pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2013.2.5.319

Detecting Inconsistent Code Identifiers

Sungnam Lee[†] · Suntae Kim[††] · Sooyoung Park[†††]

ABSTRACT

Software maintainers try to comprehend software source code by intensively using source code identifiers. Thus, use of inconsistent identifiers throughout entire source code causes to increase cost of software maintenance. Although participants can adopt peer reviews to handle this problem, it might be impossible to go through entire source code if the volume of code is huge. This paper introduces an approach to automatically detecting inconsistent identifiers of Java source code. This approach consists of tokenizing and POS tagging all identifiers in the source code, classifying syntactic and semantic similar terms, and finally detecting inconsistent identifiers by applying proposed rules. In addition, we have developed tool support, named CodeAmigo, to support the proposed approach. We applied it to two popular Java based open source projects in order to show feasibility of the approach by computing precision.

Keywords : Inconsistent Identifiers, Source Code Analysis, Natural Language Processing

코드 비 일관적 식별자 검출 기법

이 성 남[†] · 김 순 태[††] · 박 수 용[†††]

요 약

소프트웨어 유지 보수 담당자는 코드 식별자를 중심으로 소프트웨어의 소스 코드를 이해한다. 그렇기 때문에 코드의 식별자를 비 일관적으로 사용하게 되면 소프트웨어를 이해하는데 어려움을 격게 되어 결국 소프트웨어의 유지보수 비용이 증가하게 된다. 이러한 비 일관적인 식별자 사용의 문제를 해결하기 위하여 개발자가 상호 검토하는 방법이 있으나 코드의 양이 많은 경우에 전체 코드를 확인하는 것은 불가능할 수 있다. 본 논문에서는 자연어 처리 기법을 사용하여 자동으로 Java 코드 내의 비 일관적인 식별자를 검출하기 위한 기법을 소개한다. 이 기법에서는 프로젝트 내의 모든 식별자를 추출 및 구문 분석하고, 구조상 유사어와 의미상 유사어를 분류한 후 최종적으로 제안된 규칙을 기반으로 비 일관적인 식별자를 검출한다. 본 논문에서는 지원 도구인 CodeAmigo를 개발하여 제안된 방법을 지원하였다. 우리는 지원 도구를 두 가지의 널리 알려진 Java기반 오픈 소스 프로젝트에 적용하고, 검출 결과의 정확도를 계산하여 제안된 접근 방법의 타당성을 확인하였다.

키워드 : 비일관성 식별자, 소스 코드 분석, 자연어 처리

1. 서 론

소프트웨어 유지 보수 담당자는 소프트웨어를 이해하기 위하여 소스 코드 식별자(Identifier)와 코드 내 주석(Comment)의 내용을 의존하게 된다. 하지만 코드 내의 주석이 충실히 되어 있는 경우는 흔치 않아 결국 유지보수 담당자는 식별자를 중심으로 프로그램을 이해하게 된다[1]. 식별자는 개발자가 명명하는 클래스, 메소드, 필드, 파라미터 등의 코드 구성 요소의 고유한 이름으로 해당 구성 요소의 특성과 행위 등을 표현한다. 그래서 적절히 명명되지 않고, 비 일관적으로 사용된 식별자는 소프트웨어의 이해를 상당히 어렵게 한다[2]. 다수의 개발자가 장시간에 걸쳐 개발하는 소프트웨어에서 비 일관적인 식별자는 흔하게 발견된다. 이는 개발자마다 표현 방식이 다르고, 시간에 따라서 요구사항 등의 구현이 달라지기 때문이다.

비 일관적인 식별자를 검출하기 위하여 여러 연구가 진행되어 왔다([2-6] 참조). 이는 크게 두 가지로 분류 가능하다. 첫째, 식별자와 식별자가 의도하는 개념 사이의 매핑 정보를 기반으로 비일관성 식별자를 검출하는 방법으로[2-4], 식별자 구성 패턴과 WordNet[7]을 활용하여 동음이의어(Homonym)와 동의어(Synonym)를 분석하였다. 둘째, 코드 식별자로부터 온톨로지 정보를 구축하여 코드에 대한 이해를 높이고, 이를 추후에 식별자 명명에 사용하기 위한 접근 방법[5-6]이 있다. 이 방법은 현재 코드의 비일관성 식별자

† 정 회 원 : 방위사업청 과장
†† 정 회 원 : 강원대학교 컴퓨터공학과 조교수
††† 정 회 원 : 서강대학교 컴퓨터공학과 교수
논문접수 : 2013년 1월 4일
수 정 일 : 1차 2013년 1월 14일
심사완료 : 2013년 1월 15일
* Corresponding Author : Suntae Kim(stkim@kangwon.ac.kr)

검출보다는, 분석 이후 비 일관적 식별자를 줄이는데 기여할 수 있다. 본 연구는 첫 번째 접근 방법의 확장으로 생각될 수 있다.

본 논문에서는 자연어 처리 기법(NLP: Natural Language Processing)을 사용한 Java 소스 코드 구성 요소의 비 일관적인 식별자를 검출하는 기법을 제안한다. 이 기법은 1) 모든 코드 구성 요소를 Java의 명명 규칙을 기반으로 형태소 분석을 하여 식별자를 구성하는 용어(Term)를 추출하고, 구문 분석기[8]를 통하여 각 용어의 품사(POS: Part of Speech)를 식별한다. 2) 각 용어는 온톨로지[7]를 사용하여 의미상 유사어(Semantic Similar Word; Synonym)를 식별하고, 용어 내 문자간 거리 측정 알고리즘[9]을 사용하여 구조상 유사어(Syntactic Similar Word)를 추출한다. 3) 마지막으로, 제안된 규칙을 통하여 용어의 품사, 의미 그리고 구조상의 비일관성 식별자를 검출한다.

본 논문의 기여는 다음과 같이 요약할 수 있다. 첫째, 코드에 대한 배경지식이 없는 개발자도 비일관성 식별자 검출을 용이하게 할 수 있다. 둘째, 코드의 전수 검사를 통하여 비일관성 식별자 검토의 누락 비율을 감소시켜 코드의 유지보수성을 높일 수 있다. 마지막으로, 본 논문의 제안 기법을 2종류의 Java 기반 오픈 소스 프로젝트에 적용하였으며, 검출한 결과를 정확도(Precision) 계산을 통하여 제안한 접근 방법의 타당성을 보였다.

본 논문은 다음과 같이 구성되어 있다. 2 장에서는 소스 코드의 비일관성 식별자를 검색하기 위한 관련 연구를 소개하고, 3장에서는 Java 명명 규칙과 비일관성 식별자에 대한 배경 지식을 소개한다. 4장에서는 자연어 처리 기법을 사용한 Java 소스 코드 내의 비일관성 식별자 추출 기법을 제안하고, 5장에서는 지원 도구인 CodeAmigo를 소개한다. 6장에서는 본 연구의 접근 방법을 오픈 소스 프로젝트에 적용한 사례를 소개하고, 마지막으로 7장에서는 결론 및 향후 연구에 대하여 논의한다.

