http://dx.doi.org/10.3745/KIPSTD.2012.19D.1.039

역공학을 이용한 자바 소스 코드의 변화량 분석 및 시각화 도구 개발

권 진 욱[†] · 최 윤 자[††] · 이 우 진[†††]

요 약

소프트웨어 시스템을 효율적으로 유지보수 및 관리하기 위해서는 변경 전후의 시스템의 변경사항을 쉽게 파악할 수 있도록 시각적으로 표현해주는 것이 중요하다. 소프트웨어 변경사항 분석에 대한 연구는 많이 진행되어 왔으나 변경 유형뿐만 아니라 변경 정도를 정량화하고 이를 시각화하여 나타내려는 연구는 많지 않다. 이 연구에서는 자바로 구현된 시스템에 대해 클래스 다이어그램에서의 변경사항 분석 및 정량화 방법과 이를 시각적으로 나타내는 방법에 대해 제시한다. 먼저 소스 코드의 구조적인 변화를 검사하기 위해, 역공학을 이용하여 클래스 다이어그램을 추출한다. 그리고 클래스 다이어그램 정보를 이용하여 변경 유형 및 변경량을 정량화한 다음, 색상 스펙트럼을 이용하여 클래스 다이어그램에 변경량을 시각적으로 나타낸다. 이러한 시각화 기법을 이용하면, 유지보수 관리자가 쉽게 변경된 부분을 파악할 수 있어 유지보수에 소요되는 시간과 노력을 조금이라도 줄일 수 있다.

키워드 : 변화량 분석, 변화량 시각화, 자바, 역공학

Development of Analysis and Visualization Tool for Java Source Code Changes using Reverse Engineering Technique

Jinwook Kwon[†] · Yunja Choi[††] · Woo Jin Lee[†††]

ABSTRACT

In order to quickly understand which changes of source codes have been made and to perform effective maintenance of a system, it is important to visualize the changed parts. Although there are many works for analyzing software changes, there are few works for visualizing both of the change types and change quantifications for Java based systems. In this paper, we propose a change analysis technique based on class diagram and provide a change visualization technique by using change quantification information. In order to check the structural changes in source codes, source codes are transformed to class diagrams by reverse engineering methods. On the class diagrams, the changes are analyzed and quantified by numbers. Based on the change quantification, the changes are visualized on the class diagram by color spectrum. By using visualization techniques, maintainers can easily recognize the code changes to reduce the cost and time of maintenance.

Keywords : Change Quantification Analysis, Change Visualization, Java, Reverse Engineering

1. 서 론

일반적으로 소프트웨어의 유지보수 비용은 개발 비용보다 더 많다[1]. 효율적인 시스템 유지보수 관리를 위해서는 사용자와 관리자 입장에서 시스템의 변경 사항들을 얼마나 빨리 파악하고 이해하는 것이 중요하다. 일반적으로 CVS (Concurrent Versions System), SVN(SubVersioN) 등의 버전관리 도구를 활용하여 결과물을 관리하는 경우에는 변경 이력을 파악할 수 있어서 소프트웨어의 변화량을 파악하는 것이 상대적으로 용이하다. 하지만 버전관리 도구를 활용하지 않고 변경사항을 기록에 남기지 않는 경우에는 변경된 부분을 파악하는 것이 쉽지 않다.

기존의 소프트웨어의 변경사항을 분석하는 방법은 문자 기반으로 변경사항을 분석하는 Tiff 방식, 추상구문트리 기반의 변경사항 분석 방식[2][3], XML(eXtensible Markup Language) 기반의 분석 방식[4] 등이 있다. 그리고 이러한

※ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.2010-0010606).
† 정 회 원 : LG전자 P1실 프레임워크 담당
†† 정 회 원 : 경북대학교 IT대학 컴퓨터학부 조교수
††† 정 회 원 : 경북대학교 IT대학 컴퓨터학부 부교수
논문접수 : 2011년 8월 5일
수 정 일 : 1차 2011년 11월 1일, 2차 2011년 11월 2일
심사완료 : 2011년 11월 2일

소프트웨어 변화량을 시각화하는 연구로는 시스템 진화과정을 보이기 위해 다수의 여러 버전간의 변화량을 시각화하는 연구[5][6][7]와 두 버전간의 차이를 시각화하는 연구[8][9]로 나뉠 수 있다. 시스템 진화 과정에서의 변화량의 시각화 연구로는 소프트웨어의 구조적 특성뿐만 아니라 프로그램 파일 수 또는 크기의 변화, 입출력 의존성 변화량 등을 Evolution Spectrographs[5], Evolution Storyboard[6] 등의 다양한 스펙트럼 형식의 그래프로 시각적으로 나타낸다. 두 버전간의 차이를 시각화하는 연구의 하나로 구조적 프로그램을 대상으로 의존성 그래프상에 변화량을 나타내는 CIG(Change Impact Graph) 기법[8]과 객체지향언어를 대상으로 클래스 다이어그램에서의 클래스의 추가, 삭제, 변경사항을 나타내는 연구[9]가 있다.

