Improving Legibility and Understandability of Source Code by Using Korean Naming Convention Tool

[1]Seungjun Baek, [1*]Young B. Park

[1,] *Department of Computer Science & Engineering, Dankook University,*
tmdwns0770@naver.com, [*]ybpark@dankook.ac.kr

Abstract

The larger the software, the larger the size of the developing organization.(The software size is growing. Therefore, there is a growing software development organization.) Such increase in size of development teams lead to communication gaps between developers, which renders the legibility and understandability of source codes critical. Programmers interpret code software based on the identifiers of the source code. Disorderly nomenclature of identifiers can make it difficult to understand their meaning, requiring additional time and manpower and thereby incurring higher development costs. Moreover, as source codes are written in English, Korean programmers face the challenge of the language barrier when understanding source codes. In order to solve this issue a Korean naming convention tool can be used, which would entail the following: the natural language process can be applied to the identifiers to categorize the data, which in turn would undergo morphological analysis and matched with Korean words to be Romanized in their notation. Applying this method is expected to enhance Korean programmers' ability to read and understand source codes.

Keywords*: Legibility, Understandability, Source Code Analysis, Natural Language Processing, Naming Conventions*

1. Introduction

A software system is not written by a single programmer, but rather a team of many developers. Around 70% of a software system's source code is comprised of identifiers [1]. When these identifiers have varying naming conventions and software developers each employ different means of defining nomenclature, the legibility and understandability of source codes can be compromised severely [2]. The key reason behind such problems arise from wanton assignment of nomenclature to identifiers, which degrade legibility and understandability. This in turn increases costs and time of communication at both development and managerial levels [3]. By adhering to a clear set of rules and an official model that consistently and succinctly defines naming conventions for identifiers, the legibility and understandability of source codes can be enhanced significantly. In case of Java, a programming language that is available only in English, programmers whose native language is not English may find the language inherently challenging to read and understand. As a way to improve Korean programmers' ability to read and understand source codes, this study proposes a naming convention tool to notate identifiers in English into Korean. By using a morphological analyzer, the tool would first categorize the identifiers based on naming conventions and determine the word class. Next, the categorized English word is matched and converted into the corresponding Korean word. Third, the Korean word is Romanized into western alphabet based on a set of rules. This method will help enhance Korean programmers better read and understand source codes. Chapter 2 introduces natural language processing, Java naming conventions, and inconsistent gender identifiers. Chapter 3 presents the proposal for the Korean naming convention tool. Chapter 4 will present the conclusion of this study and discuss the way ahead for potential future studies.

[*] Corresponding Author
Received: Sep. 13, 2016, Revised: Sep. 22, 2016, Accepted: Sep. 26, 2016

2. Relation Work

2.1 Natural Language Processing

Natural language processing refers to the entire process of exploring, researching, and creating applications to enable computers understand and manipulate text or audio containing language that is natural for humans [4]. Natural language processing constitutes a part of artificial intelligence, and is comprised of morphological, syntactic, and semantic analyses, as well as a processing step. In practice, natural language processing is used to extract data from massive amounts of documents. This study has thus selected natural language processing as the means of extracting identifiers from source codes.

2.2 Java Naming Convention and Inconsistent Identifiers

2.2.1 Java Naming Convention

Although normally each company has its own naming conventions for Java identifiers, this study used the Java Code Convention [5] to define the nomenclature. The Java Code Convention [5] defines the naming convention for the component elects of the Java source code as follows:

Table 1. Java source code defines naming conventions [5]

Identifier	Definition
Class, interfaces	Nouns (phrases) should be used. First letter should be capitalized.
Methods	Verbs (phrases) should be used. First letter should be lower case.
Variables	Nouns (phrases) should be used. First letter should be lower case.
Literals	All caps.

When using two or more words in a name, use Camel Case and Pascal Case. Camel Case is used for methods and variables, and capitalizes the first letter of the subsequent words when using multiple words to define a nomenclature. Pascal Case is used for classes, and capitalizes the first letter of all words when using multiple words to define a nomenclature. As literals already capitalize all words for nomenclature, they should include an underscore (_) between words when using two or more in the naming. For example, methods are named using Camel Case (e.g. setName). Classes use Pascal Case(e.g. TestClass). The nomenclature that followed the Camel Case, Pascal Case, or underscore rule is known as a hard word. Names that did not follow any particularly distinct convention is called a soft word [6].