2. 관련 연구

Deißenbock과 Pizka[2] 는 코드의 식별자의 일관성(Consistency)에 대하여 정형적으로 정의하였다. 이들은 일관성에 대하여 정의하기 위하여 개념과 식별자를 구분하여 하나의 식별자가 두 개 이상의 개념에 대응되는 경우는 동음이의어(Hononym)로 정의하였으며, 하나의 개념에 두 개 이상의 식별자가 대응되는 경우는 동의어(Synonym)로 정의하였다. 예를 들면, *File*은 *File Name, File Handle*의 두 의미를 가질 수 있는 동음이의어이며, *account number*와 *number*는 *account number*를 의미하는 동의어이다. 또한, 지원 도구인 IDD(IDentifier Dictionary)를 소개하여 프로젝트 내에 일관성 규칙에 위배된 식별자에 대한 경고 정보를 보여주었다. 하지만 이 접근 방법에서 개념과 식별자간의 대응 관계는 개발자가 수동으로 연결시켜주어야 한다는 단점을 가지고 있다.

Lawrie et al.[3] 은 Deißenbock과 Pizka[2] 의 연구의 단점을 개선하기 위하여 식별자 구성 패턴과 WordNet[7]을 활용하였다. 동음이의어의 경우 보통 식별자내에 또 다른 식별자가 존재한다는 패턴을 기반으로 식별하였다. 예를 들면, *FileName*과 *FileHandle*내에 모두 *File*을 가지는 경우가 그 예이다. 또한, 동의어를 식별하기 위하여 의미상 유사한 단어정보를 제공하는 온톨로지인 WordNet을 활용하여 동의어 검색을 자동화하였다. 하지만 여기에서는 단어의 품사를 분석하지 않았기 때문에 동의어의 범위는 상당히 넓고, 의미의 유사 정도도 매우 상이할 수 있어 정확도는 많이 떨어지게 된다.

Abebe et al.[4] 는 Lexicon Bad Smell이라는 적절치 않은 식별자를 지칭하는 개념을 소개하고, 이는 향후 리팩토링[10]의 대상이 될 수 있다고 소개하였다. 여기에서는 잘못된 식별자의 증상과 어떻게 리팩토링을 해야 하는지에 대한 소개와 함께 이를 자동으로 탐지할 수 있는 도구인 LBSDetector를 소개하였다. LBSDetector를 사용하면 Class의 이름에 명사가 포함되었는지, 또는 Class식별자와 동일하거나 Class 식별자를 포함하고 있는 Attribute, Operation 식별자가 있는지 등을 파악할 수 있다. 하지만 이 접근 방법은 비일관적 식별자는 동음이의어에 대해서만 다루었으며, 이는 Lawrie et al.[3]의 접근 방법과 유사하다. 또한, 검출 방법이 엄격하여 개발자에게 오히려 불편을 줄 수도 있다. 예를 들면 아래의 경우 Abebe et al.[4] 접근 방법은 *checkSameAccount* 메소드 식별자는 클래스 식별자와 동일한 단어인 Account를 포함하기 때문에 Lexicon Bad Smell로 탐지한다. 하지만 이는 메소드의 정확한 의미 전달을 위하여 필요할 수 있다.

```
class Account{
    boolean checkSameAccount(Account aAccount);
    //Account is redundant information
}
```

Abebe와 Tonella[5] 과 Falleri et al.[6] 은 소스 코드의 식별자를 사용하여 개념과 개념 사이의 관계를 포함한 온톨로지를 구축하였다. 여기에서는 코드 식별자를 구성하는 단어로 분리한 후, 제안한 규칙에 따라 문장을 생성하여 자연어 파서를 통하여 단어 간의 관계를 파악하고, 이를 지식화하였다. 이를 통하여 개발자들은 코드에 사용된 단어와 관련 지식을 검색하여 식별자를 명명할 수 있다. 이 연구는 자연어 파서를 사용하여 소스 코드를 분석한다는 점에서 접근 방식은 비슷하나, 연구의 목적이 비일관성 식별자를 검출하는 것이 아닌 코드 구성 개념 간의 관계(온톨로지)를 생성하기 위한다는 점에서 다르다.

3. 명명 규칙과 비일관성 식별자

3.1 Java 명명 규칙

Java Code Convenion[11] 에서는 Java소스 코드의 구성 요소에 대한 명명 규칙을 다음과 같이 정의하고 있다.

- 클래스와 인터페이스: 구성 용어는 명사(구)를 이루어야 하며, 첫 글자는 대문자여야 한다.
- 메소드 : 구성 용어는 동사(구)를 이루어야 하며, 첫 글자는 소문자 여야 한다.
- 변수: 명사(구)를 이루어야 하며, 첫 글자는 축약어(e.g., URL, HTTP) 이외에 소문자여야 한다.
- 상수: 모두 대문자이다.

이와 함께 두 개 이상의 용어로 이루어진 코드 식별자 명명 시 Camel Case(대소문자 혼합)를 사용하고, 상수의 경우 모든 용어를 대문자로 쓰고 용어 간의 구분은 underscore로 구분한다. 예를 들어, Class 이름 *WhitespaceTokenizer*의 경우 Whitespace와 Tokenizer로 이루어진 용어의 집합으로 명사구를 이루며, Method이름 *getElementNameForView()*는 get, Element, Name, For, View 네 용어로 동사구를 이룬다. 또한, 상수의 경우 'MIN_COUNT'의 경우 'MIN'과 'COUNT'의 단어로 이루어져 있다.

위와 같이 Camel Case와 underscore를 사용하여 명시적으로 분리되는 식별자를 hard word라 하고, 분리되지 않는 경우(예, whitespacetokenizer)는 soft word라 한다[3]. Java 이외의 다른 언어의 경우 이에 대한 제약사항이 상대적으로 약하기 때문에 Soft word로부터 용어를 식별하는 것은 리서치 이슈중 하나이다[1]. 하지만 Java언어의 경우 Camel Case와 underscore를 사용한 hard word를 명명 규칙으로 정하고 있고, 타 언어에 비하여 잘 준수하고 있다.

3.2 비일관성 식별자

비일관성 식별자란 클래스, 메소드 등의 코드 구성 요소의 식별자를 의미 혹은 구조상 일관적이지 않게 명명하여 코드의 가독성을 떨어트리는 문제를 의미한다[2]. 식별자(Identifier)는 하나 혹은 둘 이상의 용어(Term)로 이루어진 복합체 (Compound)이며, 용어가 일반적인 영어 단어 사전(Dictionary)에 존재하는 경우에 이를 단어(Word)라 한다[4]. 코드 구성 요소의 식별자가 비 일관적으로 구성되었는지의 파악은 식별자를 구성하는 각 용어가 의미상, 구조상, 그리고 품사의 면에서 일관적으로 사용되는지 분석함으로써 검출할 수 있다.