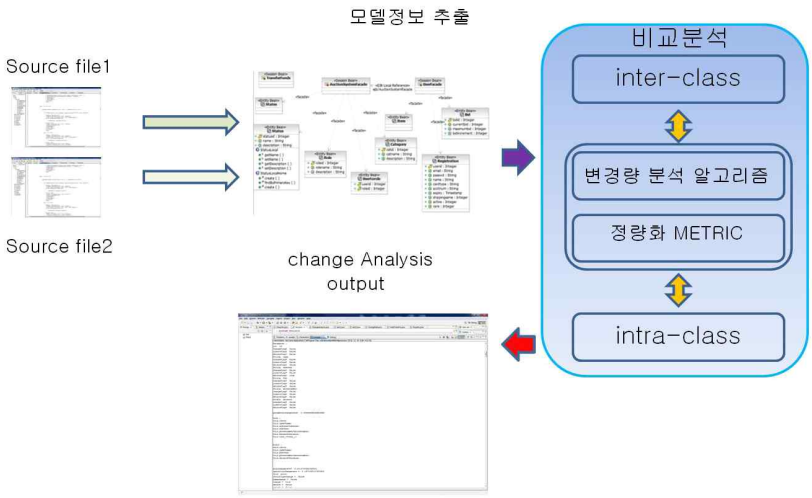
이 연구에서는 버전관리 도구를 사용하지 않는 개발 환경에서 객체지향언어인 자바로 구현된 소프트웨어의 두 버전이 존재할 때, 변경된 부분과 변경 정도를 시각적으로 나타내는 방법에 집중한다. 이 연구와 가장 관련이 많은 연구로는 클래스의 추가, 삭제, 변경사항을 나타내는 연구[9]를 들 수 있다. 하지만 이 연구에서는 단순히 추가, 삭제, 변경의 변경유형 정보만을 나타내며 변경이 되었을 경우에는 변수 이름의 변경과 같은 단순 변경인지, 제어 구조의 변경과 같은 심화 변경인지 등의 변경 정도에 대한 정보를 나타내지 못하는 단점이 있다. 본 연구에서는 변경 유형뿐만 아니라 변경 정도인 변화량을 분석하여 사용자가 쉽게 변경사항 및 정도를 파악할 수 있도록 시각화하는 것을 목표로 한다.

이 논문에서는 소프트웨어의 유지보수 입장에서 변경 전후의 프로그램의 변화 모습을 시각화 기법을 통해 표현함으로써 시스템이 구조적으로 어떻게 바뀌었는지 어떤 영향을 미치는지를 한눈에 파악할 수 있도록 새로운 모델을 제시하고자 한다. 먼저 변경된 소스코드를 역공학을 이용하여 클래스 다이어그램으로 표현하고 시각화함으로써 좀 더 직관적이고 구조적인 관점에서 프로그램을 바라볼 수 있는 방법을 제시한다. 변경 전후의 두 소스코드를 클래스 단위로 파싱하여 클래스내의 속성과 메소드의 정보들을 얻는다. 이렇게 얻어진 클래스 아웃라인 정보를 서로 비교하여 변화된 부분을 추출한다. 얻어진 모델 정보를 편집 거리(edit distance)라 불리는 레벤쉬테인 거리(Levenshtein distance)[10]에 근간을 둔 변화량 측정 규칙에 따라 서로 비교하고 변화된 부분을 테이블에 기록해 둔다. 마지막으로 클래스 다이어그램 편집기를 이용해서 소스 코드의 변화량을 다이어그램상에 시각화하여 변화된 부분과 변화량을 한눈에 파악할 수 있도록 한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 역공학을 이용한 두 모델 사이의 변경된 부분을 분석하기 위한 모델 생성에 필요한 여러 메트릭과 변화량 분석 알고리즘에 대해서 언급한다. 제 3 장에서는 모델 변화량 시각화 도구를 개발하기 위한 프레임워크에 대해서 설명한다. 제 4 장에서는 시각화 도구의 구현환경과 실제적인 변화량 분석을 통해 변경된 부분을 시각화해서 보여준다. 마지막으로 제 5 장에서 결론 및 향후 연구를 기술한다.

2. 역공학 모델을 이용한 변경분석

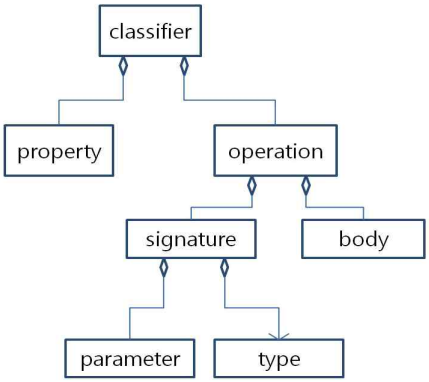
자바와 같은 객체지향 프로그램은 클래스 단위로 되어 있고 클래스 내부에는 여러 개의 메소드들이 존재하는 계층구조이다. 이러한 계층구조로 된 프로그램을 텍스트 중심으로 비교하여 변경사항을 파악하는 것은 어려움이 많아 본 연구에서는 기존의 역공학 기술을 이용하여 소스 코드를 클래스 다이어그램으로 구조화한 다음, 두 클래스 다이어그램의 차이점을 비교한다. 전체적인 과정은 (그림 1)과 같이 크게 두 부분으로 이루어진다. 먼저 역공학을 이용해 현재 시스템의 변경전과 후의 소스코드로부터 모델정보를 추출한다. 그리고 추출된 두 개의 모델로부터 변경된 부분을 분석하고 변화량을 정량화한다. 이러한 과정을 통해 변경 전후의 시스템의 구조정보를 비교분석하고 현재 변경된 부분이 미치는 영향 즉, 연관관계의 정보를 알 수 있다.



(그림 1) 역공학 모델을 이용한 변경분석

2.1 역공학 기반 모델 생성

소스 코드로부터 클래스 다이어그램 모델을 생성하기 위해 역공학을 이용한다. 소스코드에서 모델요소를 추출하기 위해 파서를 생성한다. 파서는 클래스 간의 연관관계 정보와 클래스 내부의 속성 정보들을 추출한다. (그림 2)는 클래스의 주요 내부 구성요소들을 나타낸 것이다. 클래스는 크게 선언부분(Classifier), 속성, 메소드로 구성된다. <표 1>은 선언부분과 속성의 모델 정보를 나타낸다.



(그림 2) 클래스의 주요 구성요소

〈표 1〉 클래스의 선언부분과 속성의 모델 정보

1	Name Of Class(NOC)	클래스 자체의 이름을 선언부분으로부터 얻는다.
2	Number Of Property(NOP)	클래스 내부 속성의 숫자를 나타낸다. 후에 변화량 분석을 위해 NOP를 참조한다.
3	Hidden Of Property(HOP)	속성은 캡슐화 개념에 따른 공개여부를 나타내는 public, private, protected 요소를 나타낸다.
4	Type Of Property(TOP)	TOP는 라이브러리에서 제공하는 기본 타입과 사용자 정의 타입으로 구분한다. 사용자 정의타입은 후에 클래스간의 연관 관계와 집합 연관을 찾기 위해 사용된다.
5	Name In Property(NIP)	속성의 이름을 나타낸다.
6	Array Type Of Property(AOP)	속성이 배열로 선언되었는지를 나타낸다.
7	List Type Of Property(LOP)	속성이 리스트로 선언되었는지를 나타낸다.

