객체지향 내의 절차식 코드 스타일을 리팩토링하기 위한 가시화 도구 구현

빅지훈*, 김영철**

Implementing a Visualization Tool for refactoring Procedural Style within Object-Oriented Code

Jihoon Park*, and R. Young chul Kim**

요 약

아직 많은 프로그래머들은 객체 지향 프로그래밍을 절차적 코드 스타일의 프로그래밍으로 하고 있다. 이런 코드들은 완벽한 객체 지향적인 코드화를 위한 리팩토링이 필요하다. 하지만 일반적인 개발자는 어느 코드 부분에서 리팩토링을 해야 하는지 알기 어렵다. 이를 식별하기 위해, 마틴 파울러의 Bad Smell 메카니즘 기반 코드 가시화 방법을 제안한다. 이 방법은 개선된 자바 파서와 툴체인으로 자동화된 정적 분석 도구를 구현한다. 이를 통해 코드 유지 보수로 소프트웨어 품질 향상에 도움을 줄 것으로 기대한다.

Key words

Static Analysis, Refactoring, Bad Smell, Java Parser, Code Visualization

Ⅰ. 서 론

객체 지향(object oriented) 프로그래밍은 컴퓨터 프로그래밍 패러다임의 하나이다. 1960년 노위지안 컴퓨팅 센터의 조한 달과 크리스틴이 발표한 시뮬라67을 시작으로 상당한 시간동안 널리 사용되었다[1]. 오늘날에는 객체지향이라는 말을 하지 않고는 개발을 이야기할 수 없을 정도가 되었다. 하지만 많은 프로그래머들은 절차적(procedural) 프로그래밍 스타일로 코딩을 하면서 객체지향 프로그래밍을 하고 있다고 잘못 생각하고 있다. 이런 코드들을 객체지향적(좋은 디자인)인 코드로 바꾸는 기술이 리팩토링이다[2]. 또한 리팩토링은 유지보수 비용을 줄이는 데에도 유용하다. 요구사항을 아무리 정확히 분석하더라도 변경사항은 분명히 생길 것이며 모든 것을 미리 알 수는 없다. 코드를 변경하면서 소프트웨어의 본래 모습과 디자인은 본 모습을 잃어갈 것이다. 이를 해결하기 위해 리팩토링을 사용하면 개발과 수정이 진행되면서 지속적으로 좋은 디자인을 구성할 수 있다. 그렇다면 리팩토링은 어느 순간에 해야 하는지가 문제다.

본 논문에서는 소스 코드를 가시화하여 마틴 파울러의 Bad Smell을 찾아내어 리팩토링이 필요한

* 홍익대학교 소프트웨어공학연구실(pjh@selab.hongik.ac.kr)
** 교신저자: 홍익대학교 소프트웨어공학연구실(bob@hongik.ac.kr)
· 제1저자 (First Author) : 박지훈

부분을 찾아준다. 소스 코드의 가시화란 소스 코드를 해석하여 데이터베이스에 저장한 후 원하는 형식의 그래프로 변환하는 과정을 말한다[3,4]. 2장에서는 관련연구로 Bad Smell과 기존 연구와의 차이점에 대해서 서술한다. 3장에서는 자바 파서를 이용하여 소스 코드를 가시화하는 과정에 대해 서술한다. 마지막으로 4장에서는 결론 및 향후 연구이다.

II. 관련 연구

2.1 Bad Smell

Bad Smell은 1999년 마틴 파울러가 만들어낸 리팩토링이 필요한 코드를 정의한 것이다. 표 1은 Bad Smell 중에서 잠재적 오류발생률이 높다고 판단된 것들을 메서드와 클래스별로 분류한 것이다.

표 1 가시화되는 Bad Smell 목록

범위	항목	설명
Method	Switch Statements	스위치 문이 너무 많다.
	Long Method	메서드 길이가 너무 길다.
	Long Parameter List	파라미터의 양이 너무 많다.
	Feature Envy	다른 클래스 속성을 너무 사용한다.
	Message Chains	메서드끼리의 호출연결고리가 너무 복잡하다.
	Middle Man	클래스가 간단한 위임을 너무 많이 한다.
Class	Data Clump	같은 구조의 파라미터 그룹이 너무 많다.
	Shotgun Surgery	클래스가 하는 일이 적다.
	Comments	주석의 개수가 너무 많다.
	Lazy Class	클래스에 사용하지 않는 기능이 존재한다.
	Data Class	get, set메서드만 존재한다.
	Large Class	인스턴스 변수가 너무 많다.
	Refused Bequest	상속받은 메서드를 모두 사용하지 않는다.