첫째, 용어의 의미상 비일관성은 여러 식별자에 동일한 의미를 가진 서로 다른 단어를 사용한 경우를 의미한다. 예를 들면, 메소드 이름 *getUserID()*와 *retrieveUserID()*에서 *get*과 *retrieve*는 동사로서의 의미는 유사하나 다른 용어이기 때문에 의미상 비일관성이라고 판단할 수 있다. 이는 여러 사람이 참여하는 소프트웨어 개발에서 흔히 나타나는 현상이며, 프로그램의 이해에 혼란을 야기시킬 수 있다.

둘째, 용어의 구조상 비일관성은 용어를 구성하는 알파벳이 비슷한 것을 의미한다. 이는 소스 코드에는 축약어(Abbrevation)가 많이 사용되기 때문에 발생하는 문제로, 예를 들면 *argument*를 의미하는 *args, arg0, arg1* 혹은 *parameter*라는 용어 대신 사용하는 *param, param0*등은 용어의 구조상 비일관성에 속한다. 이는 사전에는 나타나지 않지만, 코드에서는 흔하게 나타나는 용어의 구조상 비일관성의 예이다. 또한, 이는 오타에 의해서 발생할 수도 있다. 오타는 프로그램의 수행과 연관은 없으나, 개발자가 이해하기에는 방해가 된다.

셋째, 품사의 비일관성은 단어의 품사가 비 일관적으로 사용된 것을 의미한다. 몇몇 연구자들은 개발자들이 하나의 단어에 대하여 일관된 품사를 사용하는 것을 관찰하였다[12-13]. 예를 들어 *free*는 동사, 형용사 그리고 부사의 의미를 가지나, 프로그램에서는 동사만의 의미만 사용되었다. 뿐만 아니라, Deißenbock과 Pizka[2], Lawrie et al.[3] 연구에서 하나의 용어가 하나 이상의 개념과 대응된다면 이를 비일관성으로 정의하였다. 이는 하나 이상의 품사로 사용되는 자연어의 단어일 경우에는 보편적으로 적용될 수 있다. 그러므로 단어가 다양한 품사로 사용될 경우 역시 비일관성으로 정의할 수 있으며, 본 논문에서는 이를 품사 비일관성으로 정의한다. 이는 용어가 자연어 사전에 존재하는 단어일 경우에 한정된다.

품사 비일관성은 단어의 품사 비일관성과, 구문의 품사 비일관성 두 가지로 분류할 수 있다. 단어의 품사 비일관성이란 한 단어가 소스 코드에서 여러가지 품사로 활용되는 경우를 의미한다. 예를 들어, Class 식별자 *CheckAbort*에서 abort는 명사로 사용되었다. 하지만 *abort()*라는 메소드에서 abort는 동사로 사용되었다. 이는 abort의 품사를 혼용해서 사용한 경우이다. 구문의 품사 비일관성은 클래스와 메소드가 Java의 명명 규칙에 따라 명사구 혹은 동사구로 정의하지 않은 경우에 발생한다. 만일 클래스 이름이 *Aborted*라고 한다면 명명 규칙에 따라 이는 명사 혹은 명사구여야 한다. 하지만 이는 형용사이므로 구문의 품사 비일관성의 경우에 해당한다.

4. 비일관성 식별자 검출 기법

코드 내의 비일관성 식별자 검출 기법은 Fig. 1과 같이 다음의 3단계의 절차를 따른다. 1) 모든 코드의 식별자를 명명 규칙을 기반으로 용어로 분리하고, 자연어 구문 분석기를 사용하여 품사를 식별한다. 2) WordNet을 사용하여 각 용어의 의미상 유사어를 분류하고, Levenshtein Distance 알고리즘[9]을 사용하여 구조상 유사어를 분류한다. 3) 비일관성 용어 검출 규칙을 통하여 비일관성 용어와 식별자를 검출한다.

4.1 형태소 및 품사 분석

이 단계에서는 클래스, 변수, 메소드와 파라미터 등 의 코드 구성 요소의 식별자를 구성하는 용어를 식별한다. 식별자의 형태소 분석은Java 명명 규칙[11]을 고려하여 수행된다. 식별된 단어는 자연어의 구문 분석기를 사용하여 각 단어의 품사를 분석한다. 코드 식별자는 단일 용어 혹은 하나 이상의 용어로 이루어진 복합어(Compound) 혹은 구(Phrase)를 이루기 때문에 완전하지 않은 문장(Sentence)이

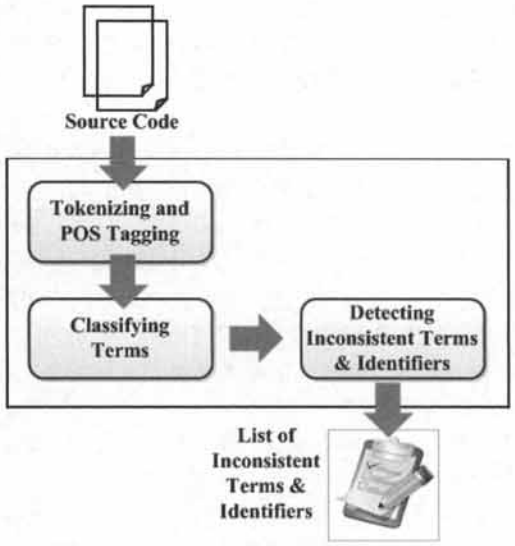

Fig. 1. Process for Detecting Inconsistent Identifiers

다. 그래서 구문 분석기를 사용하기 위해서는 구성 요소의 변환이 필요하다.

이를 위하여 먼저 식별자의 모든 용어 간에 공백을 삽입하고, 메소드의 식별자의 경우에만 마지막에 마침표를 추가한다. 이는 메소드는 동사구이므로 명령문과 같이 하나의 문장이될 수 있기 때문에, 구문 분석의 정확성을 위하여 완전한 문장을 만들도록 변환한다. 이 작업은 특히 명사와 동사의 의미를 함께 가진 하나의 단어로 이루어진 클래스와 메소드 식별자를 처리하기 위하여 중요한 작업이다. 예를 들어 Reset이라는 단어를 클래스 식별자로 단독으로 사용한 경우 개발자는 명사의 의미를 의도하고, 메소드 식별자로 사용한 경우 동사의 의미를 의도하고 사용했을 수 있다.

4.2 의미적 유사어와 구조적 유사어 분류

이 단계에서는 코드 식별자 내의 모든 용어를 의미상 유사어와 구조상 유사어로 분류한다. 의미상 유사어를 식별하기 위하여 본 연구에서는 WordNet[7]을 활용하였다. WordNet은 영어 어휘 데이터베이스로, 한 단어의 어근뿐만 아니라, 명사, 동사, 형용사 그리고 부사의 의미와 의미별 동의어 집합을 제공한다.

