역공학 분석 시스템 구현

박형철 · 간정현 · 장태진 · 이주연 · 권순각 · 이중화

동의대학교 컴퓨터소프트웨어공학과

Implementation of Reverse Engineering Analysis

Hyeongcheol Park · Jeonhyeon Gan · Taejin Jang · Juyeon Lee ·
Soonkak Kwon · Junghwa Lee

Department of Computer Software Eng., Dongeui University

요 약

역공학(Reverse Engineering)은 이미 만들어진 소프트웨어 시스템을 역으로 추적하여 처음의 문서나 설계기법 등의 자료를 얻어 내는 것을 의미한다. 본 논문은 역공학 개념을 이용하여 하나의 파일에서 뽑을 수 있는 데이터 즉, 클래스나 변수, 메소드, 연관 관계 등을 추출하여 xml문서에 저장 후 역공학한 프로젝트 내에서 선언된 변수와 함수들의 사용횟수를 도출하는 시스템을 구현한다. 이를 바탕으로 역공학한 프로젝트 내에서 선언된 변수와 함수들의 사용횟수를 도출함으로써 검출 및 유지보수가 용이하며, 클래스 다이어그램이 좀 더 자세하게 그려줄 수 있다.

Ⅰ. 서 론

Reverse Engineering이란, 소프트웨어 공학의 한 분야로 이미 만들어진 시스템을 역으로 추적하여 처음의 문서나 설계기법 등의 자료를 얻어 내는 일을 말한다. 이것은 시스템을 이해하여 적절히 변경하는 소프트웨어 유지보수 과정의 일부이다.

현재 만들어지는 거의 대부분의 소프트웨어들은 프로그래머들도 그렇고 제품을 주문한 오더도 그렇고 내부적인 구조에 대해 전혀 신경을 쓰지 않고 프로젝트 결과물에 대해서만 중시하고 있는 상황이다. 그러다보니 팀원 내 할당받은 개개인의 방식으로 만든 후 끼워 맞추기 식으로 진행되어 사용되지도 않는 변수나 함수들이 나와 쓸모없는 자원을 낭비하는 경우가 많고, 한 클래스 내에서 모든 것을 선언한다거나 제대로 클래스 연관 관계를 표현하지 못하는 경우가 많다.

따라서, Reverse Engineering 개념을 이용해 하나의 파일에서 뽑을 수 있는 정보 즉, 클래스나 변수, 메소드, 연관 관계 등을 추출하여 xml문서에 저장 후 Reverse할 대상 프로젝트에서 선언된 변수와 함수들의 사용 횟수를 도출하여 검출 및 유지보수에 용이하도록 하는 것이며, 기존 툴에서 그려주는 클래스 다이어그램을 좀 더 자세히 그려줄 수 있도록 하는 것이다.

본 논문은 Reverse할 언어의 대상을 현재 자바만을 보고 진행하였고 차후 다른 언어를 준비 예정이다.

Ⅱ. 시스템 흐름

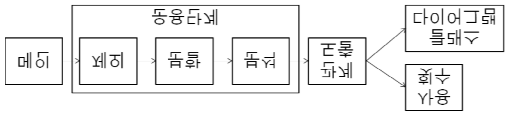

그림 1. 시스템 흐름도

네모를 하나의 클래스로 보면 메인 클래스에서 프로그램을 시작하고 운용 클래스는 제외, 분할, 분석클래스로 나누어 관리한다.

제외 클래스에서는 주석을 제거하고 스트링 변수 값을 제거한다. 분할 클래스에서는 한 클래스 파일 안에 여러 개의 클래스가 선언 되어 있을 경우 따로 파일을 만들어 저장을 한다.

분석 클래스에서는 크게 클래스 타입(클래스,

인터페이스, 이넘), 변수 타입(자료형 변수, 인스턴스 변수), 함수 타입(일반 메소드, 추상 메소드)로 생각하고 정보들을 저장한다.

그리고 저장된 정보를 상속 받은 것인지 인터페이스를 통해 구현 것인지에 대한 관계에 대해 도출하여 저장한다.

이후 사용 횟수클래스에서 선언된 변수들과 함수들의 사용 횟수들을 도출하여 보고서를 작성해주고, 클래스 다이어그램을 그려 클래스 간의 연관 관계를 보기 쉽게 해준다.

