논문 2017-1-4

소스코드 유사도 측정 도구의 성능에 관한 비교연구

김규식*, 조성제*+, 우진운**

A Comparative Study on the Performance of Source Code Similarity Measurement Tools

Gyoosik Kim*, Seong-je Cho*+, Jinwoon Woo**

요 약

소프트웨어 사용이 일반화되고 우리 일상생활에서 유용한 역할을 함에 따라, 소프트웨어 불법복제나 표절 행위가 여전히 많이 발생하고 있다. 이에 소스코드 수준에서 불법복제나 표절 여부를 탐지하기 위한 도구들이 개발되어 사용 중이다. 대표적인 도구로 JPlag, MOSS, exEyes 등이 있다. 본 논문에서는 대표적인 코드 클론(clone)들의 유형을 정리하고, 자바 프로그램들을 대상으로 JPlag와 exEyes의 성능을 비교 분석한다. 성능 비교를 위해서는 코드 클론들의 모음인 BigCloneBench 벤치마크를 사용하며, 평가 항목으로는 정확도와 재현율, F-measure를 사용한다. 실험 결과, 두 도구들이 단순한 코드 클론 유형들에 대해서는 두 도구 모두 100% 탐지율을 보였다. 그러나 복잡한 코드 클론 유형에 대해서, JPlag는 95%의 정확도와 50.1%의 재현율을, exEyes는 100%의 정확도와 27.7%의 재현율을 보였다.

Abstract

Because software is everywhere and has continued to play a major part in our daily lives, software piracy and plagiarism are still rampant. The tools has been developed and used in order to detect pirated copies of software at source code level. JPlag, MOSS, and exEyes are well-known source code plagiarism detectors. In this paper, we explain briefly different types of classical code clones, and compare the performance of JPlag and exEyes using Java source code files. The performance comparison between JPlag and exEyes are conducted using a benchmark, BigCloneBench which is a big benchmark of real clones. We evaluate the performance of the tools measuring their precision, recall, and F-measure. Experimental results show that both precision and recall rates of the two tools become 100% for simple type-1 and 2 clones. For more complicated type-3 clones, however, JPlag and exEyes have a precision rate of 95% and 100%, and a recall rate of 50.1% and 27.7% respectively.

한글키워드 : 소프트웨어 표절, 코드 클론, JPlag, exEyes, 재현율, 정확도, F-measure

※ 이 논문은 한국소프트웨어감정평가학회 2017 춘계학술대회에 발표한 '소스코드 유사도 측정 도구에 대한 고찰' 발표자료를 확장한 것임.

* 단국대학교 컴퓨터학과
** 단국대학교 소프트웨어학과
+ 조성제 (교신저자) (email: sjcho@dankook.ac.kr)

접수일자 : 2017.06.20 수정완료 : 2017.06.27

※이 논문은 2016년 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(no. NRF- 2015R1D1A1A02061946), 그리고 미래창조과학부 및 한국인터넷진흥원의 "고용계약형 정보보호 석사과정 지원사업"의 연구결과로 수행되었음 (과제번호 H2101-17-1001)

1. 서 론

양질의 소프트웨어를 개발하기 위해서는, 전문 인력들이 투입되어 요구사항 분석, 설계, 구현, 테스팅, 유지보수 등의 개발주기를 반영하여야 하며, 이를 위해서는 많은 시간과 비용이 소요된다. 또한, 회사나 기관의 업무 처리가 소프트웨어로 처리되므로, 소프트웨어에는 핵심 알고리즘 등의 지적재산권과 회사의 영업비밀 정보가 포함되기도 한다. 이처럼, 소프트웨어 가치는 매우 높다고 할 수 있다. 소프트웨어 가치가 증대되면서, 적은 비용으로 개발시간 단축, 시스템의 복잡성 증대, 개발자 부족 등의 이유로 소프트웨어를 임의로 복제하거나 표절하여 불법 사용하는 경우가 많이 발생하고 있다.

2017년 3월 21일자 Revulytics의 소프트웨어 불법복제와 라이선스 오용 관련 보고서[1]에 따르면, 불법 복제된 또는 라이선스가 없는 소프트웨어를 사용하는 국가 1위는 중국, 2위는 미국, 3위는 이란, 4위는 러시아, 5위가 인도이다. 한국은 10위로 2015년에 비해 4단계 상승하였다. 또한, BSA|The Software Alliance의 BSA Global Software Survey (May 2016)[2]에 따르면, 2015년 전 세계 컴퓨터에 설치된 소프트웨어 중 39%가 라이선스 없이 무단으로 사용되고 있다.

한국저작권위원회의 정보자료(저작권통계 중 저작권 분쟁조정 자료)에 의하면, 어문/음악/미술/영상 저작물에 비해 컴퓨터프로그램의 분쟁조정 건이 많다. 일부 학생들도 프로그래밍 언어 관련 학습 과제물을 수행하면서 표절 행위를 하고 있다고 보고되고 있다. 이처럼, 소프트웨어 표절이나 라이선스 위배로 인한 피해가 심각하다.

소프트웨어 표절의 경우, 그 유형이 다양하고 소스코드 표절 범위도 프로그램의 전체 또는 일부일 수 있다. 규모가 작은 소프트웨어 경우, 개발자나 전문가가 수작업으로 표절이나 복제 여부를 판단할 수 있지만, 규모가 큰 소프트웨어의 경우 수작업으로 표절 여부를 판단하는 것이 매우 어렵다. 이에 자동화된 코드 표절 검사 도구들이 개발되어 사용되고 있다. 대표적인 도구로는 JPlag[1,2], exEyes[3], MOSS(Measure of Software Similarity)[4,5] 등이 있다.