2.2.2. Inconsistent Identifiers

Inconsistent identifiers refer to issues wherein identifiers for code elements such as classes or methods are named without any consistency with regard to their meaning or structure, thereby degrading the legibility of the code [2]. Identifiers are compound terms comprised of one or more terms. If the term exists in standard English dictionaries, it is referred to as a word [8]. The inconsistency of the identifiers in the code elements can be determined by analyzing whether each term that constitutes the identifiers are used consistently in terms of meaning, structure, and part of speech [7].

Table 2. Inconsistent naming conventions means [7]

Inconsistency	Meaning
Inconsistency in term's meaning	Varying words with same meanings used for multiple identifiers
Inconsistency in term's structure	Similar alphabets constitute the terms

Inconsistency in part of speech	Part of speech of the word is the one used in inconsistent.

3. Technique to Enhance Legibility and Understandability of Source Code

Compared to anglophone programmers, Korean programmers find it more challenging to recognize English words. As such, in large projects, legibility of source codes may become severely limited. Moreover, as the meaning is inherent in the name of the identifier, Korean programmers may face the challenge of trying to exchange information amongst themselves in English. In order to redress such issues, designing a Korean naming convention tool can help find a solution.

The method for enhancing legibility and understandability of the code is comprised of three procedures as seen in Figure 1.

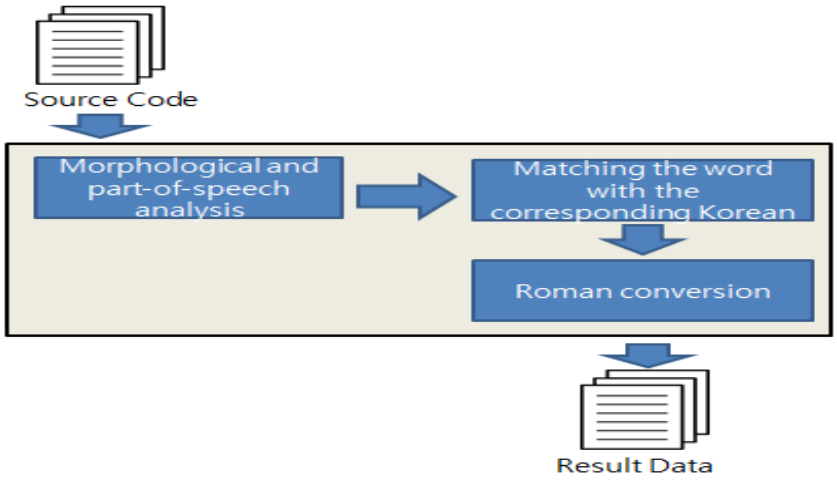

Figure 1. Korean naming tool

3.1 Morphological and part of speech analysis

At this stage, the word used to name the identifier is analyzed for morphology and categorized in accordance with the naming convention of the tool. The separated word is analyzed to determine the part of speech by using the natural language syntax analyzer. The words used to name identifiers are single words or multiple words that form compounds or phrases, and thus do not form complete sentences [7]. Thus the component elements need to be converted in order to use the syntax analyzer. To do so, a space should be entered between each term of the identifiers, and a period (.) should be added at the end of method identifiers. As mentioned in 3.1, spaces should be entered between all terms so that as in Figure 2, the nomenclature ArrayList is morphologically analyzed and split into Array and List. Each of these words were tagged for parts of speech.

```
import java.io.*;
import java.util.List;
import java.lang.reflect.Field;
import java.util.Scanner;
import kr.co.shineware.nlp.posta.en.core.EnPosta;
public class English {
    public static void main(String[] args) throws Exception {
            EnPosta posta = new EnPosta();
            FileWriter reader = new FileWriter("C:\\Users\\lyj\\test.txt");
            posta.load("datas");
            posta.buildFailLink();
            List<String> resultList = posta.analyze("Array List");
            for (String result : resultList) {
                  System.out.print("    "+result+" ");
                  reader.write(result+ "   ");
            }
            reader.close();
    }
}
```

oblems @ Javadoc Declaration Console Servers

nated> English [Java Application] C:₩Program Files₩Java₩jre1.8.0_71₩bin₩javaw.exe (2016. 9. 30.

