

ISSN 2466-0825

2016

동계학술발표회 논문집

2016년 12월 21일~23일
강원도 휘닉스 파크

kiise

<http://www.kiise.or.kr>



한국정보과학회
KOREAN INSTITUTE OF INFORMATION SCIENTISTS AND ENGINEERS

02.2-06 【우수논문】 모바일 환경에서의 동적 장면의 효율적인 이차 광선 추적을 위한 격자 가속 구조 서웅(서강대), 최병준(LG전자), 임인성(서강대)

02.3 소프트웨어공학 II

12.21(수) 13:00~15:00, 토파즈 I

▷좌장 : 김영철(홍익대)

▷평가위원 :

<초청논문>

1. Automated Model-Based Android GUI Testing using Multi-level GUI Comparison Criteria, 백영민(KAIST)

- 02.3-01 시스템 오브 시스템즈 모델링을 위한 다중 에이전트 시스템 메타모델링 기법 분석
정유림 · 백영민(KAIST), 윤일철(한국뉴욕주립대), 배두환(KAIST)
- 02.3-02 확장된 나쁜 코딩 습관 식별 구현
박지훈 · 장우성 · 이진협 · 손현승 · 김영철(홍익대)
- 02.3-03 가치 있는 모듈 식별을 위한 오픈 소스 기반 소프트웨어 현대화 시스템 구축
변은영 · 박보경 · 장우성 · 김영철(홍익대)
- 02.3-04 【우수논문】 모터 속도 제어를 이용한 드론 비행 안정화 기법
윤보람 · 김덕엽 · 이성희 · 이우진(경북대)
- 02.3-05 CBD 개발 방법론 기반의 촬영계획 시스템 최적화 방안 연구
장윤정 · 채태병(한국항공우주연구원)

02.4 고성능컴퓨팅

12.21(수) 13:00~15:00, 토파즈 II

▷좌장 : 엄현상(서울대)

▷평가위원 : 이재환(항공대)

<초청논문>

1. Interference Management for Distributed Parallel Applications in Consolidated Clusters, 이강욱(UNIST)

- 02.4-01 오픈소스 파일시스템을 활용한 버스트 버퍼 프로토타입 구축 및 성능 분석 연구
변은규(KISTI), 구동훈 · 이재환(항공대)
- 02.4-02 다중 가상 머신 환경에서 데이터 접근 빈도를 이용한 계층 RAID의 동적 패리티 생성 기법
송태건 · 이병현 · 안청진 · 김덕환(인하대)
- 02.4-03 몬테카를로 시뮬레이션 문제의 고속화 설계
정예찬 · 김동승(고려대)
- 02.4-04 【우수논문】 차세대 매니코어 프로세서 기반 다중 MPI 응용 실행 최적화 선행 연구
김서영 · 남덕윤 · 박근철 · 김직수(KISTI)

02.5 고신뢰컴퓨팅

12.21(수) 13:00~15:00, 토파즈 III

▷좌장 : 장성태(수원대)

▷평가위원 : 허준영(한성대)

- 02.5-01 임베디드 리눅스 기반의 BLE통신 기기에서 악성행위 탐지 시스템 개발
이명건 · 정윤식 · 조성제(단국대)
- 02.5-02 프로파일 기반 재사용 거리를 이용한 GPU 캐시 바이패싱 기법
홍성인 · 김현준 · 한환수(성균관대)
- 02.5-03 실시간 응용에서 클라우드의 스케줄링 지연시간을 고려한 오프로딩 기법
민홍(호서대), 정진만(한남대), 김봉재(선문대), 허준영(한성대)

확장된 나쁜 코딩 습관 식별 구현

박지훈⁰¹, 장우성², 이진협³, 손현승⁴, 김영철⁵
 홍익대학교 컴퓨터정보통신공학과 소프트웨어공학연구실
 {⁰¹pjh, ²jang, ³ljh, ⁴son, ⁵bob}@selab.hongik.ac.kr