본 논문에서는 자바 프로그램들을 대상으로 두 표절 탐지 도구인 JPlag와 exEyes의 성능을 비교 분석한다. 코드 클론(code clone)들의 유형을 정리하고, 소스코드 클론 탐지 방법을 평가하는데 사용되는 벤치마크에 대해 기술한다. 본 논문에서는 표절탐지 도구의 성능을 평가하기 위해 BigCloneBench[6,7]라는 벤치마크를 사용한다.

본 논문의 구성은 다음과 같다. 2장에서는 코드 클론의 대표적인 유형 4가지에 대해 설명하고, 3장에서는 관련 연구에 대해 간단히 기술한다. 4장에서는 소스코드들의 유사도 측정 도구인 JPlag와 exEyes의 개요에 대해 설명한다. 5장에서는 BigCloneBench라는 벤치마크 기반의 실험을 통해 두 도구의 성능을 확인한다. 마지막으로 6장에서는 결론을 맺고 향후 연구방향에 대해 설명한다.

2. 코드 클론의 유형

본 논문에서 코드 클론(code clone)[6,8,9], 또는 동일 코드 조각(identical code snippets)은 두 개의 소스코드 내에 존재하는 일치하거나 유사한 코드 조각(code fragment)을 말한다. Roy 등[8],

1) Revulytics, Top 20 Countries for Software Piracy and Licence Misuse (2017), March 21, 2017. www.revulytics.com/blog/top-20-countries-software-piracy-2017

2) BSA|The Software Alliance, Seizing Opportunity Through License Compliance, 2016. http://globalstudy.bsa.org/2016/

Bellon 등[9], Rattan 등[15]은 코드 클론을 다음과 같이 4가지 유형으로 정의하고 있다.

2.1 클론 유형 1

주석의 내용 및 위치, 공백문자(탭, 줄 바꿈, 스페이스 등), 레이아웃을 제외하고 모두 일치하는 코드 조각을 나타낸다. Exact clone으로 널리 알려져 있다. 전형적인 라인 단위의 검사 기법은 레이아웃에 변화가 있는 클론을 탐지하지 못할 수도 있다. 유형 1의 클론을 *exact clones*이라고도 한다.

<표 1> 클론 유형 1 [8]

Original code segment	Copy clone
if (a >= b) { c = d + b; //Comment 1 d = d + 1; } else c = d - a; //Comment 2	if (a >=b) { **//Comment1'** c=d+b; d=d+1; } else **//Comment2'** c=d-a;

2.2 클론 유형 2

사용자 정의 식별자(변수/상수/클래스/메소드 이름 등), 타입과 레이아웃, 주석, 표현식의 변경을 제외하고 모두 일치하는 코드 조각을 의미한다. 예약어(reserved words)와 문장 구조는 동일하다. 아래 <표 2>를 보면, 변수 이름과 값 할당 등에서 변경이 있지만, 구문 구조(syntactic structure)는 여전히 비슷하다. 유형 2의 클론을 r*enamed/parameterized clones*이라고도 한다.

2.3 클론 유형 3

함수의 매개변수 변경, 문장 변경(줄의 삽입/삭제/수정) 등과 같이 코드(또는 문장)가 일부 추가되거나 제거된 경우를 말한다. <표 3>을 보면, “e = 1”이라는 문장이 추가되었다.

<표 2> 클론 유형 2 [8]

원본 코드 조각 (Original)	if (**a >= b**) { **c = d + b**; // Comment1 **d = d + 1**;} else **c = d - a**; //Comment2
Copy clone	if (**m >= n**) { // Comment1' **y = x + n**; **x = x + 5**;//Comment3 } else **y = x - m**; //Comment2'

<표 3> 첫 번째 클론 유형 3 [8]

원본 코드 조각 (Original)	if (**a >= b**) { **c = d + b**; // Comment1 **d = d + 1**;} else **c = d - a**; //Comment2
Copy clone	if (a >= b) { c = d + b; // Comment1 **e = 1; // This statement is added** d = d + 1; } else c = d - a; //Comment2

또 다른 유형 3의 클론이 <표 4>에 나타나 있으며, 두 코드 조각의 차이점이 <표 5>에 나타나 있다. 첫 번째 줄에 삽입된 단어(*synchronized*)를 제외하고는 클론 유형 2라고 볼 수 있다. 유형 3의 클론을 *near miss clones*이라고도 한다.

<표 4> 두 번째 클론 유형 3
Cloned methods in JDK [8]

원본	public int **getSoLinger()** throws SocketException{ Object o = impl.getOption(SocketOptions.SO_LINGER); if (o instanceof Integer) { return((Integer) o).intValue(); } else return **-1**; }
코드 클론	public **synchronized int getSoTimeout()** throws SocketException{ Object o = impl.getOption(SocketOptions.**SO_TIMEOUT**); if (o instanceof Integer) { return((Integer) o).intValue(); } else return **-0**; }

<표 5> <표 4>의 두 코드 조각의 차이점 [8]

원본	코드 클론	상태
ε	synchronized	Insertion
<getSoLinger>	<getSoTimeout>	Replacement
<SO_LINGER >	<SO_TIMEOUT>	Replacement
<-1>	<-0>	Replacement

2.4 클론 유형 4

같은 계산을 수행하지만 다른 구문적 변형(syntactic variants, 구문 변형)을 통해서 구현될 수 있는 두 개 이상의 코드 조각을 말한다. 이는 두 개 이상의 코드 조각들 사이에 의미적 유사성(semantic similarity)을 있는 경우이다. 유형 4에서는 클론 조각이 원본 조각으로부터 복제되었음을 의미하지 않을 수 있다. 두 코드 조각은 두 명의 다른 프로그래머에 의해 개발될 수도 있다. 기능성 유사성(functional similarity)은 컴포넌트들이 비슷하게 동작하는 정도를 반영한다.

