Extract All Identifiers and comments, 4) identify the real words in the identifiers and identify the action word from comments, 5) compare the identified words and information from XML file and giving marks for matched words. Our proposed approach addresses the described problem by accepting software architecture (class diagram) in XML format, then extract some information from the software architecture (class name, methods name, attributes name). In this research we focus on program identifiers are a fundamental source of information to understand a software system, programmers are choosing identifiers to express problems, which are to be solved.

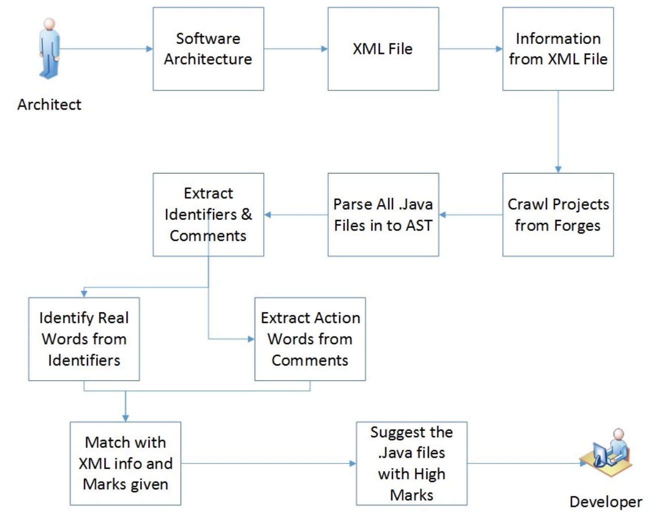
Architect
Software Architecture
XML File
Information from XML File
Crawl Projects from Forges
Parse All .Java Files in to AST
Extract Identifiers & Comments
Identify Real Words from Identifiers
Extract Action Words from Comments
Match with XML info and Marks given
Suggest the .Java files with High Marks
Developer

Fig. 1. System operational flow of the proposed solution

III. METHODOLOGY

Our tool includes modular architecture with many components. Some main components those are the main pillars of this framework are XML extractor, two crawlers for Sourceforge and GoogleCode, AST use, N-gram and WordNet use.

XML Extractor - Purpose of this is to extract information from XML file. For this purpose we used DOM (Document Object Model) parser, which is included in javax.xml package. It has two main classes such as, DocumentBuilder: Defines the API to obtain DOM Document instances from an XML document, DocumentBuilderFactory: Define a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents. As well we used 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 those API, we developed XML extractor to extract information from XML file.

Crawlers – crawler is a small program which is used to fetch a website and extract particular information from websites. Crawler starts with a keyword and a seed URL, then we can use that as needed. We have created three types of crawler to download projects from Sourceforge, GoogleCode and GitHub (is different from other two crawlers). We have discussed about GitHub crawler in detail in our earlier publication [8].We used JSOUP Java API to develop crawler.

AST use – Abstract Syntax Tree is representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. To avoid downloading all projects from our three forges, which will result in giant amount of java files, we need an AST Parser to extract identifiers and comments. We used Eclipse JDT AST parser API to represent our all downloaded .java files. After parsing java files, we extracted all identifiers and comments.

N-gram and WordNet use – We have developed an algorithm to identify the real words from abbreviated identifiers as well as split the identifiers those are not following any explicit separators. For above mentioned problems we used N-gram technique in the algorithm. N-gram is a NLP technique, depends on the N value, it splits a word into chunks. For an example, the word is "file" and N=2, the chunks are "fi", "il", and "le". Sometimes the information and identifiers are not matched even they have same meanings. To prevent synonym, we used WordNet to get all synonym of a word and do the matching process again. WordNet is a lexical database of English word and sense relations. A sense is a particular meaning of a word. For each sense of a particular word, WordNet provides the synset, a list of synonyms for that sense. We used WordNet in two places in our framework. When searching the source code also we get synonyms and search again.

TABLE I. SAMPLE RESULT OF DOWNLOADING THE RELEVANT PROJECTS

Targeted Projects names	Keywords	Downloaded Projects name	# of projects	Targeted Project is there
Vrapper	Controller -CORS -Option -Request	-ABR_controller -accountant-app -aprendendo-vraptor -vraptor multimodule	142	Yes
PrintWB	Devicelist -Create -pairedlist -Intent	-IntentFilter -ICEPOINTG -PRINTWB	230	Yes

IV. ANALYSING IDENTIFIERS AND COMMENTS

We will have a large amount of downloaded projects after crawling process and then we parse all .java files using Abstract Syntax Tree to extract identifiers and comments from those downloaded projects. Following challenges are faced after above process, (1). We receive many connected words and abbreviated words inside extracted identifiers pool. Table 2 shows the result of connected words and abbreviated words in the projects, which was taken to test. (2). Splitting the identifiers, which are not following Camel Case naming convention or any explicit separator, was very difficult.