Implementing the automatic identification of the Bad Smell Coding Patterns

Ji-hoon Park, Woo-sung Jang, Jin-hyub Lee, H. S. Son, R. Young-chul Kim
 SE Lab, Hongik University

요 약

소프트웨어 시장이 점차 확대되고 있지만, 벤처/중소기업들은 자체 개발자의 나쁜 코딩 습관에 대한 고민 없다. 이로 인해 개발된 소프트웨어에 대한 고품질을 위해 많은 노력을 해야 한다. 본 논문은 역공학 기반 양질의 코드 생산하기 위한 나쁜 코딩 패턴 식별 구현 방법을 제안한다. 이를 위해 코드 가시화 도구 구현을 통해, 나쁜 냄새 패턴 템플릿 정의 및 정보 추출 SQL 프로그램을 개발하여 수행한다. 이를 가시화 프로세스에 적용하여, 스스로 나쁜 코딩 습관 개선 동기를 제공한다. 이는 리팩토링 후, 소프트웨어 품질 향상을 가능하다. 또한 기존의 레거시에도 적용하면 나쁜 냄새 코딩 식별 기반 리팩토링이 가능하리라 본다.

1. 서 론

SW의 중요성이 점차 강조되면서 개발자들은 SW 품질관리에 중점을 두고 있다. SW의 품질을 향상시키려면 소프트웨어 공학적 접근의 개발을 해야 한다. 하지만 국내의 중소기업들은 인력, 비용 등의 여러 측면에서 어려움이 존재한다. 또한, 구현 중심으로 저 품질의 소프트웨어를 양산하고 있다[1,2]. 내부 구조화가 진행되지 않으므로 비가시성을 가진 소프트웨어의 품질 관리 및 유지보수에는 어려움이 따른다. 그렇기 때문에 소프트웨어 품질관리를 위해서는 소프트웨어 가시화가 필요하다.

본 논문에서는 기존연구의 역 공학을 통해 소스 코드를 가시화한다. 가시화를 통해 소스 코드의 결합도 및 코드의 Bad Smell을 추출하는 방법을 소개한다[3,4,5,6]. Source Navigator라는 오픈소스를 이용하여 소스 코드의 정보를 추출한다. 추출된 정보들을 분석하여 소스 코드의 결합도와 Bad Smell을 가시화한다. 가시화된 그래프를 통해서 소스 코드의 개선해야 할 부분을 찾아낼 수 있다.

본 논문은 기존 연구[3,4]를 확장한 논문으로 Bad Smell가시화에 대한 기준을 2가지 더 추가하였다. 구성은 총 4장으로 이루어져있다. 2장에서는 관련연구로 Bad Smell의 개념을 설명한다. 3장에서는 가시화 프로세스에 대해서 설명한다. 마지막으로 4장에서는 결론 및 향후연구에 대해서 설명한다.

2. 관련연구

Bad Smell(나쁜 냄새)은 Martin Fowler의 'Refactoring'이라는 책에서 처음으로 사용된 비유이다. 소스 코드를 작성하면서 쓰이는 나쁜 코딩 습관들을 정리하였다. 겉으로 보이는 소프트웨어의 기능에는 문제가 없다. 하지만 내부적으로는 불필요한 구조로 인하여 복잡한 디자인으로 이루어질 수 있다. 이런 경우 소프트웨어의 의도된 디자인은 시간이 갈수록 망가질 가능성이 있다. 이러한 불필요한 구조의 나쁜 코딩 습관은 리팩토링하여 더 좋은 코드로 수정이 가능하다. Martin Fowler는 책에서 총

22가지의 Bad Smell을 정의하였다. 본 논문에서는 총 22가지의 Bad Smell중 7가지의 Bad Smell에 대해서 구현하였다[7].