<표 6>의 원본에서 j의 최종 값은 변수 VALUE의 계승 값(factorial value)이다. <표 6>의 두 코드 조각은 의미론적면에서 기능이 유사하다. 유형 4의 클론을 *semantic clones* 이라고도 한다.

<표 6> 클론 유형 4 [8]

원본	코드 클론
반복문을 이용한 factorial 구현	재귀함수를 이용한 factorial 구현
int i, j=1; for (i=1; i<=VALUE; i++) j=j*i;	int factorial(int n) { if (n == 0) return 1 ; else return n * factorial(n-1) ; }

2.5 코드 클론 정리

코드 클론 유형 1~3은 원문/본문 유사성(textual similarity)을 나타내고, 유형 4는 기능적 유사성(functional similarity)을 나타낸다. 유형 4를 의미론적 클론(semantic clones)이라고도 한다. 이러한 클론 유형은 유형 1로부터 유형 4로 갈수록 교묘함(중요한 세부 요소)이 증대되며, 유형 4로 갈수록 클론 탐지가 복잡하다는 것을 의미한다. 특히, 유형 4의 경우 프로그램 구성이나 소프트웨어 설계에 대한 배경지식이 많더라도 탐지가 어렵다.

이 외에도, 구조적 클론(structural clone: 아키텍처 수준에서 설계와 분석 시에 나타나는 상호 연관된 패턴으로, 유지보수에 도움을 주는 설계 단계의 유사성을 반영), 함수 클론(function clone: 함수/메소드 또는 프로시저 규모의 클론), 모델 기반 클론(model based clone: 시스템 개발을 위한 핵심 산출물로 그래픽 언어가 코드를 대체하고 있는데, 모델에서의 예상치 못한 중첩이나 중복을 의미) 등이 있다.

3. 관련 연구

Bellon 등[9]은 클론 탐지 도구들을 평가하기 위한 벤치마크로 Bellon reference corpus를 제안하였다. Bellon reference corpus는, 8개의 주제 시스템[3](subject systems)에 대해 6개의 클론 탐지 도구들의 결과를 분석하여 수작업으로 구축한 클론들의 모임이다[10]. 8개의 주제 시스템은 C

3) 코드 클론 탐지 기법의 검증을 위해 사용되는 시스템을 주제 시스템(subject system)이라고 한다. 연구자들은 동일 시스템 상에서 연구를 수행하여 결과를 더 의미 있게 상호 비교한다. 주제(subject)의 대표적인 예는 (1) Computing and processing (HW & SW), (2) General topics for engineers (Math, Science, and Engineering) 등이다 [15]. 코드 클론 탐지를 위해 상용 subject system을 사용하기도 하지만, 주로 오픈소스 소프트웨어 시스템을 사용한다. Subject system의 대표적인 예는 JDK (3200 KLOC, Java), Appache-httpd (343 KLOC, C), Apache-Ant (1.41 MLOC, Java), Linux (6.2 MLOC, C), Netbeans-javadoc (19 KLOC, Java), Bison (16 KLOC, C), PostgreSQL (937 KLOC, C) 등이다 [15].

로 작성된 4개 프로그램들(weltab, cook, snns, postgresql) 및 Java로 작성된 4개 프로그램들(netbeans-javadoc, eclipse-ant, eclipse-jdtcore, j2sdk 1.4.0-javax-swing)이며, 사용된 6가지 도구는 Dup, CloneDR, CCFinder, Duplix, CLAN, Duploc이다. Reference corpus에는 문자열기반/토큰기반/트리기반/계량치기반/의존그래프기반 클론 등이 포함되어 있다. Bellon reference corpus의 문제점은, 사용된 클론 탐지 도구들이 탐지할 수 있는 클론들만을 포함하고 있다는 것이다[6]. 즉, 사용된 탐지 도구들이 탐지하지 못한 코드 클론을 제공하지 못해, 재현율(recall)에서 과대평가될 수 있다. 또한 Bellon의 벤치마크는 특정한 코드 조각에 대해 매우 적은 수의 클론을 제공한다.

최성하 등[3, 10]은 클론 탐지 도구들을 평가하기 위해 Bellon reference corpus를 사용하였다. [3]에서는 Bellon reference corpus를 사용하여 exEyes의 성능을 평가하였다. 성능 평가 지표로는 재현율(recall)과 정확도(precision)를 사용하였다.

$$\text{정확도} = \frac{\text{도구가 탐지한 클론 중 정확한 클론의 개수}}{\text{도구가 클론이라고 탐지한 코드의 총 개수}}$$

$$\text{재현율} = \frac{\textit{Reference Corpus} \text{ 클론 중 도구가 탐지한 실제 클론의 개수}}{\textit{Bellon Reference Corpus} \text{ 클론의 총 개수}}$$

[3]에서 exEyes의 성능 평가 결과, 재현율은 173/898 = 19.3%, 정확도는 779/781 = 99.7% 로 나타났다. 세부적인 실험 결과가 <표 7>에 나타나 있으며, 재현율이 낮게 측정되었음을 알 수 있다. [3]에서는 벤치마크로 BigCloneBench를 사용하지 않았으며, JPlag 도구에 대한 성능을 평가하지도 않았다.