속성타입의 경우는 라이브러리에서 제공하는 기본타입과 사용자 정의타입으로 이루어져 있다. 메소드는 함수원형과 바디로 구성되며 함수원형에는 공개여부와 리턴 타입, 메소드명, 그리고 매개변수로 구성된다. <표 2>는 메소드로부터 모델 정보를 나타낸다.

클래스 간의 연관관계를 나타내는 모델 정보는 <표 3>과 같다. 클래스간의 전체-부분 관계와 연관관계일 경우에는 연관된 객체가 배열 또는 여러 객체로 표현되어 있는지 파악하여 기수성을 분석한다.

〈표 2〉 메소드의 모델 정보

1	Hidden Of Operation(HOO)	public, private, protected의 메소드의 공개여부를 나타낸다.
2	Type Of Operation(TOO)	메소드의 리턴 타입을 나타낸다.
3	Name In Operation(NIO)	메소드의 이름을 나타낸다.
4	Parameters In Operation(PIO)	메소드의 매개변수를 나타낸다. 숫자를 구한다. 후에 변화량 분석을 위해 사용된다.
5	Body Of Operation(BOO)	메소드의 몸체 부분을 나타낸다.

〈표 3〉 클래스 관계를 정의한 모델 정보

1	Generation Of Class(GOC)	두 클래스간의 일반화 관계를 정의하기 위해 상속관계를 확인한다. 클래스의 상속여부를 선언부분에 나타난 extends와 인터페이스의 구현부분인 Implements 유무를 통해 얻는다.
2	Association Of Class(AOC)	두 클래스 간의 일반적인 협력관계가 성립하는지 여부를 확인하기 위해 다른 클래스를 참조하는지 TOP를 살펴봄으로써 확인한다.
3	Aggregation Of Class(AGOC)	전체-부분 관계가 있는지를 살펴보고 독립적인 생명주기를 가지는지를 확인한다. TOP를 살펴봄으로써 확인한다.
4	Dependency Of Class(DOC)	클래스간의 사용관계를 살펴보기 위해 메소드의 몸체에서 지역변수로 선언되었는지 매개변수의 타입으로 다른 클래스를 참조하는지를 확인한다. TOPIO와 BOO살펴봄으로써 확인한다.

이러한 모델 정보들을 소스코드로부터 추출하기 위해 자바언어의 문법을 고려해 자바 라이브러리에서 제공하는 Lexical 함수인 패턴(Pattern)과 매춰(Matcher)를 이용하여 파서를 구현한다. 파서를 통해 얻은 모델 요소들은 트리 형태로 저장된다.

2.2 최소편집거리를 이용한 변화량 분석

레벤쉬테인 거리(Levenshtein distance)[10]는 두 시퀀스 간의 최소 편집거리를 나타내며 편집 거리(edit distance)라고도 불린다. 어떤 문자열을 다른 문자열로 바꾸는데 필요한 삭제, 삽입, 대체의 작업으로 이루어지며 이러한 작업을 하는데 최소한으로 필요한 편집 횟수를 나타낸다. 동적 프로그래밍을 이용한 편집 거리는 두 문자열을 받아 n x m 사이즈의 행렬을 이용하여 계산한다. 이렇게 구해진 최소편집거리는 두 문자열의 유사성을 계산하는데 쓰인다. (그림 3)은 최소편집거리를 구하는 편집 거리 알고리즘이다.

최소편집거리는 두 값 사이의 차이를 나타내므로 비교 가능한 수치로 표현하기 위해 아래 정의 1과 같이 전체 문자열의 길이 값으로 나눈 변화율로 나타낸다.

정의 1. 두 문자열간의 차이를 정량화하기 위해서는 최소편집 거리를 전체 비교 문자열의 최대 길이로 나누어야 한다.

문자열 변화율 = 편집거리 / 긴 문자열의 길이

```
Function editoperation(i,j){  //deit operation
        // EDIT is a matrix of integers (m X n)

        if (i=0 and j=0){
                return;
        }

        if (i=0){
                editoperations(i, j-1);
                insertion;
        }

        else if (j=0){
                editoperations(i-1, j);
                deletion;
        }

        else{
                min value =min(EDIT[i-1,j]+1, EDIT[i,j-1]+1, EDIT[i-1,j-1]+∂(x[i],y[j]))

                if (min value = EDIT[i-1,j-1]+∂(x[i],y[j])){
                        editoperations(i-1, j-1);
                        susbtitutioin;
                }

                else if (min value = EDIT[i,j-1]+1){
                        editoperations(i, j-1);
                        insertion;
                }

                else{
                        editoperations(i-1, j);
                        deletion;
                }
        }
}
```

(그림 3) 편집 거리 알고리즘

예를 들어, example1의 문자열이 example5로 변경되었다면, 한글자만 변경하면 되므로 최소편집거리는 1이며 전체 문자열의 길이는 8이므로 문자열 변화율은 1/8이 된다. 일반적으로 문자열은 한 글자씩을 의미 있는 단위로 간주하지만 여기에서는 키워드, 자료유형, 인자 이름 등과 같이 분할할 수 없는 단어는 하나의 문자로 간주한다. 그리고 전체문자열을 고려할 때도 형식에 의해 반드시 포함되어야 하는 공백, 콤마, 괄호 등은 고려하지 않는다. 예를 들어, 함수 void ex (string name, int count)에서 void ex(string n1, double count)으로 변경되었다면, 첫번째는 함수인자 이름이 변경되었고 두번째 인자는 타입이 변경되었으므로 인자리스트의 최소편집거리는 2이며 전체 문자열의 길이는 4이므로 인자리스트의 변화율은 2/4가 된다.