A. Analyzing the Identifiers

First the identifiers were divided into two sets, one set contains identifiers that can be split is called splitSet, other set contains identifiers that cannot be split named as unSplitSet. Then we parse the splitSet identifiers through our own splitter to split one by one. Then split terms will be saved in a Set. After finishing all the split, all the terms will be parsed to Stanford Spellchecker to identify the meaningful words. If the term is "pntr", let us assume the programmer has written "pntr" instead of *printer*. The spellchecker gives 10 results for the term, which includes print as well, in third position. If "pntr' was written instead of "pnt", print would be at the 10th position in the result list. Table 3 shows the result of some abbreviated terms and related words. Next step is unSplitSet identifiers, which does not follow any naming convention; further it is difficult when the words are shrunk into acronyms. Those abbreviated connected identifiers are having two or more words. To identify those words, we have developed the following algorithm:

- If the identifiers following Camel Case naming convention or any explicit separator, split using our own splitter.
- Then identify the words using Stanford SpellChecker.
- If they do not follow any naming convention and number of characters is more than two, use 2-gram to make chunk and check the spelling.
- If the SpellCharecter identify the word, remove that chunk and repeat the work.
- If the Spellchecker do not identify the word, increase the N by one in N-gram technique and repeat the process until the length of words is equal to N. (Table 2 shows the result of this identifier)

B. Analyzing the 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 [10]. Hence, 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 example, "// add a word to a file", is the comment, *addWord(String wrd)* or *putWord(String wrd)*, is a method signature. In above comment and method signature, the action verb is "*add*" and method signature is "*add*" and "*put*". In the computer context the "*add*" and "*put*" are same word, so if one word from those two words will be matched with the words from class diagram. Likewise, some synonyms also will be there in comment and method signature. We used Stanford Part of Speech tagger, to tag all words whether the word is a verb or a noun or etc. if it tags as verb, we will extract the word and add to splitSet to analyze as identifiers.

V. SUGGESTING RELEVANT JAVA FILES

The Stanford SpellChecker gives 10 words for each incorrect word. Hence, through the use of Stanford SpellChecker, we increase the words to be checked with the XML information to matching purpose and the probability of matching identifiers will be increased. We take each class files in a project one by one and then parse the class in AST to access identifiers and comments. After analyzing the identifiers and comments, on the one hand we will have vast amount of words from a Java classes and on the other hand we will have some words from XML file (Class diagram). Then, we will take words one by one and check with the words from the other hands, if the words are matching we assign marks using following equation (1),

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

In (1), M is denoted marks is to be given for classes, Mi is denoted the initial marks for every loop iteration. N is denoted the number of identifiers extracted XML file. if a class diagram has 50 (N=50) identifiers, each identifier will get 2% (100% / 50) of marks, if 20 identifiers matched with a download class, the class will get 40% of marks, if the total mark is greater than 50%, the class will be selected to suggest most relevant class. Likewise every class from downloaded directory will be analyzed and suggested.

TABLE II. SAMPLE RESULTS FOR ANALYSING IDENTIFIERS

All Identifiers	# of Identifiers	Connected Word sample	# of connected word	Single word sample	# of single word	Sample Good Identifiers	# of Good Identifiers	Sample Bad Identifiers	# of Bad Identifiers

telefone
addToView()
allowed
mModeLabel
	478	mModeLabel
		writeTag()
		isAtivo()
		BUILD_TYPE
			370	Allowed
				Rider
				Size
				bo
					108	riderText
						newLocale
						btnSend
						listView
							472	isAtivo
								todoAtivos
								todos()
								timeregexp
								mywebview
								Htmlcontent()
									06

TABLE III. 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,..etc
phraseLocale	Phrase, Locale		
lstview		lstview	last, list, lost, stove, via, …etc
recordNewRider	Recode, New, Rider		

VI. SUGGESTING RELEVANT JAVA FILES

We examine the following questions to evaluate our developed system. We designed our evaluation to answer the following evaluation 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 study were identifiers from 5 open source Java programs across multiple domains and with different developers from GitHub repositories (for testing purpose we considered only the projects downloaded from Git forge). In total, the projects were comprised of 3985 lines of code from 15 source files. In this dataset, there were 169 methods, with methods documented by leading comments, and there were 309 attributes. We ran each of the four phases (1) Downloading the relevant project depend on the information from XML file (class diagram), 2) Identify the words from identifiers, which are following naming convention, 3) Identify the words from identifiers, which are not following the naming convention, 4) Extracting the word from comments) on the entire set of five Java projects. The results of the each phase were different, we could not show in one common result; hence we explain the results of the four phases separately. We discuss the result of each of the four phases below.