본 연구에서는 의미상 유사어(Synonym)를 찾기 위하여 전 단계의 용어 식별과 품사 분석의 결과를 활용한다. 이는 하나의 단어는 품사별로 다양한 의미를 가질 수 있고, 동의어는 이 품사별로 존재하기 때문이다. 코드 내의 유사어를 검색하기 위해서는 먼저 용어와 품사 정보를 기반으로 WordNet에서 동의어를 검색하고, 이 동의어가 코드 내에 존재하면 이를 의미상 유사어로 분류한다. 이때, 동의어 또한 동일한 품사여야 한다. 예를 들면, abort가 명사로 쓰인 경우에 WordNet에서는 이의 동의어가 termination, ending, conclusion이 동의어이며, 만약 코드에서 이 단어가 존재하고, 또한 그 단어가 명사로 사용되었다면 의미상 유사 단어로 분류될 수 있다.

구조상 유사어를 식별하기 위하여 Levenshtein Distance 알고리즘[9]을 활용하였다. 이 알고리즘은 두 단어 간의 구조적 유사 정도를 측정해주는 알고리즘으로 단어를 구성하는 알파벳의 차를 구하고, 이를 알파벳 수로 나누어 구한다. 예를 들면, kitten과 sitting의 알파벳 차이(D)는 Levenshtein Distance 알고리즘으로 3 (kitten ←→ sitting)이고, 이를 기반으로 단어 간의 구조상 유사도는 아래의 수식으로 구할 수 있다. 이때 length는 단어를 구성하는 알파벳의 개수다. kitten과 sitting의 구조 유사 정도는 수식에 따라 0.5이다.

$$SynSimilarity=1-(D/max(length(word1),\ length\ (word2))$$

4.3 비일관성 식별자 검출 방법

비일관성 식별자 검출 방법의 기본 원리는 다수결 원칙(Majority Rule)이다. 즉, 프로젝트의 참여자 대부분이 사용하는 단어와 품사가 그 프로젝트에서는 일반적으로 동의하는 것으로 판단한다. 다음은 단어, 품사, 유사 단어 분류 결과를 기반으로 식별자의 의미상 비일관성, 구조상 비일관성, 품사 비일관성을 검출하는 방법에 대하여 소개한다.

1) 의미상 유사어 비일관성 식별자 검출 기법

모든 식별자를 구성하는 용어에 대하여 WordNet을 통하여 동의어를 검색하여 이 중에서 의미의 유사 정도가 Threshold 이상(본 논문에서는 0.8로 정함)인 단어를 검출한다. 유사 정도는 WordNet에서 유사어에 대하여 Frequency별로 정렬한 순서를 기반으로 측정한다. Algorithm 1은 이를 계산하는 알고리즘을 보여준다.

이 알고리즘에서 이야기하는 모든 의미상 유사어는 WordNet에서 추출한 유사어 중에서 소스 코드 내에 존재하는 유사어만을 의미한다. 이 알고리즘을 통하여 계산된 결과 IC_i는 특정 $word_i$와 상당히 유사도가 높은 단어의 집합이다. 이 중에서 가장 코드에서의 출연 빈도가 높은 단어를 기준으로 하여 사용 빈도가 상대적으로 낮은 단어들은 의미의 비일관성 용어로 검출한다. 예를 들어, 소스 코드에서 동사 *get*의 동의어 중에서 코드에 존재하는 *acquire, grow*를 검색했다면, 유사 정도는 *get*의 동사 의미(sense) 30가지 중 첫 번째 의미인 'come into the possession of something concrete or abstract.' 과 동의어로 *acquire*가존재하고, *grow*는 열두 번째 의미인 'come to have or undergo a change of physical features'의 의미로써 동의어이다. 그러므로 *acquire*의 *get*과의 의미상 유사어 정도(semsim)는 (1-1/(30))=0.96이고, *grow*와의 의미상 유사어 정도(semsim)은 (1-12/30) =0.6이다. Threshold가 0.8이라고 하면 *acquire*만이 유사어가 된다. 그 이후 *get*과 *acquire*의 사용 빈도를 각각 측정한 후 적게 사용한 단어의 사용처가 의미상의 비일관성 식별자로 검출될 수 있다.

2) 구조상 유사어 비일관성 식별자 검출 기법

이 단계의 시작은 용어의 구조상 유사한 단어부터 시작한다. 원리는 구조상 유사한 단어 중 출현 빈도가 적은 것은

Input: w_i as a word, tr as a threhold
Function:
- $uPOS(w_i)$ as a function for examining used POSes of the word w_i
- $sw(w_i)$ as a function for getting synonyms of the word w_i
- $tots(w_i, pos_j)$ as a function for getting the number of senses of word w used as POS pos_j.
- $idxs(w1, w_i, pos_j)$ as a function for getting the ordered number of frequency of word w1 among the senses of word w_i

Output: IC_i as a set of sementic inconsistent name of w_i

01: $IC_i \leftarrow 0$
02: $POSs_i \leftarrow uPOS(w_i)$
03: for all $pos_j \in POSs_i$ do
04: for all $sw_k \in sw(w_i)$ do
05: semsim=1-($idxs(sw_k, w_i, pos_j)$/($tots(w_i, pos_j)$))
06: if(semsim > tr) $IC_i \leftarrow sw_j$
07: end for
08: end for

Algorithm 1. Extracting Semantic Inconsistent Words

구조상 비일관성 이름으로 검출된다. 하지만 이 접근 방법의 문제점은 코드에서 *String accent*와 *String[] accents*와 같이 단수/복수를 구분하여 사용하는 경우가 흔하게 있기 때문에 이 경우도 모두 구조 비일관성 용어로 검출될 수 있다. 하지만 이는 코드의 가독성을 높여주는 것이기 때문에 비일관 용어로 판단하기 어렵다. 이 문제를 해결하기 위하여 품사가 명사이고, 해당 단어의 어근이 동일한 경우(accent와 accents의 어근은 모두 accent)는 구조상의 비일관성 용어 후보군에서 제외하도록 하였다. 본 논문에서는 용어의 구조상 유사도가 0.8 이상의 단어를 구조상 유사어로 분류한다.

3) 품사 비일관성 식별자 검출 기법

품사의 비일관성 식별자 검출 기법은 구문 품사 비일관성과, 단어 품사 비일관성 검출의 두 가지 접근 방법으로 검출한다. 첫째, 구문 품사 비일관성은 구문 분석기의 분석 결과를 기반으로 Java 명명 규칙에서 제시하는 잘못된 품사 사용을 검출한다. 명명 규칙에서는 클래스는 명사 혹은 명사구로, 메소드는 동사 혹은 동사구로 명명하도록 하고 있다. 구문 품사 비일관성은 클래스 혹은 메소드의 이름을 구성하는 단어가 하나인 경우와 둘인 경우를 분리하여 식별한다. Fig. 2는 각 경우에 구문 품사 비일관성 식별자 검출 알고리즘을 보여준다. 구문 품사 비일관성은 Threshold가 존재하지 않고 명명규칙이 위배된 경우는 모두 일관성이 위배된 것으로 간주한다.