Ⅲ. 클래스 분석 방법

3-1 제거

맨 처음 제거를 하는 이유는 예약어 판별에 방해가 될 수 있기 때문이다. 예를 들어 String str = “class”; 라고 선언하였을 경우 class라는 단어를 보고 클래스를 분할하기 때문에 위험 요소가 되어 문장을 분석할 때 방해 요소를 사전에 제거할 필요가 있다.

그리고 프로젝트를 분석할 때 한줄 주석(//~~~\n)과 다중 주석(/* ~~~ */)에 대해서는 분석할 필요가 없기 때문에 여기서 사전에 제거한다.

3-2 클래스 분할

클래스 분할은 하나의 자바 파일 안에는 내부 클래스라던가 한 클래스 안에 여러 클래스가 선언될 수 있다. 이렇게 되어 있는 부분을 각각의 독립된 클래스로 분할하는 역할을 한다. 분할된 각각의 클래스들은 임시 파일로 저장되며 이 임시 파일들이 클래스 분석 때 쓰인다.

실제 구현에서 분할 클래스는 현재 라인이 클래스 선언 부분인지 함수 선언 부분인지 등을 상위 클래스인 운영 클래스에게 리턴 값을 넘겨주는 방식으로 알려주기만 한다. 즉, 파일 생성 및 저장 등은 운영 클래스가 담당한다.

3-3 구문 분석

구문 분석을 하기 위해 파일에서 라인을 받아온 다음 문자들을 읽으면서 특수문자(공백, 소괄호, 중괄호, 대괄호, 탭, 꺽새, 등호 등)를 통해 단어 단위로 분리시키고 예약어와 키워드를 정의해 둔 해쉬맵에 단어들을 넣어 분석한다. 분석한 단어들에 해당하는 값들을 저장 후 라인에 대한 속성, 즉 클래스, 인터페이스, 이넘, 메소드, 추상 메소드, 자료형 변수, 인스턴스 변수, 연관 관계인지를 특수문자(소괄호, 중괄호, 세미콜론)의 존재 유무를 통해 파악함으로써 현재 라인에 대한 구문 분석을 완료한다.

3-4 xml문서 저장

구문분석을 통해서 나온 값들을 순차적으로 xml 구조로 저장하는데, 노드의 이름과 구문분석을 통해 저장된 값들은 조건문을 거쳐서 구성하고, 4개의 깊이로 구성한다.

깊이 1은 패키지이다.

깊이 2는 패키지 이름, 클래스 타입(클래스, 인터페이스, 이넘)이다.

깊이 3은 클래스 타입 정보(extends, implement, visibility, static, final 등), 메소드, 변수이다.

깊이 4는 메소드 정보(visibility, return type, static, final 등), 변수 정보visibility, return type, static, final 등)이다.

다음과 같은 깊이 구조로 xml 문서에 저장 한다.

```
<패키지>
    <패키지 이름>
    </패키지 이름>
    <클래스 타입>
        <클래스 타입 정보>
        </클래스 타입 정보>
        <메소드>
            <메소드 정보>
            </메소드 정보>
        </메소드>
        <변수>
            <변수 정보>
            </변수 정보>
        </변수>
    </클래스 타입>
```

그림 2. xml 구조

3-5 연관 관계 도출

xml에 저장된 정보들 중에서 extends를 통해 상속받아야 할 클래스 이름과 implements를 통해 구현해야 할 클래스들의 이름을 통해 부모 클래스에서 상속받아온 함수 또는 변수인지, 인터페이스를 통해 추상 메소드를 오버라이드하여 구현해야할 함수인지 등을 찾아 정보를 저장한다.

Ⅳ. 구현 내용

```
public class A{
    // This is A class
    String str = "class";
    /*
        qwerty
        asdfgh
    */
    public void print() {
    }
}
```

그림 3 제거 전

```
public class A{
    String str = "";
    public void print() {
    }
}
```

그림 4 제거 후

그림 5 클래스 분할 전

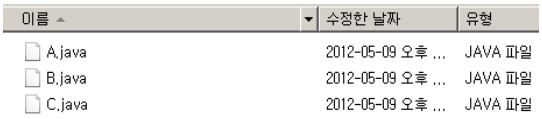

그림 6 클래스 분할 후

```
classHashmap["public"] = 0;
classHashmap["protected"] = 1;
classHashmap["private"] = 2;
classHashmap["static"] = 3;
classHashmap["final"] = 4;
classHashmap["abstract"] = 5;
classHashmap["extends"] = 6;
classHashmap["implements"] = 7;
classHashmap["class"] = 8;
classHashmap["interface"] = 9;
classHashmap["enum"] = 10;
```