<표 7> Bellon reference corpus를 사용한 exEyes v4.1의 실험 결과 [3]

	JScrollBar.java	JTable.java	JToolBar.java	JTree.java
JList.java	재현율: 4/7 정확도: 34/34	재현율: 34/299 정확도: 157/157	재현율: 3/4 정확도: 45/45	재현율: 30/258 정확도: 171/173
JScrollBar.java	-	재현율: 2/2 정확도: 29/29	재현율: 3/3 정확도: 18/18	재현율: 0/1 정확도: 22/22
JTable.java	-	-	재현율: 1/2 정확도: 12/12	재현율: 95/321 정확도: 279/279
JToolBar.java	-	-	-	재현율: 1/1 정확도: 12/12

[10]에서는 클론 유형 1과 2에 대해 Bellon reference corpus의 일부 문제점을 보정하였다. 즉, 잘못된 클론 유형으로 ① 클론의 범위가 부족한 클론, ② 클론을 포함하는 클론, ③ 잘못 탐지된 클론을 제시하였다. Bellon reference corpus에서 실제 클론보다 작은 범위의 클론만 포함되어 있는 경우, 실제 클론을 탐지했는데도 클론을 탐지하지 못한 것으로 처리될 수 있어 실제보다 재현율이 떨어질 수 있다. 보정을 통해, postgresql과 eclipse-ant에서 각각 2,410개, 48개의 잘못된 클론을 보정하였다.

Svajlenko 등은 Bellon reference corpus보다 많은 코드 클론을 제공하는 BigCloneBench[6] 및 Big Clone Eval[11]을 제안하였다. BigCloneBench는 IJaDataset 2.0 (25000 subject system, 365MLOC)라는 big data inter-project repository로부터 발굴한(mined) 진짜 클론(true clone)과 가짜 클론(false clone)들의 벤치마크다. 클론 탐지기를 사용하지 않고, 자주 구현되는 기능(functionality)들의 클론들을 수집하기 위해 IJaDataset을 마이닝하였다. 실제 대상 기능

(target functionality)을 구현하는 코드 조각(code snippets)을 자동으로 식별하기 위해 탐색 휴리스틱(search heuristics)을 사용하였다. 후보 조각은 감정가(judge)의 수작업에 의해 대상 기능에 true positive 또는 false positive로 태그를 붙이고, 구문적 유사성(syntactical similarity)을 측정하였다.

4. 소프트웨어 유사도 측정도구

코드 클론은 실제 프로그램 개발에 유용하게 사용될 수도 있고, 또한 타인의 코드를 불법 도용될 수도 있다. 이에, CCFinder, CloneDR, Covet 등의 클론 탐지 도구들[15]도 있고, JPlag, MOSS, exEyes 등의 표절 탐지 도구들도 있다. 본 논문에서는 표절 탐지기인 JPlag와 exEyes에 대해서만 다룬다.

4.1 JPlag v2.11.9 [1,2,12,13]

1997년에 개발된 소스코드 표절 탐지 도구이다[12]. 비교 대상 프로그램들의 집합을 입력으로 받아, 이들 프로그램들을 쌍으로 비교하여(각 쌍에 대해 전체 유사도 값과 유사성 영역들의 집합을 계산하여), 유사한 코드를 검색하고 이해할 수 있도록 HTML 페이지들을 출력한다. JPlag는 소스코드의 몇 번째 줄부터 몇 번째 줄까지 클론이라는 것을 텍스트로 표시하여 준다.

클론 탐지를 위해 토큰화(tokenization) 기법을 사용하는 표절 탐지기(plagiarism detector)라고도 한다. 각 프로그램을 표준적 토큰(canonical tokens)들의 연속(stream)으로 변환하고, 하나의 토큰 스트링이 다른 토큰에서 취한 서브스트링에 의해 커버될 수 있게 한다. 대상 파일을 토큰화된 문자열(tokenized string)으로 변환하기 전에, 공백과 주석을 제거하는 전처리 과정을 수행한다. 이를 통해, 모든 어휘 변경(lexical changes)을 무효화할 수 있으며, 좋은 토큰 집합은 많은 구조적 변경(structural changes)의 효험을 경감시킬 수 있다.

제출된 모든 파일들을 다-대-다 비교를 수행하여 유사도(similarity score)에 의해 정렬된 리스트를 출력한다. 이 리스트를 이용하여, 어떤 쌍이 표절을 포함하고 있는 가를 판단할 수 있다. N을 모임들(collection)의 크기(파일의 개수), $f(n)$을 길이가 n인 두 파일로 된 쌍들 사이들 간의 비교하는데 소요되는 시간이라고 할 때, 구현 복잡도는 $O(f(n)N^2)$이다 [13].

4.2 exEyes [3]

한국저작권위원회가 자체 개발한 GUI 기반 소프트웨어 표절 검사 도구로, 소프트웨어 감정평가 프로그램으로 지원하는 언어는 Java, C, C++, VB, Text(Web) 등이다. 소프트웨어 소스코드 유사도 감정에 사용되고 있는 감정도구이기도 하다. 코드 클론 탐지 또는 소프트웨어 유사도 측정 시에 선택할 수 있는 기능은, 동일/유사한 블록의 크기, 유사라인의 동일 토큰수 비율, 유사라인의 최소 토큰 수, 유사라인의 최대 토큰 비, 확장자 구분 등이다.

5. 실험 및 평가

5.1 BigCloneBench 벤치마크 [6,11]

Java 언어로 작성된 IJaDataset-2.0 내에 6,260,000개의 유효성이 검증된 클론들의 모음이다. IJaDataset 2.0은 25,00개의 Java 오픈소스 데이터 저장소로, 2014년도 BigCloneBench 버전에

는 6,000,000개의 진짜 클론 쌍들과 260,000개의 가짜 클론 쌍들이 포함되어 있으며, github.com/clonebench/BigCloneBench URL에서 획득할 수 있다.

소스코드 유사도 측정 도구들의 평가를 위한 데이터 집합, 유형 1~3으로 다음 <표 8>과 같다. 코드 클론 유형 4에 대해서는 실험하지 않았다.