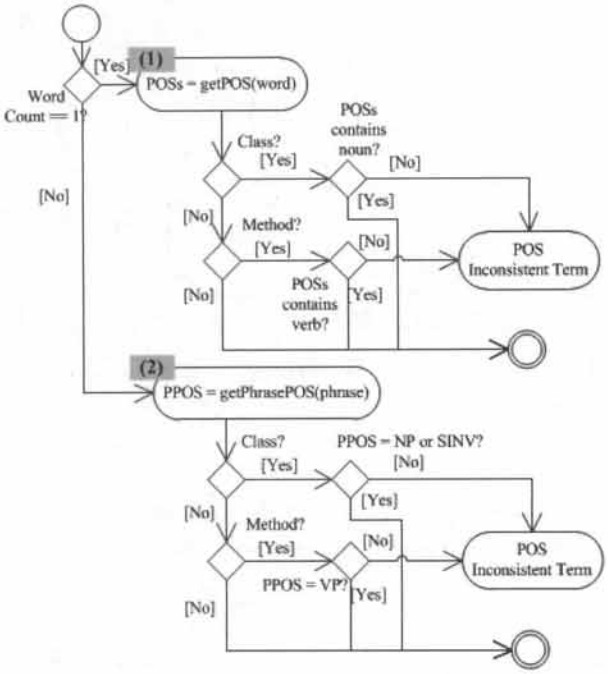

Fig. 2. Algorithm for Detecting Phrase-POS Inconsistent Identifiers

하나 이상의 용어로 이루어진 복합어는 구를 이룬다. 구로 이루어진 클래스의 식별자인 경우에는 명사구(NP:Noun Phrase)인 경우뿐만 아니라 도치구문(SINV: Inverted decrarative sentence) 역시 타당한 것으로 간주한다. 도치구문은 동사(구) + 명사(구)의 형식을 의미한다. 이는 클래스 이름의 경우 가장 후반부의 단어가 명사(구)인 것이 가장 중요하기 때문이다. 예를 들면, 클래스 이름 *Read OnlyDirectoryReader*의 경우 *Read Only*는 동사구를 이루고 *Directory Reader*는 명사구를 이룬다. 이때 *Read Only*는 컴퓨터 영역에는 고유명사로 활용되나, 자연어 구문 분석기는 동사구로 인식하기 때문에 이런 경우를 처리하기 위하여 도치 구문 역시 타당한 이름으로 간주한다.

둘째, 단어 품사 비일관성을 검출하기 위해서 한 단어의 품사별 사용 빈도를 측정 비교하여 Threshold(본 논문에서는 0.8로 정함) 이상 차이가 난다면 사용 빈도가 상대적으로 적은 품사의 사용처가 품사 비일관성 이름으로 검출한다. 예를 들면 abort의 총 출연 횟수가 50회인데, 이중 동사로 사용된 경우가 전체의 80%를 상회할 경우, 다른 품사로 사용된 부분은 품사의 비 일관적 이름으로 판단될 수 있다.

5. 지원 도구: CodeAmigo

비일관성 식별자를 검출하기 위하여 본 연구에서는 지원 도구인 CodeAmigo를 개발하였다. CodeAmigo는 Fig. 3과 같은 구성 요소로 이루어졌다. 이는 Eclipse를 기반으로 한 Plugin으로 개발되었으며, 구문 분석과 품사 분석을 위하여 StanfordParser[14] 를 활용하였고, 의미상 유사어를 찾기

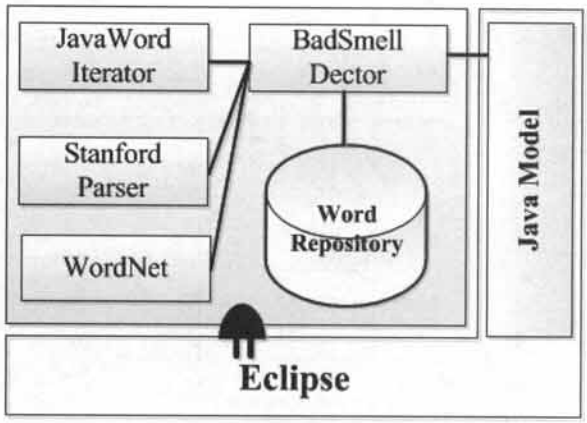

Fig. 3. CodeAmigo Software Architecture

위하여 WordNet과 지원 라이브러리인 JAWS(Java API for WordNet Searching)[15]와 JWI(Java WordNet Interface) [16] 를 활용하였다. 또한 식별된 단어와 구문 분석의 결과 등을 저장하기 위하여 HyperSQL[17] 을 활용하였다. Java 소스 코드 구성 요소에 대한 정보를 활용하기 위하여 Eclipse내의 Java Model을 사용하였다.

Fig. 4는 CodeAmigo의 실행화면을 보여준다. Code Amigo는 선택한 Eclipse Project내의 모든 소스 구성 요소를 분석하여 이를 구성하는 이름에 대한 비 일관적 이름을 식별하고, 그 결과를 보여준다. 그 결과는 코드 구성 요소의 타입, 구성 요소 식별자, 비일관성 타입과, 비일관성 검출에 대한 설명에 대하여 보여준다. 비일관성의 타입은 구문 품사 비일관성, 단어 품사 비일관성, 의미상 유사어 비일관성, 구조상 유사어 비일관성에 대하여 각각, POS-PHR, POS-WORD, SEM, 그리고 SYN으로 표현하였다. 결과에서 각 라인을 선택하면 해당 코드 블록으로 이동하여 코드의 사용처를 직접 조회할 수 있다. 검색 결과의 타당성 검증을 지원하기 위하여 해당 단어가 적절한지를 기록할 수 있도록 체크 박스를 배치하였으며, Excel로 모든 정보를 추출하도록 하였다.

6. 사례 연구

6.1 사례 연구 소개 및 방법

본 장에서는 논문에서 제안한 접근 방법을 Java 기반 오픈 소스인 Apache Lucene[18] 과 Apache Ant[19]에 적용한 사례를 소개한다. Apache Lucene은 오픈 소스 검색 엔진으로 문서와 이의 속성(Field)을 Index를 통하여 관리하고, 키워드와 조건 등을 입력하여 문서를 검색을 지원하기 위한 프로젝트이며, Apache Ant는 Command Line에서 Java 프로젝트를 빌드 및 배포하기 위한 개발 지원 프로젝트이다. Table 1은 두 오픈 소스 프로젝트의 주요 정보를 요약한 것이며, 여기에서 Classes는 Class와 Interface의 수를 합한 것이다.