그림 7 구문 분석 해쉬맵(클래스)

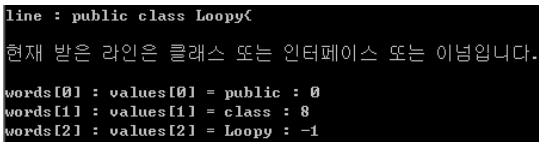

그림 8 구문 분석 결과 (클래스)

```
methodHashmap["public"] = 0;
methodHashmap["protected"] = 1;
methodHashmap["private"] = 2;
methodHashmap["final"] = 3;
methodHashmap["sychronized"] = 4;
methodHashmap["void"] = 5;
methodHashmap["boolean"] = 6;
methodHashmap["char"] = 7;
methodHashmap["byte"] = 8;
methodHashmap["short"] = 9;
methodHashmap["int"] = 10;
methodHashmap["long"] = 11;
methodHashmap["float"] = 12;
methodHashmap["double"] = 13;
methodHashmap["throws"] = 14;
methodHashmap["boolean[]"] = 15;
methodHashmap["char[]"] = 16;
methodHashmap["byte[]"] = 17;
methodHashmap["short[]"] = 18;
methodHashmap["int[]"] = 19;
methodHashmap["long[]"] = 20;
methodHashmap["float[]"] = 21;
methodHashmap["double[]"] = 22;
methodHashmap["static"] = 23;
methodHashmap["String"] = 24;
methodHashmap["String[]"] = 25;
methodHashmap["main"] = 26;
```

그림 9 구문 분석 해쉬맵 (함수)

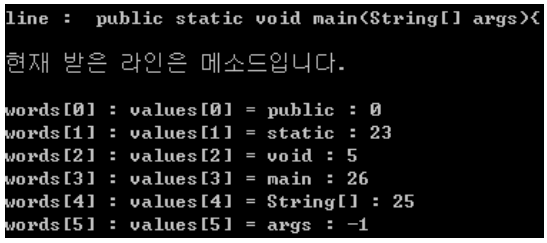

그림 10 구문 분석 결과(함수)

```
variableHashmap["public"] = 0;
variableHashmap["protected"] = 1;
variableHashmap["private"] = 2;
variableHashmap["static"] = 3;
variableHashmap["final"] = 4;
variableHashmap["abstract"] = 5;
variableHashmap["transient"] = 6;
variableHashmap["volatile"] = 7;
variableHashmap["boolean"] = 8;
variableHashmap["char"] = 9;
variableHashmap["String"] = 10;
variableHashmap["byte"] = 11;
variableHashmap["short"] = 12;
variableHashmap["int"] = 13;
variableHashmap["long"] = 14;
variableHashmap["float"] = 15;
variableHashmap["double"] = 16;
```

그림 11 구문 분석 해쉬맵(변수)

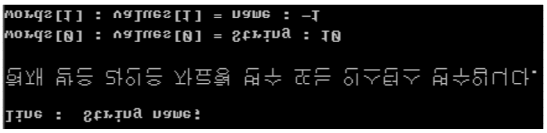

그림 12 구문 분석 결과(변수)

V. 결 론

리버스 엔지니어링은 프로그램을 주문한 오더와 그 프로그램을 직접 구현한 개발자가 놓치고 있던 불필요한 함수, 변수 등의 쓰임에 의한 자원 낭비를 줄이고, 이미 만들어진 시스템을 역으로 추적하여 소스 코드에서 필요한 최대한의 정보를 추출하여 소프트웨어의 재사용성을 높이고 유지보수를 용이하게 하는데 목적이 있다. 본 프로젝트에서 구현하고 있는 리버스 엔지니어링은 예약어 판별을 용이하게 하기 위해 주석과 스트링을 제거하는 단계, 내부 클래스와 하나의 클래스 안에 선언된 다수의 클래스를 독립된 클래스로 분할하는 단계, 라인을 받아와서 해쉬맵을 통한 분석단계, 분석단계를 통해 생성된 값들을 XML에 저장하는 단계, 저장된 XML을 통한 연관 관계를 도출하여 저장하는 단계로 구성된다.

참고문헌

[1] Kathy Sierra, Bert Bates / HeadFirst JAVA / 2011. 04. 28

[2] 남궁 성 / Java의 정석 / 2009. 01. 20