A. Downloading the relevant projects

Research question: How well does our system download relevant projects?

Procedure: We draw five class diagrams for the five projects, which exist in GitHub repositories. Then we saved the diagram in XML file format, and then we extract some information from the file. Finally we parse this information as keywords to our developed GitHub crawler.

Result: The developed GitHub crawler downloaded 735 projects for all different keywords; it downloaded all the targeted 5 projects, so accuracy of our developed crawler for this test case was 100%. Table 1 shows the sample result of the process.

B. Identify the words from identifiers

Research Question: How well does our System identify the real words from Good identifiers?

Procedure: All downloaded projects are in zipped or compressed format, and we extracted all 5 projects using our developed de compressor component. After that we parsed all java source files into AST parser. Finally we extract all identifiers from a java file; likewise we extracted those identifiers one by one from all java files. We put those identifiers into two categories, such that, Good identifiers which is following a reliable naming convention, and Bad identifiers, which are not following any naming convention. First we consider the Good identifiers.

Result: We got 34 java files from all 5 projects, from those java files, we extracted 472 Good Identifiers out of 478 identifiers, we split those identifiers using our Splitter, and then if the terms from identifiers are abbreviated, we identified the real word of the terms using Stanford Spellchecker, Table 3 shows the sample result of this process.

Research Question: How well does our System identify the real words from Bad identifiers?

Procedure: We mention Bad identifiers are identifiers which do not follow any naming convention; it means Camel Case or any explicit separator is not followed. For example, Instead of "add number" they have written "adnum", through human intuition we may easily identify the real word but it is difficult to identify the real word for the machine. Therefore we used the developed algorithm to identify the matching real words.

Result: With the received java files from the 5 projects, we extracted 6 Bad Identifiers out of 478 identifiers. We parse those identifiers through our developed algorithm to identify the real words; Table 2 shows the sample result of this process.

C. Extracting the action word from comments

Research question: How accurate is the automatic extraction of the action verb from a comment?

Procedure: The purpose of comments in source code is to describe the task of the class or task of the method. Most of the time methods implement functions as part of the system use cases. Hence in the leading descriptive comment the developer is using some action verb to describe the method action. Therefore, we extract all leading comments using AST parser, then we analyze the action words as identifiers with xml details.

Result: We got 34 java files from all 5 projects; from those java files we extract of 126 action words. We parse those action words as identifiers to match with XML file details; most probably those words were same to the methods name.

VII. CONCLUSION

As discussed earlier, there are several constraints and challenges for analyzing the Identifiers to code reuse. Although it is a challenging process we have developed a framework to address this problem. The framework we contributed has higher accuracy in all phases as we presented earlier: the tool showed 100% accuracy in downloading the related projects based on the class diagram information of the projects as given in Table 1. From those 5 projects, we extracted 34 java files, and 3985 lines of codes. We got 478 identifiers from those java files in which 472 good identifiers and 6 bad identifiers, from Good identifiers; totally there were 679 split terms, 131 Dictionary meaningful words and 548 Unknown terms. From Bad identifiers, there were 176 non split identifiers, in which 70 dictionary words, and 106 unknown terms. Using our Stanford SpellChecker, we got 2103 related word for 548 split term. The tool could not identify approximately about 6% of words, which were abbreviated heavily and badly. We therefore claim the tool accuracy for this level to be 94% in identifying the real words from Good identifiers. For 106 unknown terms, our proposed algorithm suggested 304 related words. Because of unknown terms are so badly abbreviated, and did not follow any Naming Convention, our Algorithm identified approximately ~87% of real words, however it failed to identify about 13% of words. Nevertheless, through the overall analyzing and matching process our targeted 5 projects were identified correctly.

A. Implications of the Research

The implication of this research is vital for the software development industry. At the rate of increasing demand for large scale and complex software solutions the software development is done by developers whose skills and experience can vary substantially. While experienced software engineers and programmers can easily implement their software with significant amount of code reuse due to their knowhow and awareness of existing code, novice developers tend to spend more time either on developing from scratch or looking for a good code to be reused. With the accuracy we have received from our tool, it can be safely said that this tool can significantly help software developers who are not familiar with already available and relevant source codes in popular repositories. This can help train these novice developers to work as experienced software professionals saving considerable amount of time and money for the project that they are engaged in.