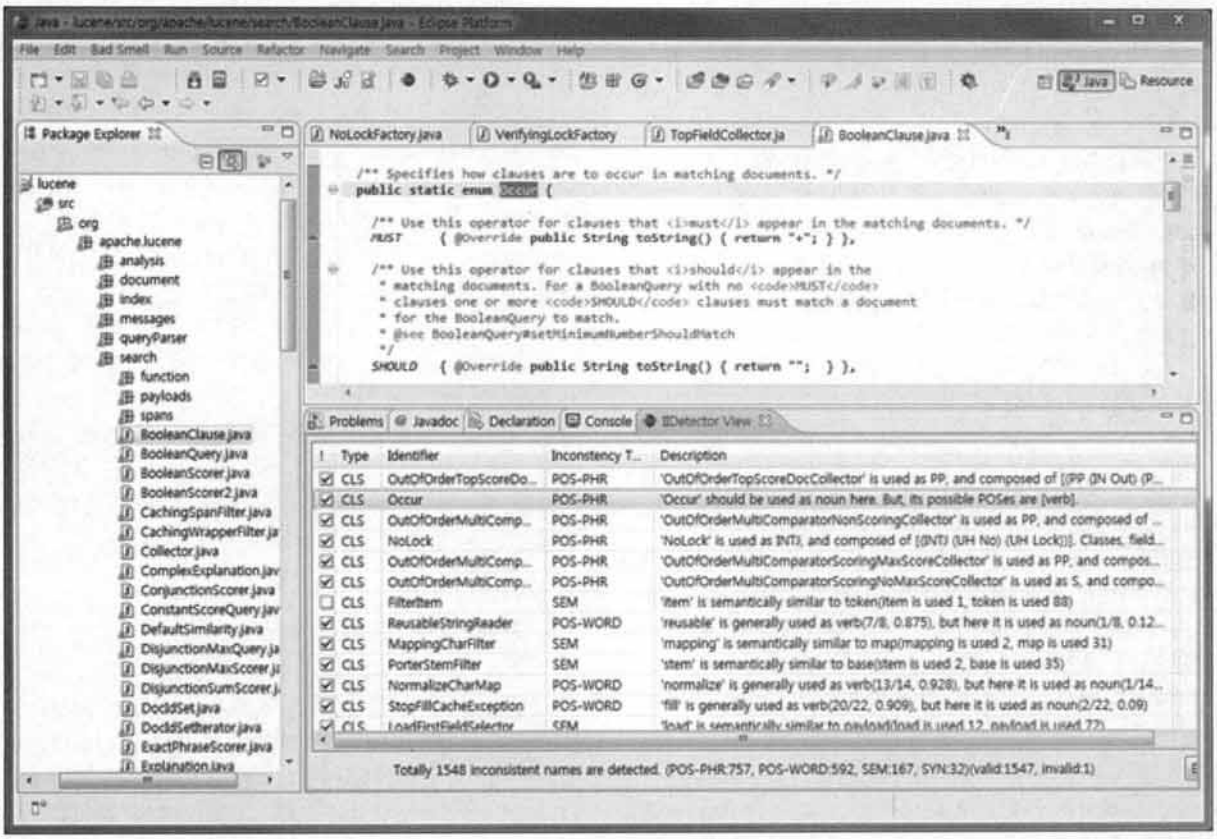

Fig. 4. Screen Capture of CodeAmigo

Table 1. Open Source Projects for this Case Study (SLOC: Source Lines of Code)

	Version	Classes	SLOC	Identifiers
Lucene	3.0.3	659	45,835	11,457
Ant	1.8.3	1,279	104,287	24,955

적용 사례의 목표는 제안된 접근 방법으로 식별한 비일관적인 식별자가 적절한지 확인하는 것이다. 이를 위하여 10년 이상의 개발 경력을 가진 5명의 Java 개발자에게 검출된 비일관성 식별자를 다음의 가이드라인을 기반으로 평가를 요청하였다. "만약 검출된 식별자가 의미상/구조상 일관적, 그리고 사용 품사가 일관적이라면 본 접근 방법으로 검출된 비 일관적 식별자는 적절하지 않다." 그 결과를 바탕으로 아래의 수식을 사용하여 Precision[20]을 계산하였다. 이는 CodeAmigo에서 검출한 결과(*#Retrieved*)와 이 중에서 평가를 통하여 타당하다 평가되는 것(*#Relevant*)의 비율로 계산한다. 하지만 Recall(전체의 비일관성 식별자와 본 접근 방법을 통하여 검출된 비일관성 식별자의 비율)을 계산하기 위해서는 방대한 코드를 분석하기 위한 시간과 비용의 문제로 계산하지 않았다.

$$Precision=\sum_i(\#Relevant \cap \#Retrived)_i)/\sum_i Retrived_i$$

6.2 평가 결과

Table 2는 CodeAmigo를 사용한 비일관성 식별자 검출 결과를 보여준다. Apache Lucene과 Ant 프로젝트에 대한 Precision 분석에서는 각각 83.8%와 62.5%의 결과를 보여주었다. 특히, Lucene의 구문 품사 비일관성(POS_PHR)에서는 정확도가 상당히 높게 나왔으나, 상대적으로 의미상 비일관성(SEM)과, 구조상 비일관성(SYN)은 적게 나왔다. Ant는 모든 비일관성 검출이 비슷하게 나왔으며 단어의 품사 비일관성(POS_WORD)은 상대적으로 적게, 구문의 품사 비일관성은 상대적으로 높게 검출되었다.

Table 2. Results of Detecting Inconsistent Identifiers using CodeAmigo

Apache Lucene			
	Detected	Valid	Precision(%)
POS_PHR	757	699	92.3
POS_WORD	592	470	79.4
SEM	167	108	64.7
SYN	32	20	62.5
Sum	1,548	1,297	83.8
Apache Ant			
	Detected	Valid	Precision(%)
POS_PHR	773	520	67.3
POS_WORD	1,088	645	59.3
SEM	498	310	62.2
SYN	11	7	63.6
Sum	2,370	1,482	62.5

Ant에서 검출한 비일관성의 정확도(62.5%)가 Lucene (83.8%) 에 비하여 상대적으로 정확도가 낮게 나온 이유는 Ant는 소프트웨어 프로젝트의 빌드를 지원하기 위한 시스템으로 상대적으로 컴퓨터 도메인에서만 사용하는 용어를 집중적으로 사용하였기 때문이다. 그렇기 때문에 일반적인 영어 단어 품사 분석이 되어 있는 Corpus를 기반으로 한 구문 분석기가 적절히 동작할 수가 없다. 대표적인 예로, File은 일반 영어에서 명사의 의미로는 '서류철', '정보'의 의미로, 동사로는 '문서를 보관하다.', '소송을 제기하다.'라는 의미로 활용되나, 컴퓨터 도메인에서는 자료를 저장하기 위한 물리적인 저장 공간을 의미하며 일반적으로 명사로 활용된다.

Lucene의 구문의 품사 비일관성이 92.3%로 상당히 높게 나온 이유는 동사(구)로 이루어져야 하는 메소드 식별자에 *allThreadIdle(), any(), anyChanges(), binarySearch(), directory(), difference()*와 같이 명사나 형용사로 이루어진 명사(구)를 사용한 경우가 많았다. 또한 명사(구)로 명명 되어야 하는 클래스, 필드, 그리고 파라미터에 *Create, checkAbort, closeReader*와 같은 동사(구)로 이루어진 식별자를 명명한 것이 많았으며, CodeAmigo는 이를 적절히 식별하였다. 다음은 대표적인 검출 사례를 소개한다.