<표 8> 실험에 사용된 BigCloneBench의 클론 유형

	유형 1	유형 2	유형 3
True clone	120	53	493
False clone	0	2	73

5.2 평가 항목

평가항목으로는 재현율과 정확도, F-measure를 정의하여 고려하였다[14]. 정확도는 정상 탐지된 요소 수를 정상이라고 분류된 요소들의 총 개수로 나눈 것으로 FP (False Positive)가 0일 때 최대가 된다. 재현율은 정상 탐지된 요소 수를 실제 정상 부류에 속한 요소들의 총 개수로 나눈 것으로 FN(False Negative)이 0일 때 최대가 된다.

<표 9> 정확도와 재현율 설명

용어	설명	별칭
정확도 (precision)	How many selected items are relevant?	Positive predictive value
재현율 (recall)	How many relevant items are selected?	Sensitivity

"정확도"는 도구가 (실제 클론이 아닌 것 포함하여) 클론이라고 탐지한 코드 조각들 중에서 정확한 클론의 개수를 의미하며, "재현율"은 무작위로 선정된 실제 클론들 중에서 도구에 의해 성공적으로 탐지된 클론의 개수를 의미한다. 정확도와 재현율을 아래와 같이 표현할 수 있다.

<표 10> 정확도와 재현율 관련 용어 설명

용어	설명
TP (True Positive)	The number of Predicted Positives that were correct (정상으로 탐지 - 정탐)
FP (False Positive)	The number of Predicted Positives that were incorrect (클론이 아닌 것을 클론으로 분류/탐지) Type I error
TN (True Negative)	(실제 클론을 탐지하지 못 함 - 미탐)
FN (False Negative)	(실제 클론을 클론이 아니라고 분류/탐지) Type II error

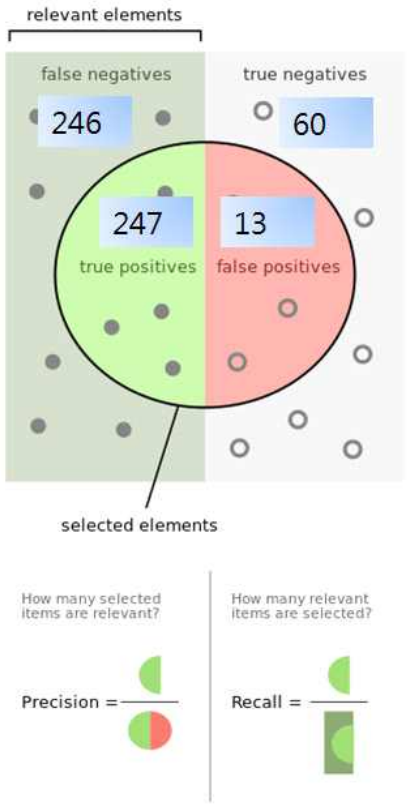
- 정확도(precision) = $\frac{TP}{TP+FP}$
- 재현율(recall) = $\frac{TP}{TP+FN}$

F-measure는 정확도와 재현율의 조화평균(harmonic mean)으로 다음과 같다.

- $F = \frac{2\times(\text{precision}\times\text{recall})}{(\text{precision}+\text{recall})}$

5.3 JPlag v2.11.9의 성능 평가

JPlag 설치 후, Java 1.7 소스코드 대상으로 유사도를 측정하였다. 클론 유형 1과 2에 대해서는 우수한 성능을 보이지만, 클론 유형 3의 재현율은 50.1%였다. 클론 유형 3에 대한 정확도와 재현율 세부 사항은 다음 <그림 1>과 <표 11>에 나타나 있다. JPlag가 클론 유형 3에 대해 정상 탐지한 예가 <그림 3>에, 탐지하지 못 한 예가 <그림 4>에 나타나 있다.



<그림 1> 클론 유형 3에 대한 JPlag 탐지 결과(원 그림 출처: wikipedia.org)

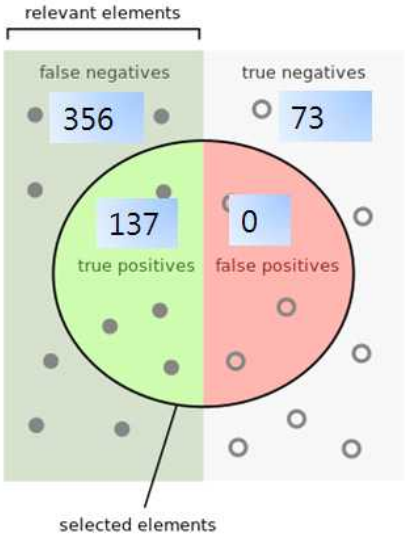
<표 11> JPlag v2.11.9의 성능 평가 결과

	유형 1	유형 2	유형 3
정확도	120/120 (100%)	53/53 (100%)	247/260 (95%)
재현율	120/120 (100%)	53/53 (100%)	247/493 (50.1%)

클론 유형 3에 대한 JPlag의 F-measure는 2*(95*50)/(95+50) = 65.5이다.

5.4 exEyes 5.0의 성능 평가

기본 비교 설정으로 간단하게 실험을 진행하였다. 유사 판정 기준은 3줄 이상, 3개 토큰 이상, 최대 토큰비 2.0으로 설정하여 유사율이 80% 이상이면 클론으로 판정하였다. 실험 결과가 <표 10>에 나타나 있다. exEyes 5.0이 전반적으로 코드 클론을 잘 탐지하고 있지만, 유형 3에 대한 재현율은 27.7%로 낮은 편이다. 이는 JPlag에 비해 FN가 높게 나왔기 때문이다.