2.2 기존 연구와의 차이점

기존 연구[3]에서는 가시화 툴체인을 이용하여 소스 코드를 가시화했다. 툴체인에는 소스 코드를 파싱하는 도구로 소스내비게이터를 사용하였다. 소스내비게이터는 코드 파싱부터 데이터베이스 생성까지 자동화가 되어있는 오픈소스이다. 하지만 문제는 소스내비게이터에서 분석해주는 데이터베이스만으로는 Bad Smell을 모두 찾아낼 수 없었다는 것이다. 그렇기 때문에 최대한으로 8개의 Bad Smell만 분석할 수 있었다.

기존 연구[4]에서는 이를 보완하기 위하여 가시화 툴체인의 파싱도구인 소스내비게이터를 대신할 수 있는 도구를 연구하였다. 자바 파서는 AST기반으로 자바 코드를 추상화 트리로 분석한다. 이를 사용하면 소스 코드의 모든 부분을 트리 구조로 분석하기 때문에 Bad Smell에 필요한 데이터를 모아 데이터베이스를 구축할 수 있다. 하지만 자바 파서를 통한 소스 코드 분석 후 데이터베이스만 구축하였을 뿐 툴체인에 적용하지 못하여 자동화된 소스 코드의 가시화 및 Bad Smell 추출이 불가능하였다.

본 논문에서는 기존연구[3.4]를 통합하여 자바 파서 기만의 코드 정적 분석기를 구현하였다. 또한 코드 가시화를 통해 표 1의 회색 부분의 Bad Smell을 추가로 찾아내었다.

III. 자바 파서를 이용한 Bad Smell 가시화

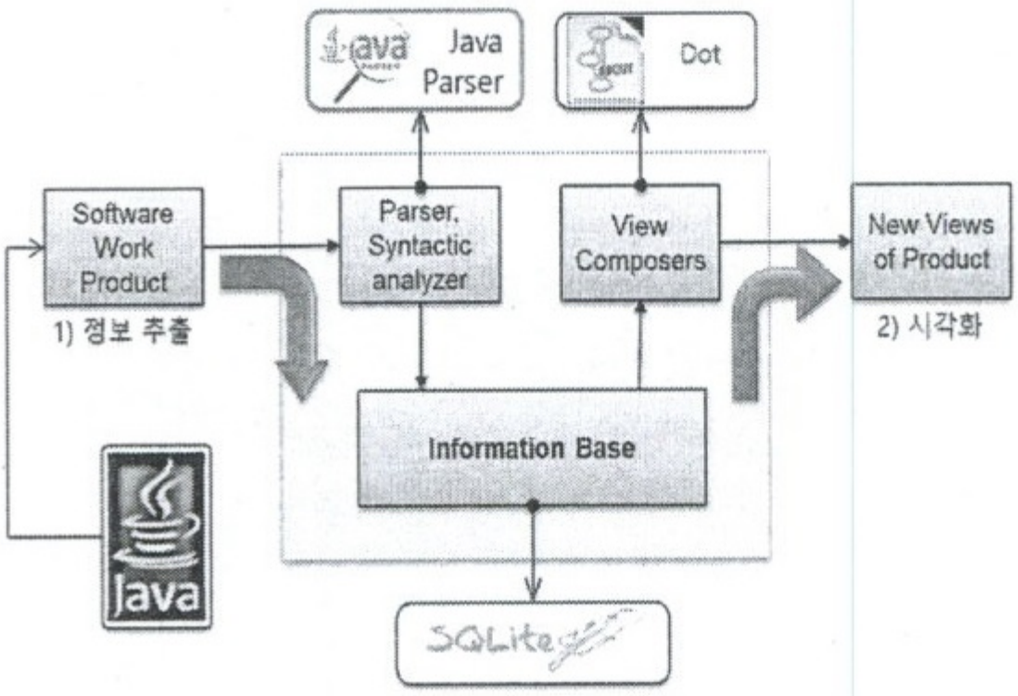


그림 1 자바 파서를 이용한 자동화된 코드 가시화 도구

그림 1은 자동화된 코드 가시화 도구의 구성도이다. 기존의 파싱 도구였던 소스내비게이터를 자바 파서로 교체하였다. 그로 인해 소스 내비게이터가

기본적으로 제공했던 데이터베이스가 아닌 Bad Smell추출에 적합한 데이터베이스를 재구성 하였다 [4]. 그로 인해 추출되었던 기존의 Bad Smell 항목에서 Switch Statements, Long Method, Data Clump, Shotgun Surgery, Comments가 추가되었다.

```
ResultSet ss = statement.excuteQuery(
        "select SWITCH_COUNT from JPDB_METHOD
                        where NAME = [name]");
while (ss.next()) {
  count ++;
}
return count;
```