3. 가시화 프로세스

3.1 Source Navigator를 통한 정보 추출

본 논문에서는 기존 연구[3,4]의 결합도 추출 외에 코드의 Bad Smell요소를 추가적으로 검출하였다. 프로세스의 구성은 다음과 같다. Source Navigator로 JAVA 소스 코드를 분석하여 SNDB파일을 생성한다. 생성된 SNDB파일들의 정보를 이용하여 각 Bad Smell에 부합하는 부분을 찾아낸다. 표 1은 추출한 SNDB파일의 데이터베이스 테이블을 나타낸다.

표 1 SNDB 파일의 정보

테이블명	정보
SNDB_CL	클래스의 이름과 속성
SNDB_IV	클래스 인스턴스변수의 정보
SNDB_LV	메서드 로컬변수의 정보
SNDB_MD	메서드의 이름과 속성
SNDB_BY	각 모듈끼리의 호출정보
SNDB_IN	클래스의 상속관계
SNDB_COUPLING	클래스의 결합도 정보

3.2 Bad Smell에 해당되는 코드형식

다음은 각각의 Bad Smell에 해당되는 코드 설명이다[7]. Large Class는 클래스 내의 인스턴스 변수가 너무 많은 경우이다. 지나치게 많은 인스턴스 변수들을 가지고 있다면 비슷한 변수들을 묶어서 클래스 추출을 할 수 있다. Long Parameter List는 메서드 파라미터 값의 개수가 많은 경우이다. 파라미터의 개수가 많다면 객체를 이용하면 된다. Feature Envy는 자신의 클래스보다 다른 클래스에 관심을 가지는 메서드가 원인이다. Lazy Class는 클래스 내의 메서드 모두가 사용되지 않으면

해당된다. Data Class는 클래스 전체가 getter나 setter메서드로 이루어진 데이터 클래스에 해당된다. Message Chains는 어떠한 정보를 얻으려 할 때 여러 객체를 거쳐 갈 경우에 해당된다. Refused Bequest는 서브클래스가 슈퍼클래스의 메서드를 거부할 때 해당된다.

3.3 Bad Smell을 추출하는 SQL Query문

실제 소스코드에서 예시코드와 같은 형식의 Bad Smell을 찾아내기 위해 SNDB파일의 데이터를 이용한다. 다음은 SNDB의 각 테이블의 정보를 검색 및 접근하여 판단하기 위한 SQL문에 대해 설명하였다.

- Large Class (LgC)

표 2에서 Large Class는 각 클래스들 내에 선언된 인스턴스 변수들의 개수를 조회한 후 판단한다. SNDB_CL 테이블에서 클래스들의 이름을 조회한다. 조회된 클래스들의 이름을 이용하여 SNDB_IV에서 각 클래스들의 인스턴스 변수 개수를 조회한다. 조회된 결과는 lgc에 저장된다.

표 2 Large Class 추출 SQL 프로그램

```
Resultset lgc = statement.executeQuery( "
    select count(*) as CNT from SNDB_IV where
    CLASS_NAME in (select [name] from SNDB_CL) );
int count_lgc = Integer.parseInt(lgc.getString( "CNT" ));
```

count_lgc의 개수가 10개 이상이라면 Large Class로 판단한다. 클래스 이름 뒤에 [LgC]를 표시한다.

- Long Parameter List (LPL)

표 3에서 Long Parameter List는 각 메서드의 파라미터 개수를 조회한 후 판단한다. SNDB_MD 테이블에서 ARGUMENT_TYPE을 조회한다. 조회된 결과를 split메서드로 분할하여 파라미터의 개수를 체크한다. 조회된 결과는 count_lpl에 저장된다.

표 3 Long Parameter List 추출 SQL 프로그램

```
Resultset lpl = statement.executeQuery( "
    select ARGUMENT_TYPE from SNDB_MD" )
int count_lpl = lpl.getString( "ARGUMENT_TYPE" )
    .replaceAll("[ ]").split(",").trim().length()
```

lpl에 저장된 결과의 개수를 측정한다. count_lpl이 6개를 넘어가면 메서드의 이름 뒤에 [LPL]을 표시한다.