The other main advantage comes from the quality assurance of software projects being developed. When developers engage in fresh code development from scratch their code has to go through the complete quality assurance cycle from unit tests to integration testing to system testing. Irrespective of the developer capability, no software company or project leader would like to take the risk of blindly accepting a freshly wrote code without having the quality assurance and testing on that. As the code is new it is more likely to take more effort and time to go through these testing stages and the chances for defect fixing efforts and improving code quality will be high. In contrast large majority of code segments committed to online code repositories such as the one we considered here are thoroughly tested before their commits and even after being available in the repository there are so many developers use those code and give feedback; as a result of this crowdsourced testing, although informal manner the code available in repositories with public access can be safely considered as high quality provided that there is no feedback comments from users otherwise. Therefore, the more we reuse code from repositories for our matching architectures the better the quality of our final system. Furthermore, because of this reason the project effort and cost related to quality assurance can be minimized as mostly we have to carry out only integration and system testing. The tool we have provided in this research can easily address the challenge of finding suitable code and motivate developers to reuse existing quality code segments thereby reducing the efforts for the quality assurance in the project.

B. Limitations and Future work

Our developed framework initiates its operational process with a class diagram in XML format; therefore developer has to enter the class diagram in XML format. When developing a class diagram and assigning a name into an attributes or method, if it has more than two words such names must be put by following a naming convention. Whether it is Camel case or any explicit separator based naming conversion it is fine. At the same time it is important to keep in mind that there should not be any abbreviated words, which can affect the meaningful term identification. One may consider these constraints in the operation as the current limitations of the software tool we developed and this can be improved as future extensions. After entering a class diagram, our framework can perform entire process itself automatically. Another limitation is experienced in downloading the projects from repositories; due to the performance reasons the current crawlers are limited to support projects with the size less than 50MB; if the project is larger than that our crawler will skip the URL. However, this can be easily fixed through performance enhancements.

In future development, it is planned to include automatic Design pattern suggestion with our developed framework. When software architecture gets into the development process, the developer needs to search for relevant sample code segments or libraries; for that our developed framework, which is capable of completing the process automatically can be used confidently; an extension feature for the proposed tool would be to support developers by giving them help to select a proper design pattern. For this selection process we planned to use Case-Based Reasoning (CBR) as one of successfully applied AI technologies recently. It helps to find solutions to unsolved problems based on pre-existing solutions of a similar nature. With that we expect to help address the architectural erosion and degradation during the implementation.

REFERENCES

[1] G. Sridhara, E. Hill, L. Pollock and K. Vijay-Shanker, "Identifying Word Relations in Software: A Comparative Study of Semantic Similarity Tools," *2008 16th IEEE International Conference on Program Comprehension*, Amsterdam, 2008, pp. 123-132.

[2] M. R. Marri, S. Thummalapenta and T. Xie, "Improving software quality via code searching and mining," *2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, Vancouver, BC, 2009, pp. 33-36.

[3] N. Madani, L. Guerrouj, M. Di Penta, Y. G. Gueheneuc and G. Antoniol, "Recognizing Words from Source Code Identifiers Using Speech Recognition Techniques," *2010 14th European Conference on Software Maintenance and Reengineering*, Madrid, 2010, pp. 68-77.

[4] E. Hill, L. Pollock and K. Vijay-Shanker, "Automatically capturing source code context of NL-queries for software maintenance and reuse," *2009 IEEE 31st International Conference on Software Engineering*, Vancouver, BC, 2009, pp. 232-242.

[5] A. Kritikos, G. Kakarontzas and I. Stamelos, "A Semi-Automated process for Open Source Code reuse", *ENASE 2010 - International Conference on Evaluation of Novel Approaches to Software Engineering*, Jan 2010, pp. 179-185

[6] S. Gupta, S. Malik, L. Pollock, “Part-of-Speech Tagging of Program Identifiers for Improved Text-based Software Engineering Tools”, *2013 IEEE 21st International Conference on Program Comprehension (ICPC),* 21 May 2013. pp. 1-10

[7] L. Heinemann, M. Broy, M. Robillard, McGill University, Montréal, Kanada, “Effective and Efficient Reuse with Software Libraries” *the 20th ACM SIGSOFT International Symposium on Foundations of Software Engineering* Jully 2012

[8] P.Pirapuraj, I. Perera, “GITHUB Application Program Interface and WordNet for code reuse”, Fifth Annual Science Research Session-2016, South Eastern University of Sri Lanka. pp. 1-5