韓國 OA學會 論文誌
第7卷 第1號, 2002. 3.

2002-7-1-1-4

계승관계에서 구성원 함수 수준의 변경 영향 분석

방 정 원*

Change Analysis for Inheritance Relation in Method Level

Jung-Won Bang*

요 약

소트트웨어의 재사용은 소프트웨어의 위기를 맞으면서 프로그래머의 생산성을 향상시키기 위한 방안으로 주목받게 되었다. 이러한 배경에서 탄생한 객체 지향 기술은 소프트웨어 분석에서부터 프로그래밍 언어의 선택, 테스트, 유지 보수에 이르기까지 소프트웨어 엔지니어링 전 영역에 걸쳐 영향을 미쳤다. 클래스, 계승관계, 추상화 등의 새로운 개념들은 새로운 테스트 문제를 야기 시켰을 뿐만 아니라, 객체 지향 프로그램의 수정 테스트에도 새로운 문제들을 던져 주었다. 수정 테스트의 첫 번째 문제는 일부의 변경으로 인해 영향받는 부분들을 찾아내는 것이다. 이 논문에서는 계승관계에 있어 영향받는 부분들을 구성원 함수 수준에서 찾아내는 알고리즘을 제안하고 있다.

Abstract

Software reuse was focused for the way of improving programmer productivity from the crisis of software. Object oriented technology impact on overall area of software engineering, such as software analysis, programming language, testing and maintenance.

The new concepts, Class, Inheritance and encapsulation, not only introduce new testing problems and they raise a new challenging question of how to conduct regression testing for O-O programs. the first problem of regression testing is how to identify the affected components due to the changes of some components. We propose a method firewall to enclose all classes and methods affected by the changes to one or more methods in inheritance relation

* 청강문화산업대학 컴퓨터소프트웨어과 조교수

논문접수 : 2002. 1. 5
심사완료 : 2002. 2. 27

Ⅰ. 서론

소프트웨어의 재사용은 소프트웨어의 위기를 맞으면서 프로그래머의 생산성을 향상시키기 위한 방안으로 주목받게 되었다. 이러한 배경에서 탄생한 객체 지향 개념은 소프트웨어 엔지니어링 전반에 걸쳐 영향을 미쳤다. 클래스, 계승관계(inheritance), 추상화(encapsulation)등의 새로운 개념들로 인해 클래스와 그들의 속성간에는 복잡한 관계가 형성되었다. 이로 인해 객체 지향 소프트웨어의 오류발생의 기회는 더욱 증가 되었다. 계승관계, 동적 바인딩, 다형 현상(polymorphism)등으로 수행 가능한 경로의 숫자를 현저히 증가 시켰으므로, 기존의 코드 분석에 따른 테스트 방법만으로는 해결할 수 없는 새로운 테스트 문제가 발생되었다. 수정 테스트(regression test)의 경우에는 더욱 문제가 된다. 전통적인 테스팅 방법들은 제어 흐름 모델에 기초를 두고 있으나 클래스 객체는 다양하게 변화하는 상태 의존적인 행위를 갖고 있기 때문이다.

수정 테스트의 목적은 수정된 프로그램이 여전히 프로그램 요구사항을 만족한다는 것을 보여 주는데 있다. 테스트에 드는 비용을 절약하려면, 어떤 컴포넌트의 변경으로 인해 영향받는 컴포넌트만을 찾아내야 한다.

이 논문에서는 계승관계에 있는 클래스의 구성원 함수에 변경이 일어났을 때, 변경으로 인해 영향을 받는 클래스들을 밝혀내고, 그 클래스들 중에서도 변경의 영향을 직접적으로 받는 구성원함수만을 찾아내는 방법을 제안하고자 한다.

Ⅱ. 영향받는 클래스를 찾아내는 방법

수정이 일어났을 때 재 테스트해야 하는 클래스 수를 줄이려면, 우선 클래스 레벨에서 특정 클래스의 변경으로 인해 영향을 받는 클래스를 알아내는 것이 중요하다. David C. Kung이 제안한 클래스 방화벽은 이러한 방법을 제안하고 있다.[1] 이를 위해 ORG(Object Relation Graph)를 사용한다. 클래스를 노드로 하고, 클래스들간의 관계를 변의 라벨로 하는 그래프이다.

객체 지향 프로그래밍 언어에서의 계승관계는 보편화와 특정화로 나타난다. 계승관계에 있는 두 클래스는 한 클래스에서 정의된 모든 속성이 자동적으로 서브 클래스에서도 정의되는데, 서브 클래스에서는 이러한 속성들을 선택적으로 받아들이거나 , 다시 정의하거나, 다른 속성들을 추가 정의할 수 있다.