2.3 함수의 변화량 분석

일반적으로 함수는 헤더 부분과 몸체 부분으로 나뉘어진다. 하지만 변경관리 관점에서는 함수의 인자와 몸체 간에는 밀접한 연관 관계가 있으나 함수이름과 몸체와는 연관성이 떨어진다. 예를 들어 함수 인자가 바뀌면 몸체도 바뀌게 되지만 함수 이름은 몸체 수정 없이도 변경 가능하다. 그러므로 함수 간의 변경 분석에는 함수이름, 인자 리스트, 몸체를 각각 고려하여야 한다. 아래 정의 2는 함수의 변화율을 나타낸다.

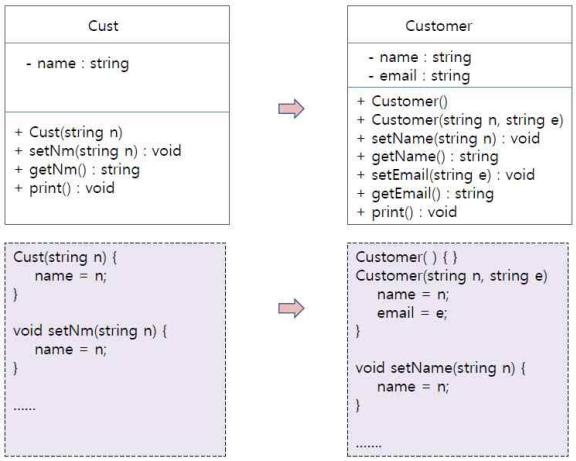
정의 2. 함수의 변화율은 함수의 이름, 리턴 타입 및 인자 리스트, 몸체의 각 변화율을 더한 것을 3.0으로 나눈 값이다. 여기에서 fNameRate는 함수 이름의 문자열 변화율, paraReturnRate는 함수이름일 뺀 함수헤더의 문자열 변화율, bodyRate는 함수 몸체의 문자열 변화율을 나타낸다.

FuncChangeRate = (fNameRate + paraReturnRate + bodyRate) / 3.0

예를 들어, 함수 void ex1(int x, int y)에서 int ex11(double x, double y)로 변경되었으며 함수의 몸체는 문자열 변화율이 30%라고 가정해보자. 먼저 함수이름의 변화율은 1/4이며, 리턴타입과 함수인자 리스트의 변화율은 3/5이므로 전체 함수 변화율은 (0.25 + 0.6 + 0.3)/3 = 0.38이다.

2.4 클래스간의 변화량 분석

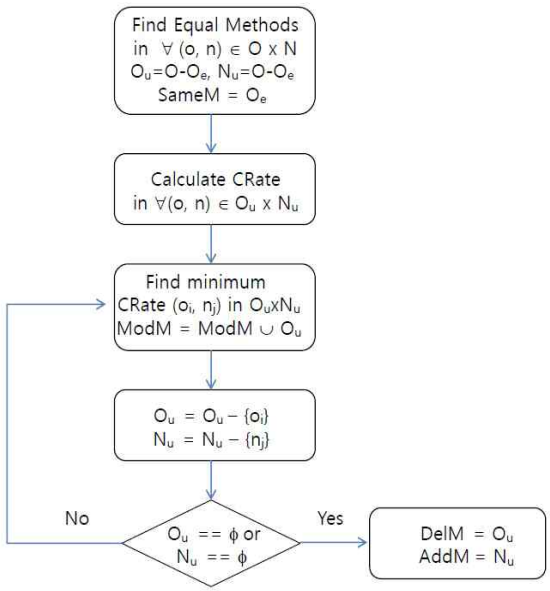
클래스는 클래스 이름, 애트리뷰트, 메소드 리스트로 구성되어 있다. 수정 전후의 두 클래스가 동일한지 여부를 확인



(그림 4) 클래스 다이어그램 예제

하기 위해서는 이러한 3가지 항목들을 비교하여야 한다. (그림 4)는 고객 정보를 나타내는 클래스의 변경 전후의 정보를 보여준다. 클래스 이름이 Cust에서 Customer로 변경되었으며 애트리뷰트로 email 이 추가되었으며 디폴트 생성자와 email의 set/get 메소드가 추가된 것을 알 수 있다.

클래스 이름과 애트리뷰트의 변화량은 앞서 정의한 문자열 변화량을 이용하여 계산할 수 있다. 위 예제에서는 클래스 이름의 문자열 변화율은 4/8이며 애트리뷰트의 변화율은 2/4이다. 하지만 메소드의 변화량을 검사하는 것은 단순하지 않다. 오버로딩된 메소드가 여러 개 있는 경우나 메소드의 시그너처가 많이 변경된 경우는 어느 메소드가 추가되고 삭제되었는지, 어느 메소드가 얼만큼 변경되었는지를 판단하는 것은 단순하지 않다. (그림 5)는 메소드 리스트에서 어느 메소드가 수정되었으며, 어느 메소드가 추가 또는 삭제되었는지 판단하는 알고리즘을 순서도로 나타낸다.



(그림 5) 메소드 리스트 매칭 알고리즘

먼저, 두 메소드 리스트에서 일치되는 메소드들을 찾아낸다. (그림 4)의 예제에서는 print()의 몸체가 일치한다고 가정하면 print()만이 일치한다. 일치되지 않는 나머지 메소드들은 변경전의 메소드는 O_u={Cust(string n), setNm(string n), getNm()}, 변경후의 메소드는 N_u={Customer(), Customer(string n, string e), setName(string n), getName()}로 나타낸다. O_u와 N_u의 메소드들 중에서 변화율이 가장 낮은 메소드를 찾아서 변경된 메소드로 표시한다. 예제에서는 getNm() → getName(), setNm(string n) → setName(string n)은 몸체와 인자리스트는 동일하고 이름만 변경되었으므로 가장 낮은 변화율인 (2/7+0+0)/3 = 2/21을 갖는다. Cust(string n) → Customer()는 (4/8+2/2+3/3)/3 = 5/6, Cust(string n) → Customer(string n, string e)는 (4/8+2/4+3/6)/3 = 1/2 이므로 변화율이 작은 Cust(string n) → Customer(string n, string e)로 변경되었다고 판단한다. 이러한 과정은 O_u 또는 N_u가 공집합이 될 때까지 진행한다. 만약 O_u에 메소드가 남아 있으면 삭제된 메소드를 의미하며 N_u에 메소드가 남아 있으면 추가된 메소드를 의미한다. 위의 예제에서는 N_u에 남아 있는 Customer() 메소드가 추가되었다고 판단한다. 앞서 찾은 변경된 메소드들 중에서도 변화율이 지나치게 높은 것들은 변경되었다기 보다는 삭제/추가된 메소드로 간주하는 것이 타당할 수 있다. 이러한 기준치 변화율은 사용자가 지정하도록 한다. 메소드 리스트의 변화율은 아래의 정의3과 같이 정의한다.