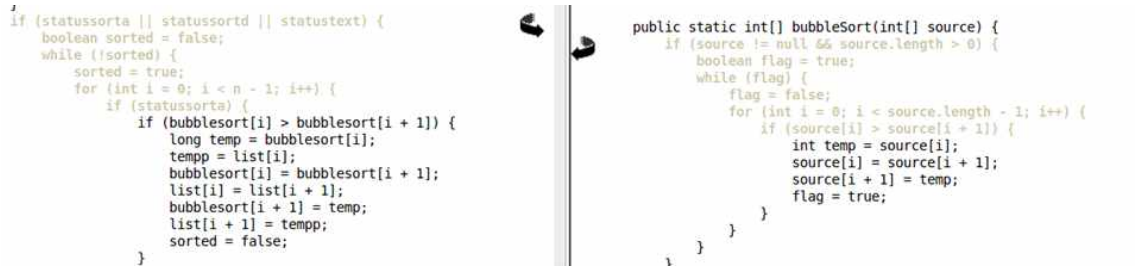
<그림 2> 클론 유형 3에 대한 exEyes 탐지 결과(원 그림 출처: wikipedia.org)

<표 12> exEyes 5.0의 성능 평가 결과

	유형 1	유형 2	유형 3
정확도	120/120 (100%)	53/53 (100%)	137/137 (100%)
재현율	120/120 (100%)	53/53 (100%)	137/493 (27.7%)

exEyes가 클론 유형 3에 대해 정상 탐지한 예가 <그림 5>에, 탐지하지 못 한 예가 <그림 6>에 나타나 있다.

실험에 사용된 코드 클론 유형 3에 대해서는 JPlag와 exEyes가 공통적으로 낮은 재현율을 보였다. 실험에 사용된 벤치마크 및 클론 유형을 고려하지 않고, 본 논문의 실험 결과와 [3]의 연구 결과와 단순 비교하여 보면, 전체 재현율은 310/666 = 46.5%로 두 배 이상 많이 개선되었으

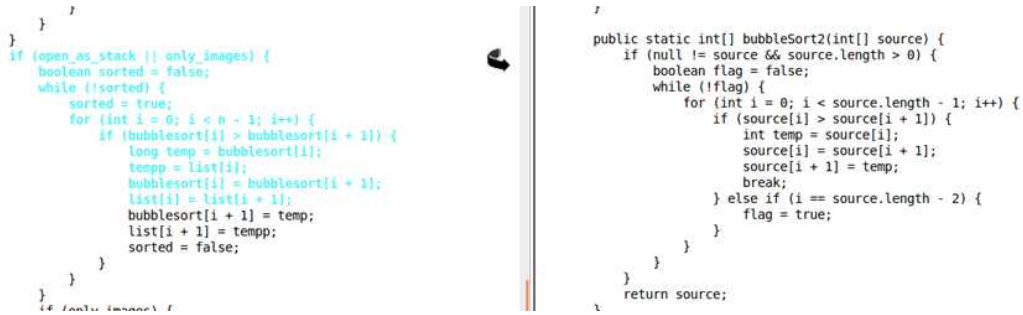


```
if (statussorta || statussortd || statustext) {
    boolean sorted = false;
    while (!sorted) {
        sorted = true;
        for (int i = 0; i < n - 1; i++) {
            if (statussorta) {
                if (bubblesort[i] > bubblesort[i + 1]) {
                    long temp = bubblesort[i];
                    tempp = list[i];
                    bubblesort[i] = bubblesort[i + 1];
                    list[i] = list[i + 1];
                    bubblesort[i + 1] = temp;
                    list[i + 1] = tempp;
                    sorted = false;
                }
```

```
public static int[] bubbleSort(int[] source) {
    if (source != null && source.length > 0) {
        boolean flag = true;
        while (flag) {
            flag = false;
            for (int i = 0; i < source.length - 1; i++) {
                if (source[i] > source[i + 1]) {
                    int temp = source[i];
                    source[i] = source[i + 1];
                    source[i + 1] = temp;
                    flag = true;
                }
            }
        }
```

(a) DICOM_Sort.java 116-261 (b) SimpleCricketDartGame.java 34-65

<그림 3> 클론 유형 3에 대한 JPlag v2.11.9의 정탐 예

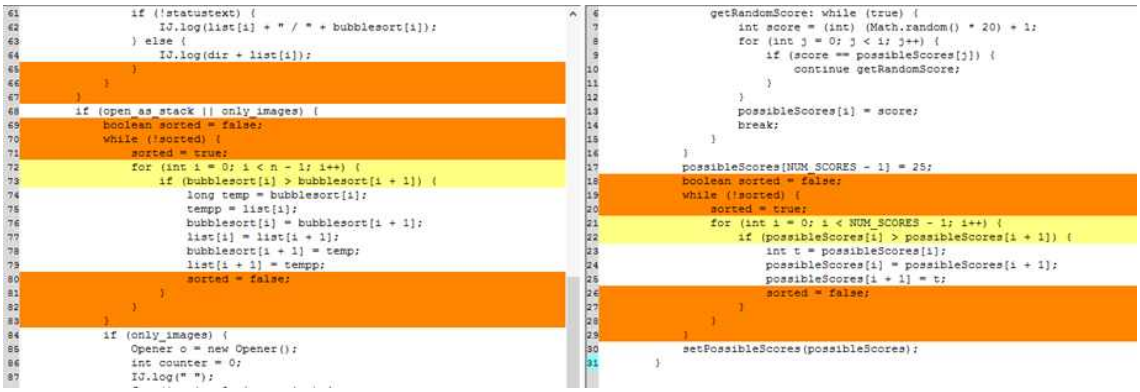


```
    }
}
if (open_as_stack || only_images) {
    boolean sorted = false;
    while (!sorted) {
        sorted = true;
        for (int i = 0; i < n - 1; i++) {
            if (bubblesort[i] > bubblesort[i + 1]) {
                long temp = bubblesort[i];
                tempp = list[i];
                bubblesort[i] = bubblesort[i + 1];
                list[i] = list[i + 1];
                bubblesort[i + 1] = temp;
                list[i + 1] = tempp;
                sorted = false;
            }
        }
    }
```

```
public static int[] bubbleSort2(int[] source) {
    if (null != source && source.length > 0) {
        boolean flag = false;
        while (!flag) {
            for (int i = 0; i < source.length - 1; i++) {
                if (source[i] > source[i + 1]) {
                    int temp = source[i];
                    source[i] = source[i + 1];
                    source[i + 1] = temp;
                    break;
                } else if (i == source.length - 2) {
                    flag = true;
                }
            }
        }
    }
    return source;
}
```