- Feature Envoy (FE)

표 4에서 Feature Envoy는 호출하는 get메서드의 개수를 조회한 후 판단한다. SNDB_BY 테이블에서 get으로 시작하는 REFERRED_SYMBOL_NAME의 개수를 조회한다.

표 4 Feature Envoy 추출 SQL 프로그램

```
Resultset fe = statement.executeQuery( "
    select count(*) as CNT from SNDB_BY where
    REFERRED_SYMBOL_NAME LIKE 'get%' and
    REFERRED_CLASS_NAME in
    (select [name] from SNDB_CL) );
int count_fe = Integer.parseInt(fe.getString( "CNT" ));
```

fe에 저장된 결과의 개수를 측정한다. 개수가 4개를 넘어가면 메서드의 이름 뒤에 [FE]를 표시한다.

- Lazy Class (LzC)

표 5에서 Lazy Class는 클래스 안에 선언된 메서드가 모두 사용되고

있는지를 판단한다. SNDB_BY 테이블에서 클래스 별로 중복 값을 제외하여 REFERRED_SYMBOL_NAME을 조회한다. SNDB_MD 테이블에서 METHOD_NAME을 조회한다.

표 5 Lazy Class 추출 SQL 프로그램

```
Resultset lzc1 = statement.executeQuery( "
    select distinct REFERRED_SYMBOL_NAME
    from SNDB_BY where CLASS_NAME in
    (select [name] from SNDB_CL) );
ArrayList arrLzc1 = add(lzc1.getString( "REFERRED_SYMBOL_NAME" ));
Resultset lzc2 = statement.executeQuery( "
    select METHOD_NAME from SNDB_MD where
    LASS_NAME in (select [name] from SNDB_CL) );
ArrayList arrLzc2 = add(lzc2.getString( "METHOD_NAME" ));
```

arrLzc1과 arrLzc2의 결과 값이 같지 않다면 모든 메서드가 호출되지 않는 것으로 판단한다. 이 경우 클래스의 이름 뒤에 [LzC]를 표시한다.

- Message Chains (MC)

표 6에서 Message Chains는 어떤 객체를 얻으려 할 때 다른 여러 객체를 거쳐야 하는 경우인지 판단한다. SNDB_BY 테이블에서 각 클래스에 해당하는 REFERRED_SYMBOL_NAME을 중복 값을 제거하여 조회한다. SNDB_MD 테이블에서는 각 클래스에 해당하는 METHOD_NAME을 조회한다.

표 6 Message Chains 추출 SQL 프로그램

```
Resultset mc = statement.executeQuery( "
    select REFERRED_SYMBOL_NAME from SNDB_BY
    where REFERRED_SYMBOL_NAME LIKE 'get%'
    CLASS_NAME in (select [name] from SNDB_CL) );
int mcCount += 1;
```

mc를 재귀함수로 선언하여 get으로 시작하는 메서드를 검색한다. 재귀할 때마다 get으로 시작하는 메서드를 찾아내면 mcCount를 1씩 증가시킨다. mcCount를 측정하여 4이상이면 메서드의 이름 뒤에 [MC]를 표시한다.

- Data Class (DC)

표 7에서 Data Class는 클래스가 데이터로만 구성되어있는지 판단한다. SNDB_MD 테이블에서 get이나 set으로 시작하는 METHOD_NAME의 개수를 판단한다.

표 7 Data Class 추출 SQL 프로그램

```
Resultset dc1 = statement.executeQuery( "
    select REFERRED_SYMBOL_NAME from SNDB_BY
    where CLASS_NAME in
    (select [name] from SNDB_CL) );
Resultset dc2 = statement.executeQuery( "
    select REFERRED_SYMBOL_NAME from SNDB_BY
    where (REFERRED_SYMBOL_NAME LIKE 'get%' or
    REFERRED_SYMBOL_NAME LIKE 'set%') and
    CLASS_NAME in (select [name] from SNDB_CL) );
ArrayList arrDc = add(dc2.getString( "REFERRED_SYMBOL_NAME" ));
```

dc1과 dc2의 결과 값을 비교한다. 값이 같으면 클래스의 이름 뒤에 [DC]를 표시한다.