정의 3. 메소드 리스트의 변화율은 메소드에서 삭제된 메소드의 수와 추가된 메소드의 수, 그리고 변경된 메소드의 변화량의 합을 더한 다음, 추가 / 삭제 / 변경된 메소드의 개수로 나눈 값이다. 여기에서 numD는 삭제된 메소드의 수, numA는 추가된 메소드의 수, numC는 변경된 메소드의 수를 나타낸다.

methodRate =

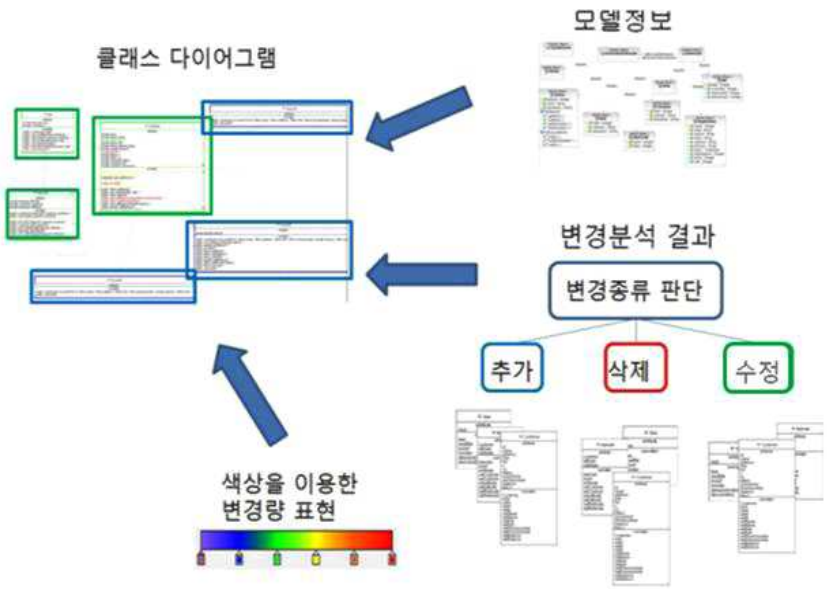
$$(numD + numA + \sum_{i \in Changed} FuncChangeRate_i) \ / \ (numD + numA + numC)$$

시스템의 변경사항을 분석하기 위해서는 어떠한 클래스가 삭제되고 추가되었는지, 또는 기존의 클래스가 얼마나 변경되었는지, 클래스간의 연관관계가 얼마나 변경되었는지를 분석하여야 한다. 다음 정의4는 클래스의 변화율을 정의하고 있다.

정의 4. 클래스의 변화율은 클래스의 이름, 애트리뷰트, 메소드의 각 변화율을 더한 것을 3으로 나눈 값이다. 여기에서 cNameRate는 클래스 이름의 문자열 변화율, attributeRate는 애트리뷰트의 문자열 변화율, methodRate는 메소드 리스트의 변화율을 나타낸다.

ClassChangeRate = (cNameRate + attributeRate + methodRate) / 3.0

두 클래스 다이어그램의 변경사항을 판단하는 과정은 메소드 리스트의 매칭 알고리즘과 유사한 방법으로 구할 수 있다. 메소드 리스트의 매칭 알고리즘에서 메소드 대신에 클래스를 넣으면 된다. 클래스 간의 변경사항은 단순히 클래스의 변경사항 뿐만 아니라 클래스간의 연관관계의 변화도 검사하여야 한다. 클래스 간의 관계에 대한 추가, 삭제, 변경은 다음과 같이 판단한다. 앞서 정의한 GOC (Generalization Of Class), AOC(Association Of Class), AGOC(Aggregation Of Class), DOC(Dependency Of Class) 요소를 위에서 언급한 클래스 전체의 추가, 삭제, 변경 과정



(그림 6) 클래스 다이어그램을 이용한 시각화 도구

을 참고하여 삭제된 관계와 추가된 관계를 판단한다. 관계에서의 변경은 클래스 내부의 변경으로 인한 관계의 종류가 바뀐 것으로 판단한다. 클래스 간의 관계 변경은 클래스의 변화량에 영향을 미치지 않으며 단지 클래스 다이어그램 상에 시각적으로 변경사항을 표시한다.

3. 모델 변화량 시각화 도구의 설계

앞 장에서 제시하였던 모델간의 변경분석 내용을 시각화를 통해서 UML 클래스 다이어그램으로 표현하여 변경내용을 좀 더 직관적으로 한눈에 알아볼 수 있도록 한다. 모델 변화량 시각화 도구를 지원하기 위해서는 역공학을 통해 추출된 클래스 다이어그램 정보를 나타낼 수 있는 클래스 다이어그램 편집기가 필요하며 분석된 변화량의 수치를 시각적인 스펙트럼으로 나타내는 기법이 필요하다. (그림 6)은 변화량 시각화 도구의 전체적인 프로세스를 나타낸다. 역공학으로 추출된 모델 정보, 변경사항 정보, 그리고 각 변경사항의 변화량 정보 등을 시각적으로 클래스 다이어그램에 나타낸다.