Array/NNP List/NN

Figure 2. English morphological analyzer

3.2Matching the word with the corresponding Korean

At this stage, all words from the identifier names are matched with the corresponding Korean word. Once the terms in the identifier names are separated, each word is matched with a Korean word to re-designated meaning. For example, ArrayList is separated into "Array" and "List" in step 3.1. These words are then matched with the corresponding Korean words, "Array" to "배열" and "List" to "목록." The final name of the identifier will thus be ″배열목록″.

3.3 Roman Conversion

In this step, the name of the identifier converted into Korean is Romanized back into western alphabet. The identifier name extracted from step 3.2 is changed according to the Romanized equivalent. Each word is scanned syllable by syllable to determine the beginning, middle, and final sounds. The separated Korean words are converted into western alphabet based on the Romanization rules. In Figure 3, Array List has become ″배열목록″. Romanizing this would convert it to″ BaeYeolMokRok″.

Figure 3. Roman conversion

4. Conclusions and Future Studies

This study proposed a way to enhance Korean programmers' ability to read and understand source codes by developing a Korean naming convention tool. Identifiers would be extracted from the entire Java source code of a program, then processed into natural language words. These words are then separated for morphology and matched with corresponding Korean words. The Korean words are converted back into English alphabet based on the Romanization rules. This process can facilitate Korean programmers' understanding of the meaning of identifiers, and thereby enhance the legibility and understandability of source codes. This technique could also be applied to other non-Anglophone countries like Japan or China. Further research should also determine ways to address other programming languages in addition to Java.

5. Acknowledgements

This research was supported by The Leading Human Resource Training Program of Regional Neo industry through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT and future Planning(No.NRF-2016H1D5A1909989) and supported by the MISP(Ministry of Science, ICT & Future Planning), Korea, under the SW master's course of a hiring contract program supervised by the KISA(KOREA INTERNET & SECURITY AGENCY)

6. References

[1] Abebe, Surafel Lemma, and Paolo Tonella. "Natural language parsing of program element names for concept extraction." Program Comprehension (ICPC), 2010 IEEE 18th International Conference on. IEEE, 2010

[2] Deissenboeck, Florian, and Markus Pizka. "Concise and consistent naming."Software Quality Journal 14.3, pp. 261-282, 2006.

[3] N. Madani, L. Guerroju, M.D. Penta, Y. Gueheneuc and G. Antoniol, “Recognizing Words from Source Code Identifiers using Speech Recognition Techniques”, Techniques”, In Proceedings of 14th European Conference on Software Maintenance and Reengineering(CSMR), Madrid, Spain, pp. 68-77, 2010.
[4] Chowdhury, G. “Natural language processing. Annual Review of Information Science and Technology, 37.pp.51-89 ISSN 0066-4200, http://dxdoiorg/10/1002/aris.1440370103”
[5] Mircosystems, Sun, “Code Conventions for the Java Programming Language”, http://www.oracle.com/technetwork/java/index-135089.html, 1999.
[6] Lawrie, Dawn, Henry Feild, and David Binkley. "Syntactic identifier conciseness and consistency." 2006 Sixth IEEE International Workshop on Source Code Analysis and Manipulation. IEEE, 2006.
[7] Sungnam Lee, Suntae Kim, and Sooyoung Park. "Detecting Inconsistent Code Identifiers." KIPS Transactions on Software and Data Engineering 2.5, pp. 319-328, 2013.
[8] S.F. Abebe, S. Haiduc, P. Tonella and A. Marcus, "Lexicon Bad Smells in Software", In Proceedings of 16th Working Conference on Reverse Engineering, Antwerp Belgium, pp.95-99, Oct., 2008.