(a) DICOM_Sort.java 116-261 (b) Task2.java 34-65

<그림 4> 클론 유형 3에 대한 JPlag v2.11.9의 미탐 예

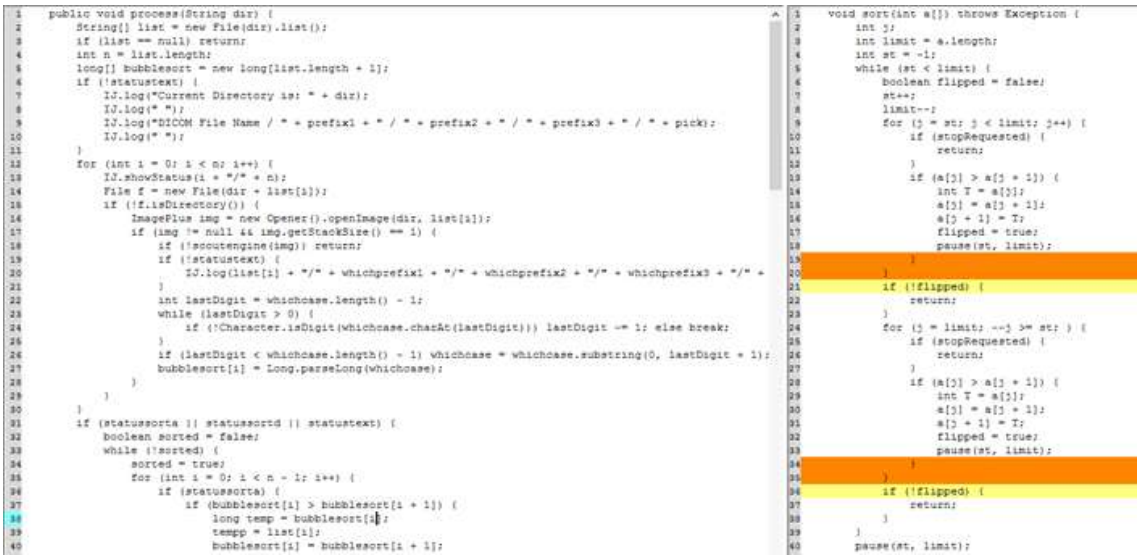


```
61              if (!statustext) {
62                  IJ.log(list[i] + " / " + bubblesort[i]);
63              } else {
64                  IJ.log(dir + list[i]);
65              }
66          }
67      }
68      if (open_as_stack || only_images) {
69          boolean sorted = false;
70          while (!sorted) {
71              sorted = true;
72              for (int i = 0; i < n - 1; i++) {
73                  if (bubblesort[i] > bubblesort[i + 1]) {
74                      long temp = bubblesort[i];
75                      tempp = list[i];
76                      bubblesort[i] = bubblesort[i + 1];
77                      list[i] = list[i + 1];
78                      bubblesort[i + 1] = temp;
79                      list[i + 1] = tempp;
80                      sorted = false;
81                  }
82              }
83          }
84          if (only_images) {
85              Opener o = new Opener();
86              int counter = 0;
87              IJ.log(" ");
```

```
6               getRandomScore: while (true) {
7                   int score = (int) (Math.random() * 20) + 1;
8                   for (int j = 0; j < i; j++) {
9                       if (score == possibleScores[j]) {
10                          continue getRandomScore;
11                      }
12                  }
13                  possibleScores[i] = score;
14                  break;
15              }
16          }
17          possibleScores[NUM_SCORES - 1] = 25;
18          boolean sorted = false;
19          while (!sorted) {
20              sorted = true;
21              for (int i = 0; i < NUM_SCORES - 1; i++) {
22                  if (possibleScores[i] > possibleScores[i + 1]) {
23                      int t = possibleScores[i];
24                      possibleScores[i] = possibleScores[i + 1];
25                      possibleScores[i + 1] = t;
26                      sorted = false;
27                  }
28              }
29          }
30          setPossibleScores(possibleScores);
31      }
```

(a) DICOM_Sort.java 일부 (b) SimpleCricketDartGame.java 일부

<그림 5> 클론 유형 3에 대한 exEyes 5.0의 정탐 예



(a) DICOM_Sort.java 일부 (b) BubbleSortAlgorithm.java 일부

<그림 6> 클론 유형 3에 대한 exEyes 5.0의 미탐 예

며, 전체 정확도도 100%로 높게 나왔다. 이유 중의 일부는 exEyes의 버전 업그레이드와 본 논문에 사용된 벤치마크의 특성 때문이다.

클론 유형 3에 대한 exEyes의 F-measure는 2*(100*27.7)/(100+27.7) = 43.4로 JPlag의 F-measure보다 낮다. 본 논문에서 수행된 간단한 실험 결과, exEyes의 경우 클론 유형 3에 대한 재현율을 개선할 필요가 있다는 것을 알 수 있다.