3.1 GMF를 이용한 다이어그램 편집기 생성

변경분석 내용을 클래스다이어그램으로 나타내기 위하여 이클립스 기반의 플러그인 개발 도구인 GMF[11]를 이용한다. GMF는 EMF(Eclipse Modeling Framework)[12]와 GEF(Graphical Editing Framework)[13]에 기반을 둔 그래픽 편집기 제작을 위한 프레임워크이다. 앞서 설명한 파서를 통해 모델을 생성하고 두 모델간의 변경내용을 시각화하기 위해서는 GMF 편집기에서 정의된 포맷으로 변경정보를 변환해야 한다. GMF 다이어그램 편집기는 일반적으로 XMI 형태의 문서 포맷으로 이루어져 있다. 일반적으로 트리 형태의 클래스 모델구조와 연관관계에 따른 아이디 할당은 GMF 프레임워크가 자동으로 변환을 해준다. 즉 다이어그램에 필요로 하는 요소들만 XMI 파일 형태로 변환을 해주면 나머지는 GMF가 알아서 포맷을 변환한다. 클래스 다이어그램에서 표현되는 모델 정보로는 클래스 노드와 클래스가 포함하고 있는 속성, 메소드 및 클래스 간의 연관관계 정보들이다. 포맷으로 변환을 할 때 주의해야 할 사항은 클래스간의 연관관계를 관계가 있는 클래스의 이름으로 표현하지 않고 아이디로 값을 가지고 있기 때문에 처음 파일에 입력할 때 모델로 변환하고자 하는 순서를 기억하고 연관관계를 지정해줘야 한다는 것이다. 순서가 달라지면 관계정보가 바르게 표현이 되지 않을 수 있기 때문에 주의해야 한다. GMF를 이용하여 다이어그램 편집기를 생성하는 상세한 과정은 [14]에 소개된 과정을 따랐다.

3.2 변화량 시각화 방법

앞 절에서 설명한 GMF로 생성한 클래스다이어그램을 이용하여 두 버전의 모델 간 변경분석 내용을 시각화하기 위해 시각화 규칙을 정의한다. 다이어그램 편집기에 나타날 수 있는 클래스 노드 자체와 그 안에 포함된 속성, 메소드 그리고 클래스 간의 연관관계로 나타날 수 있는 상속관계, 연관관계, 집합연관, 의존성에 대해서 추가/삭제/변경에 따른 구분을 색깔을 다르게 입힘으로써 표현하고자 한다. 표현방법은 아래와 같다.

- 추가된 부분 : 추가된 부분은 새로운 느낌의 파란색을 사용한다.
- 변경된 부분 : 위의 변경분석을 통해 정량화된 변화량을 고려하여 안정감 있는 짙은 녹색 계열에서 가장 많이 변화된 부분은 점점 붉은색을 띄는 스펙트럼으로 나타낸다.

- 삭제된 부분 : 가장 많이 변경된 부분으로 짙은 붉은색으로 표시한다. 변경된 부분의 스펙트럼은 0.1을 단위로 총 10가지의 스펙트럼을 가지고 표현한다.

(그림 7)은 표현하고자 하는 스펙트럼의 대표적인 값을 나타낸다. 변화량이 가장 작은 왼쪽의 파란색에서, 초록색, 노란색, 가장 오른쪽의 빨강색의 스펙트럼까지 나타낸다.

(그림 7) 시각화 스펙트럼

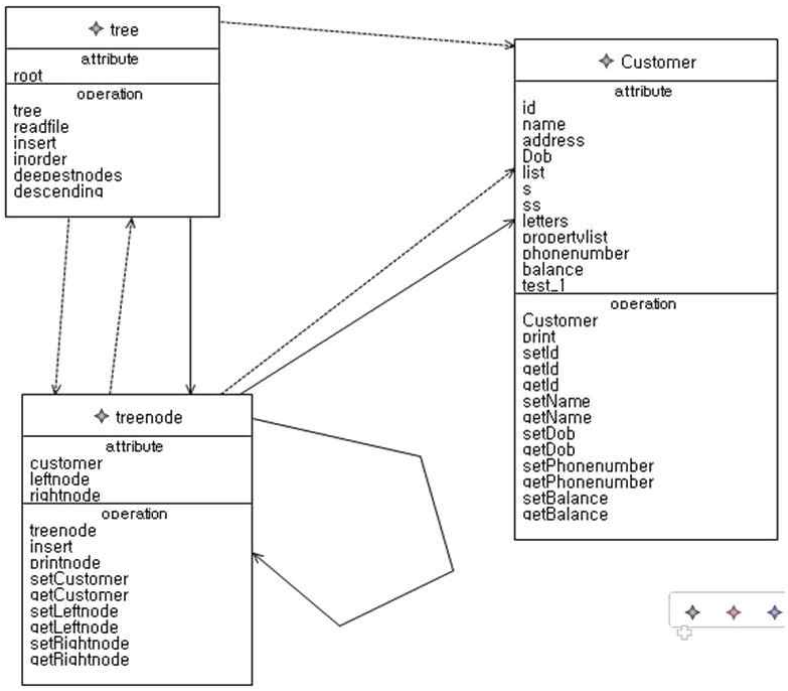
4. 시각화 도구의 구현 및 적용

이 장에서는 제시한 변경분석 및 시각화 도구 개발방법으로 도구를 구현하고 간단한 예제 프로그램으로 테스트를 해본다. 실제 개발 과정은 입력받은 서로 다른 버전의 프로그램으로부터 모델을 생성하고 두 모델간의 변경부분을 변경분석 알고리즘을 통해 추출하고 개발한 클래스 다이어그램의 포맷으로 변경한 후, 시각화를 통해 최종적으로 표현하는 과정으로 이루어진다. 자바 코드의 변화량을 분석과 시각화 도구를 개발하기 위해 이클립스 자바 프레임워크인 Europa를 사용하고 플러그인으로 EMF와 GMF를 사용한다. 자바 코드의 변화량을 분석과 시각화 도구를 개발하기 위해 이클립스 자바 프레임워크인 Europa를 사용하고 플러그인으로 EMF와 GMF를 사용한다.