- Refused Bequest (RB)

표 8에서 Refused Bequest는 서브클래스가 슈퍼클래스로부터 상속받은 변수와 메서드를 모두 사용하는지 판단한다. SNDB_IV 테이블에서 상속관계를 조회한다. 조회된 결과를 이용하여 오름차순으로 슈퍼클래스의 메서드와 호출정보를 정렬한다.

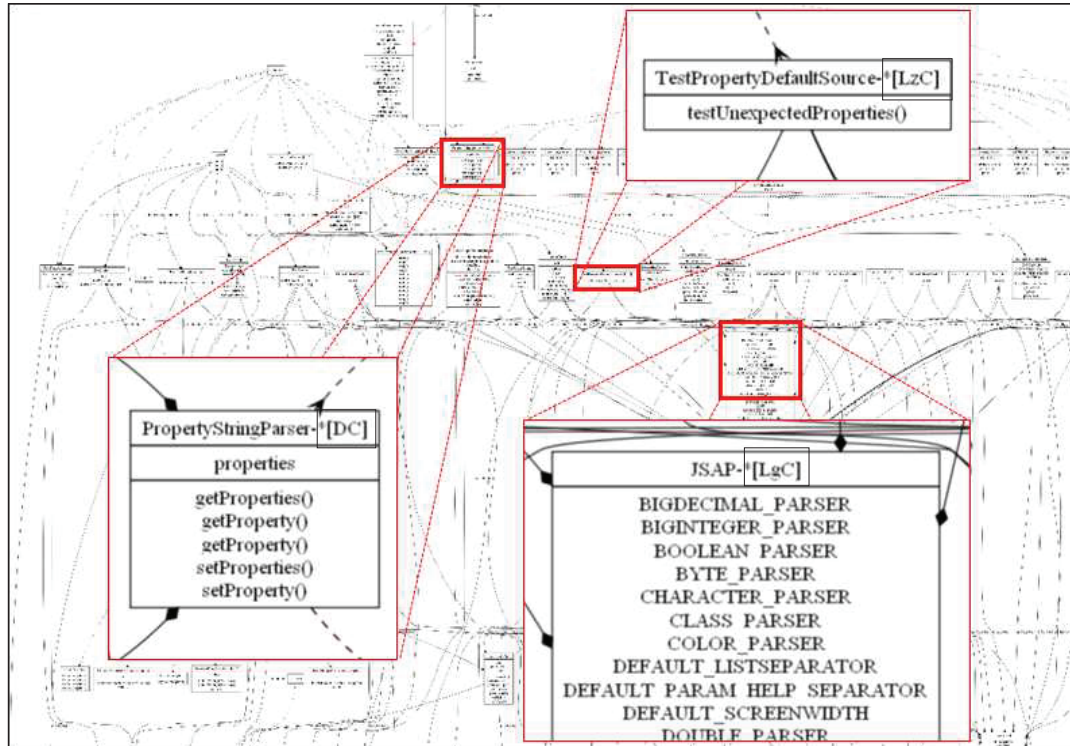


그림 1 오픈소스 JAVA Parser의 Bad Smell을 가시화한 그래프

표 8 Refused Bequest 추출 SQL 프로그램

```

Resultset rb = statement.excutequery("
    select * from SNDB_IV") {
String subname = rb.getString("SUB_CLASS_NAME");
String supname = rb.getString("SUPER_CLASS_NAME");
Resultset rb1 = statement.excutequery( "
    select distinct REFERRED_SYMBOL_NAME from SNDB_BY
    where REFER_CLASS_NAME='+name+' and
    REFERRED_CLASS_NAME=REFER_CLASS_NAME
    order by asc" );
Resultset rb2 = statement.excutequery( "
    select METHOD_NAME from SNDB_MD
    where CLASS_NAME='+name+' order by asc" );
}

```

rb1의 결과 값과 rb2의 결과 값을 비교한다. rb1에 rb2의 값들이 모두 존재하지 않으면 상속받은 메서드를 모두 사용하지 않는 것으로 판단한다. 두 클래스의 상속연결선을 빨간색으로 표시한다.