6. 결론 및 향후 연구방향

코드 재사용, 개발 기간 단축 등으로 코드 클론을 사용할 수도 있지만, 이는 지적재산권 침해, 버그 전파, 설계 단계의 악영향 등을 유발할 수 있다. 본 논문에서는 대표적인 코드 클론 유형 4가지를 살펴보고, BigCloneBench 벤치마크에서 제공하는 일부 Java 코드 클론을 사용하여 두 개의 소스코드 유사도 측정 도구들의 성능을 평가하였다. 실험에 사용된 유사도 측정 도구는 JPlag v2.11.9와 exEyes 5.0이며, 평가 지표는 재현율과 정확도, F-measure이다. 두 개의 도구 모두 클론 유형 1과 2에 대해서는 100%의 탐지율을 보였다. 그러나 실험에 사용된 클론 유형 3에 대해서는 높지 않은 재현율을 보였다. 특히, exEyes 5.0의 경우에는 27.7%의 재현율을 보였다. 이를 통해, 실험에 사용된 클론 유형 3의 대표성을 확인할 수는 없지만, 도구들이 클론 유형 3을 정확하게 탐지하는 것이 어렵다는 것을 알 수 있다.

향후에는 BigCloneBench의 좀 더 다양한 코드 클론들을 사용하여, 소스코드 유사도 측정 도구들의 성능을 평가할 예정이다. 그리고 C 언어 기반의 코드 클론들에 대해서도 실험을 진행할 계획이다.

참 고 문 헌

[1] Prechelt, L., Malpohl, G., Philippsen, M. “Finding plagiarisms among a set of programs with JPlag”, Journal of Universal Computer Science, 2002, vol. 8, no. 11, pp.1016-1038.

[2] Malpohl, G. “JPlag: detecting software plagiarism”, URL *https://jplag.ipd.kit.edu*, 2006.

[3] 최성하, 도경구 “exEyes의 성능평가” 2014년 한국소프트웨어감정평가학회 추계학술대회, pp. 13-21.

[4] Bowyer, K. W., Hall, L. O. “Experience using "MOSS" to detect cheating on programming assignments.” Proceedings of the 29th Frontiers in Education Conference, 1999, Vol. 3, pp. 13-18.

[5] “MOSS - A System for Detecting Software Similarity”, URL *https://theory.stanford.edu/~aiken/moss*

[6] Svajlenko, J., Islam, J. F., Keivanloo, I., Roy, C. K., Mia, M. M, “Towards a big data curated benchmark of inter-project code clones”, Proceedings of In Software Maintenance and Evolution (ICSME), IEEE, 2014, pp. 476-480.

[7] “BigCloneBench - a clone detection benchmark of known clones in the IJaDataset source repository”, URL *https://github.com/clonebench/BigCloneBench*

[8] Roy, C. K., Cordy, J. R. “A survey on software clone detection research”, Queen's School of Computing Technical Report, 2007, vol. 541, no. 115, pp.64-68.

[9] Bellon, S., Koschke, R., Antoniol, G., Krinke, J., Merlo, E. “Comparison and evaluation of clone detection tools”, IEEE Transactions on software engineering, 2007, vol. 33, no. 9, pp. 577-591.

[10] 최성하, 도경구, "Bellon Reference Corpus의 보정", 한국정보과학회 학술발표논문집, pp. 1719-1721, 2015.06

[11] Svajlenko, J., Roy, C. K. (2016, October). “Bigcloneeval: A clone detection tool evaluation framework with bigclonebench.” Proceedings of In Software Maintenance and Evolution (ICSME), IEEE, 2016, pp. 596-600.

[12] Garg, U. “Plagiarism and detection tools: An overview” An International Journal of Engineering Sciences, vol 2, pp. 92-97.

[13] Mozgovoy, M., Fredriksson, K., White, D., Joy, M., Sutinen, E. “Fast plagiarism detection system” Proceedings of In String processing and information retrieval, Springer Berlin/Heidelberg, 2005, pp. 267-270.

[14] Powers, D. M. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation”, Journal of Machine Learning Technologies, 2011, vol. 2, no 1, pp. 37-63

[15] Rattan, Dhavleesh, Rajesh Bhatia, and Maninder Singh. "Software clone detection: A systematic review." Journal of Information and Software Technology, 2013, vol. 55, no. 7, pp.1165-1199.

저 자 소 개

김 규식 (Gyoosik Kim)

2016년 2월 단국대학교 응용컴퓨터공학과 학사 졸업
2017년 현재 단국대학교 컴퓨터학과 석사과정 재학 중

<관심분야>
컴퓨터보안, 소프트웨어 지적재산권 보호, 시스템 소프트웨어, 스마트폰 보안

우 진운 (Jinwoon Woo)

1980년 서울대학교 수학교육과 학사
1989년 미국 University of Minnesota 전산학과 박사
1989년 3월～현재 단국대학교 소프트웨어학과 교수

<관심분야>
소프트웨어 보증, 소프트웨어 감정, 알고리즘 등

조 성제 (Seong-je Cho)

1989년 서울대학교 컴퓨터공학과 공학사
1991년 서울대학교 컴퓨터공학과 공학석사
1996년 서울대학교 컴퓨터공학과 공학박사
2001년 미국 University of California, Irvine 객원 연구원
2009년 미국 University of Cincinnati 객원 연구원
1997년 3월~현재 단국대학교 소프트웨어학과/컴퓨터학과 교수

<관심분야>
컴퓨터보안, 소프트웨어 지적재산권 보호, 시스템 소프트웨어, 스마트폰 보안 등