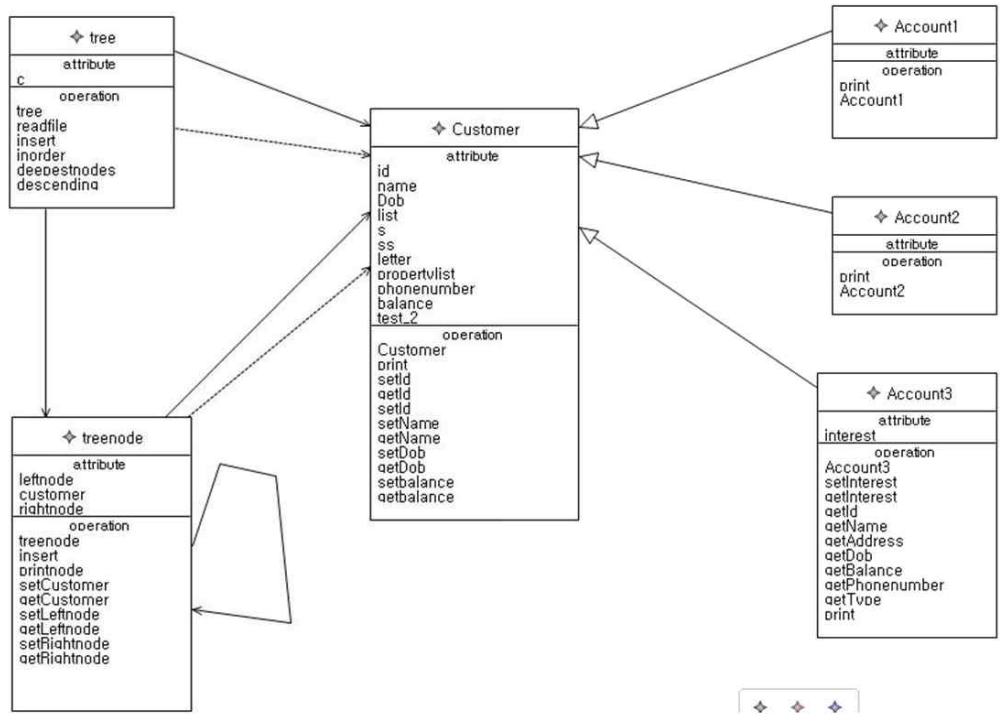
자바로 구현한 은행의 고객 관리 프로그램의 이전 버전과 다음 버전간의 변화를 변화량 분석을 통해 시각화 한다. 은행의 고객 관리 프로그램은 처음 3개의 클래스인 Customer, Tree, TreeNode로 이루어져 있다. 다음 버전은 Account, Account2, Account3 클래스가 추가되어 총 6개의 클래스로 이루어져 있고 두 버전 간에 속성과 메소드에 변경을 가하였다. 또한 클래스간의 관계 또한 조금 변경된다. 속성의 변화량과 메소드의 변화량을 변경 분석 알고리즘을 통해 그 변화량을 구해보고 얻어진 변화량을 통해 GMF 다이어그램 편집기로 시각화 해본다. (그림 8)은 변경 전후의 클래스 다이어그램을 보여준다.

두 버전 사이에 일어난 변화를 몇 가지로 나누어 짚어보면 아래와 같다.

- 기존 Customer 클래스를 상속받은 Account1, Account2, Account3 클래스가 추가되었다. 이러한 클래스는 파란색으로 추가되었음을 나타낸다.
- Tree 클래스와 Customer 클래스에 변경사항이 발생하였다. 애트리뷰트의 일부가 삭제 또는 추가되었으며 해당 애트리뷰트의 set/get 메소드들 또한 추가 및 삭제되었다.
- Customer 생성자와 print 메소드는 매개변수의 변경으로 인한 함수원형과 바디 모두 변화가 일어났으며 전체 변화량에 따라 색상을 다르게 나타냈다.
- Customer 클래스의 GetId 메소드가 변경전에는 오버로딩되어 2개로 존재하였으나 변경 후에서는 하나가 삭제되고 하나만 남았다. 삭제된 메소드를 구분하기 위해 변화량을 이용해서 알아낸다.
- Customer 클래스에는 기존의 setId 메소드와 별도로 새로운 setId 메소드가 오버로딩되어 추가되었다.