그림 2 Switch Statements 추출 쿼리 프로그램

그림 2는 Switch Statements를 추출하는 쿼리문이다. Switch문의 개수가 사용자가 정의해놓은 일정 개수를 넘어가면 Switch Statemets로 Bad Smell을 체크하여 리팩토링이 필요함을 보여준다.

```
ResultSet lm = statement.excuteQuery(
    "select START_LINE, END_LINE from JPDB_METHOD
                        where NAME = [name]");
while (ss.next()) {
  int max = max(ss.getString("END_LINE");
  int min = min(ss.getString("START_LINE");
  int line = max - min;
  if (line > count) {
    return count;
  }
}
```

그림 3 Long Method 추출 쿼리 프로그램

그림 3은 Long Method를 추출하는 쿼리문이다. 메서드의 시작 라인 숫자와 마지막 라인 숫자를 계산하여 사용자가 정의한 일정 숫자를 넘어가면 Long Method로 Bad Smell을 체크한다.

```
ResultSet dcp = statement.excuteQuery(
        "select PARAMETER_TYPE, PAREMETER_NAME from
                    JPDB_METHOD where NAME = [name]");
while (dcp.next()) {
  dataClump.put(
 dcp.getString("PARAMETER_TYPE + PARAMETER_NAME"));
  if (dataClump.count()> count) {
    return count;
  }
}
```

그림 4 Data Clump 추출 쿼리 프로그램

그림 4는 Data Clump를 추출하는 쿼리문이다. 메서드의 파라미터들이 일정한 패턴을 가지고 자주 나타난다면 해당된다. 이는 Long Parameter에도 해당될 수 있다. 예를 들어, 간단한 자료형의 파라미터들을 여러 메서드로 주고받는 횟수가 많다면 객체를 생성하여 파라미터로 보내주어야 한다. 하지만 이는 Data Class에 해당될 수도 있기 때문에 신중한 리팩토링이 필요하다.

```
ResultSet sgs = statement.excuteQuery(
                        "select count(REFER_CLASS) as refer,
    count(REFERRED_CLASS) as referred from JPDB_REFER
                        where REFERRED_NAME = [name]");
while (sgs.next()) {
  int refer = sgs.getString("refer");
  int referred = sgs.getString("referred");
  int shotgunSurgery = refer – referred;
  if (shoygunSurgery > count) {
    return count;
  }
}
```

그림 5 ShotgunSurgery 추출 쿼리 프로그램

그림 5는 ShoutgunSurgery를 추출하는 쿼리문이다. 인스턴스 변수나 메서드가 속해있는 클래스에서보다 다른 클래스에서 더 많이 호출되고 사용되는 경우에 해당된다. 이 경우에는 더 많이 호출하는 클래스로 필드를 이동하여야 한다.

```
ResultSet cmt = statement.excuteQuery(
 "select COMMENT from JPDB_CLASS where NAME = [name]");
while (cmt.next()) {
  int comment = cmt.getString("COMMENT");
  if (comment > count) {
    return count;
  }
}
```

그림 6 Comments 추출 쿼리 프로그램

그림 6은 Comments를 추출하는 쿼리문이다. 클래스안에 주석의 라인 수가 사용자가 정의한 개수보다 많다면 해당된다. 코드에 대한 설명이 필요하여 주석을 많이 작성한 경우 주석이 없어도 이해가 될 수 있게 변수나 메서드의 이름을 잘 작성해야한다.