3-4. Graphviz를 통한 소스 코드 가시화

추출하고 정제한 데이터를 이용하여 Graphviz에 입력한다. 그림 1은 오픈 소스인 JAVA Parser의 결함도와 Bad Smell을 가시화한 그래프이다. 네모 친 부분들은 소스 코드에서 나쁜 냄새가 발견된 부분들이다.

PropertyStringParser클래스는 get-, set-메서드만 가지고 있기 때문에 Data Class에 해당하여 [DC]가 표시됐다. TestPropertyDefaultSource클래스는 두 개의 호출선을 보아 어떠한 기능을 수행하지만 자신을 호출하는 선이 없다. 그러므로 사용되지 않는 클래스에 판단한다. Lazy Class에 해당하므로 [LzC]가 표시됐다. JSAP클래스는 인스턴스변수를 10개 이상 가지고 있다. Large Class에 해당하므로 [LgC]가 표시됐다.

4. 결론 및 향후 연구

현재 소프트웨어를 구현중심의 개발방법만 지향한다면 큰 문제가 발생할 가능성이 높다. 본 논문에서 제안한 가시화 프로세스를 적용시켜, 비가시성의 소프트웨어를 한층 쉽게 파악하

게 했다. 기존 완성된 소프트웨어도 역 공학을 통해 구조화를 진행할 수 있다. 개발 중인 소프트웨어는 개발과 동시에 가시화를 하여 좋은 품질의 소프트웨어를 생산할 수 있다. 그로 인해 막대한 유지보수비용을 절감할 수 있다.

현재는 기존의 결함도를 추출하는 논문을 확장하여 Bad Smell의 패턴을 7가지 더 추가하였다. 차후 다양한 오픈 소스들을 이용하여 나머지 항목들도 패턴을 추출할 계획이다.

ACKNOWLEDGEMENT

이 논문은 2015년 교육부와 한국연구재단의 지역혁신창의인력양성사업의 지원을 받아 수행된 연구임(NRF-2015H1C1A1035548)

참 고 문 헌

- [1.] NIPA, "SW 개발 품질관리 매뉴얼(SW Visualization)", SW 공학센터, pp. 3, 2013.
- [2.] 조현훈, 최용락, 류성열, "McCabe 및 BP Win도구를 이용한 소프트웨어 역공학 사례연구", 정보과학회논문지 2000 제 6권 제5호, pp.528-535
- [3.] 박지훈, 권하은, 강건희, 이근상, 김영철, "코드 가시화를 통한 나쁜 코딩 습관 개선 방안 연구", 한국인터넷방송통신학회 2015 종합학술대회 논문집 제13권 제1호 pp.47-48
- [4.] 박지훈, 김영철 "나쁜 코딩 습관 개선을 위한 코드 가시화 연구", 한국정보처리학회 2016 추계학술발표대회 논문집 제23권 제2호 pp.477-480
- [5.] 강건희, 손현승, 김영수, 박용범, 김영철 "SW 가시화 기반 리팩토링 기법 적용을 통한 정적 코드 복잡도 개선", 한국정보처리학회, 제 21권, 제2호, pp.646-649, 2014
- [6.] 권하은, 박보경, 이근상, 박용범, 김영수, 김영철, "코드 가시화부터 모델링 추출을 통한 역공학 적용", 한국정보처리학회, 제21권, 제2호, pp.650-653, 2014
- [7.] Martin Fowler, "Refactoring", Addison-Wesley, 2002