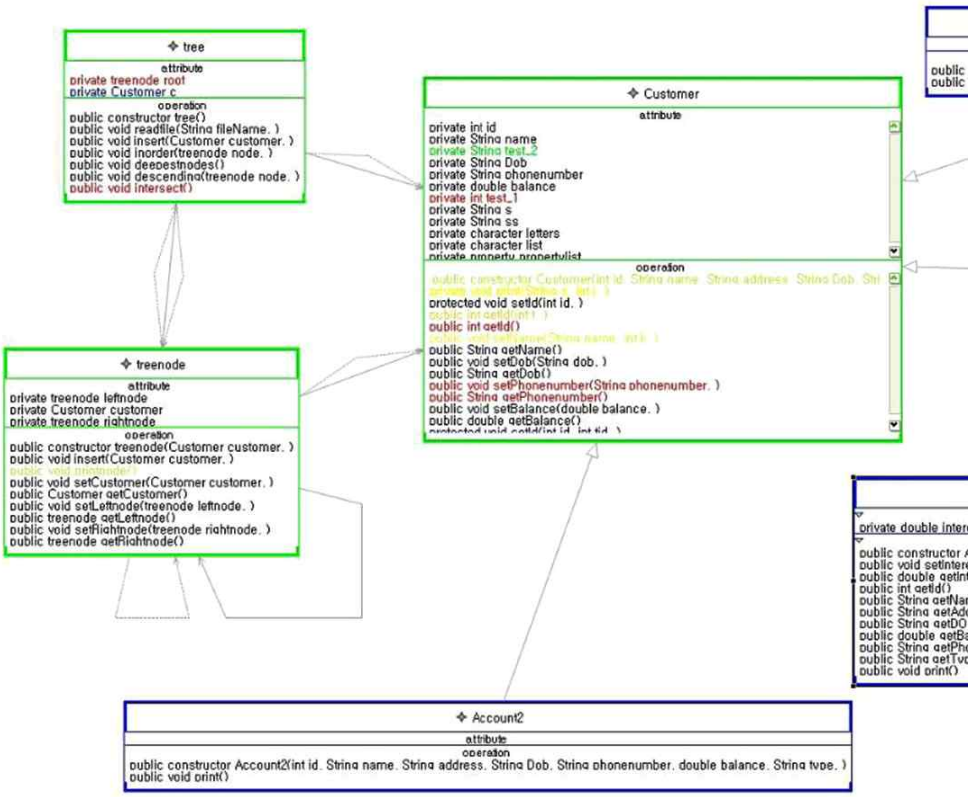


(a) 변경전 클래스 다이어그램



(b) 변경후 클래스 다이어그램

(그림 8) 고객관리 시스템의 변경 전후의 클래스 다이어그램



(그림 9) 시각화 도구를 통한 모델 간 변경 분석

(그림 9)는 위와 같은 클래스 다이어그램의 변경사항들을 정량화하여 클래스 다이어그램상에 시각화하여 나타낸 것이다. 그림에서 Account1, Account2, Account3 클래스는 새롭게 추가되어 푸른색으로 표시되었으며, 기존의 클래스 중에서는 Customer 클래스가 가장 많이 변경되었어 노란색으로 표시되어 있다. Customer 클래스 내부에는 새롭게 추가되어 녹색으로 표시된 속성과 삭제되어 붉은색으로 표시된 속성이 있으며 노란색으로 표시되어 수정된 메소드와 붉은색으로 표시된 삭제된 메소드들이 있다. 그리고 클래스 간의 관계가 추가 및 삭제된 경우는 색상을 달리 표시합니다.

5. 결 론

본 논문에서는 소프트웨어의 유지보수 측면에서 발생하는 구조적인 변화를 쉽게 파악할 수 있도록 하기 위하여 클래스 다이어그램 상에서 변경량을 시각화하는 방법을 제안하였다. 두 버전간의 변경부분을 모델로 변환하고 분석 알고리즘을 통해 변화량을 정량화했으며 클래스 다이어그램을 통해 시각화해 줌으로써 구조적인 변화를 한눈에 파악할 수 있도록 하였다. 단순 변경 유형만을 나타내는 기존 연구의 한계를 극복하여 변경 정도를 정량적으로 나타내고 역공학으로 생성한 모델에 시각화하여 나타내었다. 또한 객체지향 언어인 자바 개발환경 이클립스에 적합하게 파서를 개발하였고 이클립스 플러그인인 GMF 클래스 다이어그램 편집기를 통해 시각화해줌으로써 시각화 도구의 활용성을 높였다.

향후 연구로는 자바 언어에 국한하지 않고 C++와 같은 다른 객체지향언어의 변경 분석으로 그 영역을 확장할 필요가 있다.

참 고 문 헌

[1] Ian Sommerville, Software Engineering, 9th Edition, Pearson, 2011.

[2] Beat Fluri, Michael Wursch, Martin Pinzger, Harald C. Gall, “Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction,” IEEE Transactions on Software Engineering, Vol.33, No.11, Nov., 2007

[3] Masatomo Hashumoto, Akira Mori, “Diff/TS: A Tool for Fine-Grained Structural Change Analysis,” Proceedings of the 15th Working Conference on Reverse Engineering, 2008.

[4] Jonathan I. Maletic, Michael L. Collard, “Supporting Source Code Difference Analysis,” Proceedings of the 20th IEEE International Conference on Software Maintenance, 2004.

[5] Jingwei Wu, Claus W. Spitzer, Ahmed E. Hassan, and Richard C. Holt, “Evolution Spectrographs: Visualizing Punctuated Change in Software Evolution,” Proceedings of the 7th International Workshop on Principles of Software Evolution, 2004.

[6] Dirk Beyer, Ahmed E. Hassan, “Animated Visualization of Software History using Evolution Storyboards,” Proceedings of the 13th Working Conference on Reverse Engineering, 2006.

[7] Tom Arbuckle, “Visually Summarising Software Change,” Proceedings of the 12th International Conference on Information Visualization, 2008.

[8] Daniel M German, Gregorio Robles, and Ahmed E. Hassan, “Change Impact Graphs: Determining the Impact of Prior Code Changes,” Proceedings on 8th IEEE International Working Conference on Source Code Analysis and Manipulation, 2008.

[9] Jochen Seemann, Jurgen Wolff von Gudenberg, “Visualization of Differences between Versions of Object-Oriented Software,” Proceedings of the Second Euromicro Conference on Software Maintenace and Reengineering, 1998.

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

[11] Eclipse, Graphical Modeling Framework(GMF), http://www.eclipse.org/modeling/gmf/

[12] Eclipse, Eclipse Modeling Framework(EMF), http://www.eclipse.org/modeling/emf/

[13] Eclipse, Graphical Editing Framework(GEF), http://www.eclipse.org/gef/

[14] 박인수, 이정선, 조성래, 정우영, 이우진, “AUTOSAR 기반 차량용 소프트웨어 컴포넌트 모델링 도구”, 정보처리학회 논문지A, 제17-A권, 제4호, pp. 203-212, 2010년 8월.

권 진 욱

e-mail : geno.kwon@gmail.com

2009년 경북대학교 전자전기컴퓨터학부 (학사)

2011년 경북대학교 전자전기컴퓨터학부 (공학석사)

2011년～현 재 LG전자 P1실 프레임워크 담당

관심분야 : 임베디드 소프트웨어공학, 모바일 소프트웨어

최 윤 자

e-mail : yuchoi76@knu.ac.kr

1991년 연세대학교 수학과(이학사)

1993년 연세대학교 수학과(이학석사)

1993년～1996년 삼성데이터시스템

1999년 미네소타대학 전산과(이학석사)

2003년 미네소타대학 전산과(공학박사)

2003년～2006년 독일 프라운호퍼연구소 연구원

2006년～현 재 경북대학교 IT대학 컴퓨터학부 조교수

관심분야 : 소프트웨어 안전성 분석, 모델기반개발방법론

이 우 진

e-mail : woojin@knu.ac.kr

1992년 경북대학교 컴퓨터과학과(학사)

1994년 한국과학기술원 전산학과 (공학석사)

1999년 한국과학기술원 전산학과 (공학박사)

1999년~2002년 한국전자통신연구원 S/W공학연구부 선임연구원

2002년~현 재 경북대학교 IT대학 컴퓨터학부 부교수

관심분야 : 임베디드 소프트웨어 모델링 및 분석, 정형적 방법, 임베디드 소프트웨어 테스팅 등