그림 7은 오픈 소스인 JSAP-2.02를 예제로 가시화한 결과이다. 클래스 다이어그램 형태로 가시화하였으며 각 클래스나 메서드에 Bad Smell이 확인되었을 경우 빨간색으로 색을 주었다. 각 Bad Smell은 클래스명이나 메서드명 뒤에 약자로 첨부하였다

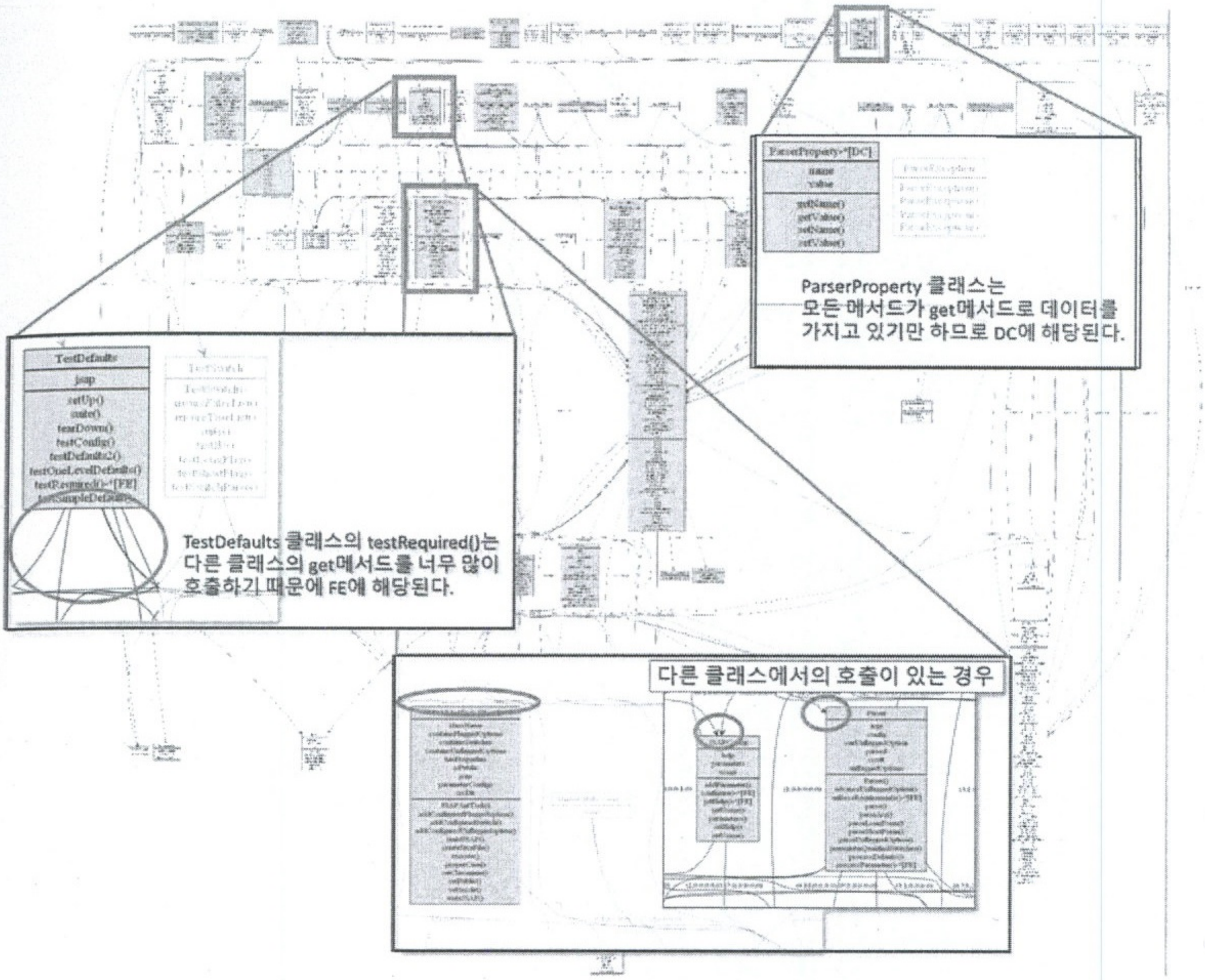


그림 7 오픈 소스인 JSAP-2.02를 가시화하여 Bad Smell을 추출한 결과

[5].

V. 결 론

많은 개발자들이 객체지향적인 코드를 구현하려 하지만 단지 코드를 클래스로 감싸는 것만으로 객체지향이라고 착각하고 있다. 그렇게 작성된 대규모의 소프트웨어는 유지보수에도 문제가 생기기 쉬우며 본래 디자인과 점점 달라질 것이다. 본 논문에서 구현한 자동화된 코드 가시화 프로그램을 이용하여 코드 내의 리팩토링이 필요한 부분을 어렵지 않게 찾아낼 수 있다. 그로인해 코드를 복잡하게 하고 비용을 낭비하는 일을 줄일 수 있을 것으로 기대한다.