1) 구문 품사 비일관성 검출 사례

- 메소드 식별자 *readerIndex()* : 동사(구)로 이루어져야 할 메소드 식별자에 명사구로 명명, 이때 CodeAmigo는 아래와 같은 설명을 제공한다.
  'readerIndex' is used as NP, and composed of [(NP (NP (NN reader)) (NP (NNP Index)) (. .))]. Methods should be named as a verb phrase.
- 클래스 식별자 *WaitFor* : 명사(구)로 이루어야 하는 클래스 식별자에 동사구로 명명함. CodeAmigo는 아래와 같은 메시지를 제공한다.
  'WaitFor' is used as VP, and composed of [(VP (VB Wait) (FRAG (PP (IN For))))]. Classes, fields and parameters should be named as a noun phrase.

2) 단어 품사 비일관성 검출 사례

- 클래스 식별자 *CCMCreateTask* : Create는 이 프로젝트에서 동사로 사용되는 경우가 많았지만, 위 식별자에서는 명사로 사용되었다. 이 식별자에서는 *CCMCreationTask*가 더 적절한 이름이다. 더불어, CodeAmigo는 *create*가 동사로 사용된 개수(357개중 341개) 대비, 명사로 사용된 사용 개수(357개중 16개)를 아래의 메시지를 통하여 보여준다.
  'create' is generally used as verb(341/357, 0.955), but here it is used as noun(16/357, 0.044).
- 메소드 식별자 *resumeAddIndexes()* : *Add*는 일반적으로 동사로 사용되었으나, 여기에서는 형용사로 사용되었다.
  'add' is generally used as verb(125/130, 0.961), but here it is used as adjective(3/130, 0.023)

3) 의미상 유사어 검출 사례

- *Resolve*와 *resolution*, 그리고 *append*와 *add* : 의미상 유사어이나 혼용되어 사용되었다. 이때 CodeAmigo는 아래와 같은 메시지를 생성한다.
  'resolve' is semantically similar to resolution(resolve is used 4 times, resolution is used 19 times)
- *Specification*과 *spec* : 축약어와 온전한 단어를 일관적으로 사용하지 않은 사례 역시 검출되었다. 이때 CodeAmigo는 아래와 같은 메시지를 생성한다.
  'spec' is semantically similar to specification(spec is used 7 times, specification is used 37 times.)

4) 구조상 유사어 검출 사례

- 메소드 식별자 *includes()*와 *include()* : 메소드의 명명은 동사구를 추천하나 이를 동사원형으로 써야 한다는 제약은 없다, 하지만 일반적으로 동사 원형의 사용을 제안한다[21]. 이는 단어의 구조가 유사한 경우 CodeAmigo는 아래와 같은 메시지를 생성한다.
  'includes' is sytactically similar to include(includes is used 10 times, include is used 50 times)
- 메소드 식별자 *getPreserve0Permissions()* : *CodeAmigo*는 *Perserve0*와 *Perserve*가 구조상 유사하다고 판단하여 다음과 같은 메시지를 생성하였다.
  'preserve0' is sytactically similar to preserve(preserve0 is used 1 times, preserve is used 14 times)

위의 사례 연구에서 CodeAmigo는 Lucene의 경우 16.2%, Ant의 경우 37.5%가 정확하지 않게 비일관성 식별자를 검출하였다. 그 이유를 다음과 같이 다섯 가지로 분석하였다. 첫째, 식별자의 구문 분석이 정확하지 않았다. Stanford Parser는 40개 미만의 영단어로 구성된 자연어 구문 분석에서는 86%의 정확도로 분석한다[14]. 그래서 정확하게 구문 분석을 하지 못하는 것이 존재한다. 예를 들면, 메소드 식별자 *loadTermIndex()*의 구문 분석을 *'[VP((VB load) NNP((NN Term), (NN Index))(..))]'*인 동사구로 분석하여야 하나, *[(NP (NP (NN load) (NNS Terms)) (NP (NNP Index)) (..))]*의 명사구로 분석하였다. 이는 자연어에 사용되는 단어 그리고 그 품사와 컴퓨터 도메인에서 사용되는 단어와 품사가 서로 다르기 때문이기도 할 것이다.

둘째, 코드에는 약어가 존재한다. 예를 들어 메소드 식별자 allocSlice()에서 alloc은 allocate의 약어이다. 하지만 이는 WordNet에는 존재하지 않고, 일반적인 자연어에도 존재하지 않으므로 자연어 구문 분석기는 *[(NP (JJ alloc) (NN Slice) (. .))]*와 같이 잘못된 품사 분석을 하게 된다. 이 이슈를 다루기 위하여 Caprile and Tonella[22]과 Madani et al.[1]은 축약어로부터 온전한 단어를 이끌어내기 위한 연구를 수행하였다. 하지만 C언어 혹은 C++와 같은 언어에 비하여 Java 언어는 약어 사용 비율이 적어 이로부터 발생하는 잘못된 분석은 상대적으로 적은 편이었다.

셋째, Java의 명명규칙을 위배하였으나, JDK(Java Development Kit)에서 주로 사용되었던 관용어구가 상당수 존재하였다. 예를 들면, 메소드 식별자인 *length(), size(), longVal(), intVal(), main(), keySet(), available()* 은 모두 명사(구) 혹은 형용사이므로 명명규칙에는 위배되나, Java 프로그램에서는 관용어에 속한다. 본 사례 연구에는 관용어보다는 명명 규칙의 위반에 가중을 두어 관용어는 비 일관적인 명명으로 간주하였다.

넷째, 다수결의 원칙의 한계가 존재한다. 다수결의 원칙은 '대부분 사용하는 의미와 품사가 정답이다'라고 간주한다. 하지만 set과 같이 동사뿐만 아니라 명사의 사용 빈도도 높은 단어의 경우는 정확하지 않은 검출을 하였다. Java 언어에서는 클래스 변수를 접근하기 위하여 동사 set을 많이 사용하고, 자료구조의 한 형태인 Set을 지칭하기 위하여 set을 명사 형태로도 많이 사용한다. 보통 동사의 set은 메소드에서 그리고 명사의 Set은 클래스에서 많이 사용되기 때문에 그 발견 빈도수가 클래스보다, 메소드에서 사용된 것이 현저하게 많다. 그래서 모든 클래스 이름의 Set은 다수결의 원칙에 따라서 품사 비 일관적인 식별자로 판단되었다.