향후 연구로는 모든 Bad Smell을 가시화할 것이며 지금은 한 번에 모든 것을 표시하여 가시성이 떨어지므로 각각의 Bad Smell을 표시할 수 있도록 개선할 것이다.

ACKNOWLEDGMENT

이 논문은 2017년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2017R1D1A3B03035421)

참 고 문 헌

[1] Object-Oriented Programming, https://en.wikipedia.org/wiki/Object-oriented_pro

gramming.
[2] Martin Fowler, “Refactoring: Improving The Design of Existing Code “, Addison-Wesley, 2002.
[3] 박지훈, 손현승, 박보셩, 이진협, 김영철, “효율적인 리팩토링을 위한 조직 특성의 품질 지표 기반 우선순위 선정 자동화”, 소프트웨어공학회, 제19권, 제1호, pp175-178, 2017.
[4] 박지훈, 박보경, 이근상, 김영철, “기존 자바 파서 확장 기반의 코드 정적 분석기 구현”, 한국정보처리학회, 제24권, 제1호, pp641-644, 2017.
[5] 박지훈, 장우성, 이진협, 손현승, 김영철, “확장된 나쁜 코딩 습관 식별 구현”, 한국정보과학회, pp401-403, 2017.

PTL Volume 5-3(ISSN 2288-8195)

2017 (사)ICT플랫폼학회

추계학술대회 및 지능정보사회 ICT플랫폼기술 워크숍 논문집

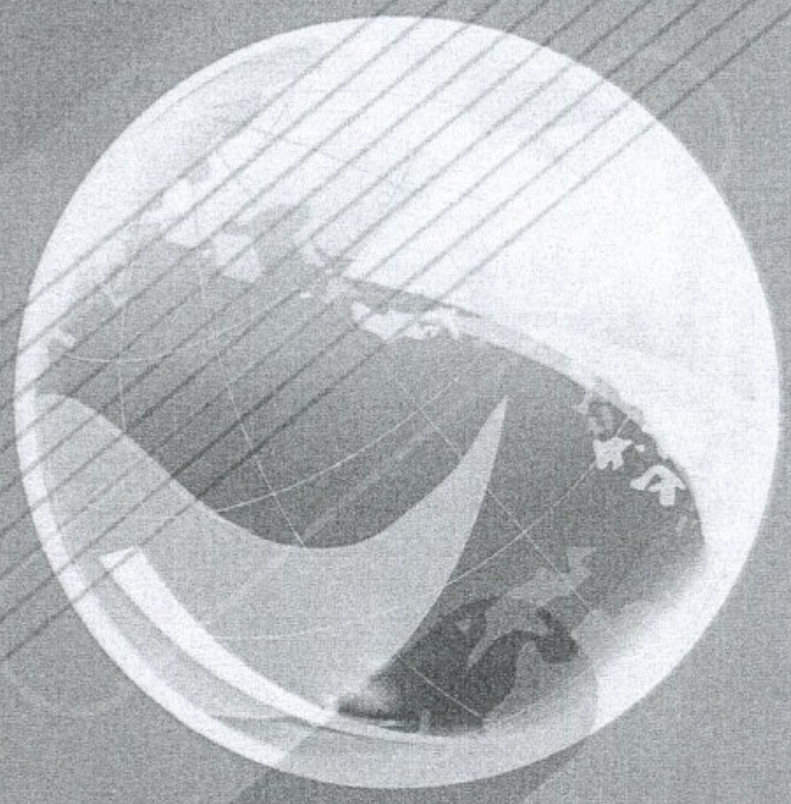
일 시 | 2017년 12월 8일(금) 오후 1시 ~ 6시

장 소 | 동국대학교

주 최 : 동국대학교 LINC+ 사업단, (사)ICT플랫폼학회

주 관 : (사)ICT플랫폼학회

후 원 : ㈜휴네시온, ㈜엠엘소프트, ㈜소프트보올,
㈜맥스테드, ㈜프로아트프로덕션, ㈜세림티에스지
㈜대신정보통신, ㈜위니텍



ICTPS
ICT 플랫폼학회

목 차