마지막으로, 자연어 사전의 한계가 존재한다. 자연어 사전에는 컴퓨터와 응용 프로그램 도메인의 언어가 존재하지 않는다. 예를 들어 read는 동사이지만, 컴퓨터 도메인에서는 Read Only는 읽기 모드를 의미하는 고유 명사처럼 사용된다. 또한 rollback은 자연어 사전에는 명사의 의미만 존재하나, 컴퓨터 도메인에서는 동사의 의미를 가지기도 한다. 뿐만 아니라, 자연어에서 pop은 start와 유사하다고 생각되나 이는 stack의 조작 언어로 다른 단어로 바꾸어서는 변경하면 어색해진다. 사례 연구에서는 적절하지 않은 단어 의미상 유사어는 잘못된 검출로 간주하였다.

7. 결론 및 향후 연구

본 연구에서는 소스 코드 내의 비일관성 식별자를 검출하기 위하여 자연어처리 기법을 적용하였다. 본 연구에서는 비일관성에 대하여 의미상 유사어 비일관성, 구조상 유사어 비일관성, 구문상 품사 비일관성, 단어 품사 비일관성으로 구분하였다. 이를 자동으로 검출하기 위하여 자연어 구문 분석기인 Stanford Parser와 WordNet을 활용하였고, Levenshtein Distance 알고리즘을 적용하였다. 또한, CodeAmigo라는 지원 도구를 소개하였으며, 오픈 소스 프로젝트인 Apache Lucene과 Apache Ant에 본 접근 방법을 적용하여 본 연구의 정확함을 보였다. 향후 연구로, 6장에서 소개한 정확하지 않은 검출 문제 즉, 자연어 구문 분석기의 불완전함, 자연어 사전의 한계, 약어 및 관용어구가 존재하는 소스 코드의 특성을 극복하기 위한 방안을 연구하고자 한다. 이를 위하여 비일관성 식별자를 코드 식별자를 기반으로 온톨로지[6-23], 관용어구 사전 등을 추가적으로 연구하고자 한다. 또한 비일관 식별자를 줄이기 위한 가이드라인[4,10]을 삽입하여 코드 품질을 높이는데 기여하고자 한다.

참 고 문 헌

[1] N. Madani, L. Guerroju, M.D. Penta, Y. Gueheneuc and G. Antoniol, "Recognizing Words from Source Code Identifiers using Speech Recognition Techniques", In *Proceedings of 14th European Conference on Software Maintenance and Reengineering(CSMR)*, Madrid, Spain, pp.68-77, 2010.

[2] F. Deibenbock and M. Pizka, "Concise and Consistent Naming", In *Proceedings of International Workshop on Program Comprehension 2005(IWPC 2005)*, St. Louis, MO, USA, pp.261-282, 2005.

[3] D. Lawrie, H. Field and D. Binkley, "Syntactic Identifier Conciseness and Consistency", In *Proceedings of Sixth IEEE International Workshop on Source Code Analysis and Manipulation(SCAM2006)*, Philadelphia, Pennsylvania, USA, pp.139-148, Sept., 2006.

[4] 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.

[5] S.L. Abebe and P. Tonella, "Natural Language Parsing of Program Element Names for Concept Extraction", In *Proceedings of 18th International Conference on Program Comprehension (ICPC 2010)*, Braga, Minho, Portugal, pp.156-159, July, 2010.

[6] J. Falleri, M. Lafourcade, C. Nebut, V. Prince and M. Dao, "Automatic Extraction of a WordNet-like Identifier Network from Software", In *Proceedings of 18th International Conference on Program Comprehension (ICPC 2010)*, Braga, Minho, Portugal, pp.4-13, July, 2010.

[7] WordNet: A lexical database for English, Home page (2012), [Internet] http://wordnet.princeton.edu/

[8] D. Klein and C.D. Manning, "Accurate Unlexicalized Parsing", In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, Sapporo, Japan, pp.423-430, 2003.

[9] V.I Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *Soviet Physics Doklady*, Vol.10, No.8, pp.707-710, 1966.

[10] M. Fowler, "Refactoring: Improving the Design of Existing Code". Addison-Wesley, 1999.

[11] "Code Conventions for the Java Programming Language: Why Have Code Conventions", Sunmicro Systems (1999), [Internet]http://www.oracle.com/technetwork/java/index-135089.html

[12] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia and E. Merlo, "Recovering Traceability links between code and documentation.", *IEEE Transactions on Software Engineering*, Vol.28, No.10, pp.970-983, October, 2012.

[13] B. Caprile and P. Tonella, "Nomen Est Omen: Analyzing the Language of Function Identifiers", In *Proceedings of Sixth Working Conference on Reverse Engineering*, Altanta, Georgia, pp.112-122, 1999.

[14] The Stanford Parser Home page, 2012, [Internet] http://nlp.stanford.edu/software/lex-parser.shtml

[15] JAWS(Java API for WordNet Searching) Homepage, 2012, [Internet] http://lyle.smu.edu/~tspell/jaws/index.html

[16] JWI(The MIT Java WordNet Interface) Homepage, 2012, [Internet] http://projects.csail.mit.edu/jwi/

[17] HyperSQL Homepage, 2012, [Internet] http://www.hsqldb.org/

[18] Apache Lucene Homegage, 2012. [Internet] http://lucene.apache.org/core/,

[19] Apache Ant Homepage, 2012 , [Internet] http://ant.apache.org/

[20] W.B. Frakes and R. Baeza-Yates, "Information Retrival : Data Structures and Algorithms." Englewood Cliffs, J.J.: Prentice-Hall, 1992.

[21] J. Bloch, "Effective Java 2nd Edition", Addison-Wesley, 2008.

[22] B. Caprile and P. Tonella. "Restructuring program identifier names". In *Proceedings of 16th International Conference on Software Maintenance(ICSM 2000)*, San Jose, California USA, pp.97-107, Oct., 2000.

[23] E. Host and B. Ostvold, "The Programmer's Lexicon, Volumn I: The Verbs", In *Proceedings of Seventh IEEE International Working Conference on Source Code Analysis and Manipulation(SCAM2007)*, Paris France, pp.193-202, 2007.

이 성 남

e-mail : dapalee@korea.kr

1982년 공군사관학교(학사)

1990년 국방대학원(석사)

1993년~1996년 KF-16 전투기 SW 연구 록히드마틴(미국)

2004년~2005년 공군 항공소프트웨어소장

2006년~현 재 방위사업청 과장

관심분야 : Weapon System Software

김 순 태

e-mail : stkim@kangwon.ac.kr

2003년 중앙대학교 컴퓨터공학과(학사)

2007년 서강대학교 컴퓨터공학과(석사)

2010년 서강대학교 컴퓨터공학과(박사)

2011년~현 재 강원대학교 컴퓨터공학과 조교수

관심분야 : 소프트웨어 아키텍처, 디자인 패턴, 요구공학, Mining Software Repository

박 수 용

e-mail : sypark@sogang.ac.kr

1986년 서강대학교 전자계산학(학사)

1988년 Florida State University(석사)

1995년 George Mason University(박사)

1996년~1998년 TRW Senior Software Engineer

1998년~현 재 서강대학교 컴퓨터공학과 교수

관심분야 : 적응형 소프트웨어, 소프트웨어 프로덕트 라인, 요구공학