집합관계는 추상화나 클래스 기능으로 제공된다. 조합 객체는 부품 객체들로 이루어지며, 조합 객체를 집합(aggregate) 클래스라 한다.

연합 관계는 두 독립적인 클래스간에 데이터 종속성이나 제어 종속성, 메시지 패싱등의 연관이 있는 경우를 말한다.

객체 관계 그래프는 ORG = (V, L, E)로 나타내는데, V는 클래스를, L은 변의 라벨{I, Ag, As}를, E는 {EIU EAGU EAS}를 나타낸다. 클래스 방화벽을 계산하려면 우선 종속 관계 R을 다음과 같이 정의한다.

$$R=\{\langle C2, C1\rangle \mid C1, C2 \in V \wedge (\exists l) \ (l \in L \wedge \langle C1, C2, l\rangle \in E)\}$$

$\langle C2, C1\rangle \in R$이면 C1이 C2로부터 계승된 클래스이거나, C1이 C2의 자료를 사용하거나 연합관계에 있음을 말한다. 이러한 모든 경우에 C1이 C2에 종속적이라고 하는데 이는 어떤 방식으로든 C2에 변경이 생겼을 때 C1의 행위에 영향을 미침을 말한다. 연속적으로 C2의 변경으로 인해 영향받은 C1에 종속적인 클래스들이 영향을 받게 된다. 클래스 C의 변경으로 인해 영향받는 모든 클래스를 CCFW(computed calss firewall)이라 하여 다음과 같이 정의하였다.

$$CCFW(C) = \{ Cj \mid \langle C, Cj\rangle \in R^* \}$$

CCFW에서 R^*는 transitice closure이다. 즉, $\langle Ci, Cj\rangle$ R이고 $\langle Cj, Ck\rangle$ R^* 이면 $\langle Ci, Ck\rangle \in R^*$이다.

그러나 이렇게 영향받는 클래스만을 찾아낸다고 하더라도 여전히 재 테스트의 문제는 커다란 부담으로 남게 된다. 이 논문에서는 이렇게 찾아낸 재 테스트 되어야할

클래스 중에서도 테스트 범위를 줄이기 위하여 영향받는 구성원 함수만을 찾아내고, 변경과 관계없는 테스트를 수정 테스트에서 제외시켜 테스트비용을 줄일 수 있는 방법을 찾아보고자 한다.

Ⅲ. 서브클래스의 테스트에서 슈퍼클래스의 테스트 정보를 사용하는 방법

슈퍼클래스에서 아무리 완벽하게 테스트된 구성원 함수라 할지라도 서브클래스에 계승되면, 다른 복잡한 관계들로 인해 재 테스트되어야 한다.[2] 여기서는 슈퍼 클래스의 테스트 정보를 사용하여 서브 클래스를 점진적으로 테스트하는 계층구조의 점진적 테스트 방법을 살펴보고 이를 구성원 함수 수준의 테스트에도 사용하고자 한다. 이는 서브 클래스가 속성을 계승할 때 테스트 기록도 같이 계승해서 새로 추가된 속성, 변경된 속성, 변경의 영향받는 속성들을 가려내어 테스트함으로써 테스트 수행 시간들을 절약할 수 있는 방법이다.[3]

서브클래스는 슈퍼클래스의 속성을 계승받고 이와 다른 속성을 정의하거나 변경한다. 슈퍼 클래스를 P, 다른 속성을 정의하거나 변경하는 수정자를 M, 이로 인해 나타나는 결과클래스를 R이라고 하면, 다음과 같은 식이 성립한다.

$$R = P \oplus M$$

수정자에 포함되는 속성들은 다음과 같이 나눌 수 있다.

새로 추가된 속성은 속성 A가 M에는 정의되어 있으나 P에는 있지 않을 때와 A가 M과 P의 구성원 함수이나 서로 파라미터 리스트가 다를 때, A는 P가 아닌 M의 정의로 대응된다. 반복되는 속성은 A가 P에는 정의되어 있으나 M에는 없는 경우이며, 재정의 속성은 A가 M과 P 모두에 정의되어 있고 매개변수 리스트가 같을 때, P의 정의는 사용되지 않고 M의 정의만 사용된다.

구성원 함수들은 서브 클래스에 계승되어 재사용 될 것이므로 스펙 기준 테스트와 프로그램 기준 테스트를 사용한다. 테스트 기록은 다음과 같이 나타낸다.

{mi, (TSi, test?), (TPi, test?)}

mi는 구성원 함수를 TSi는 스펙 기준 테스트 케이스를 TPi는 프로그램 기준 테스트케이스를 나타내며 test?는 테스트케이스가 재 수행되어야 하는 지를 나타낸다.

클래스 내부테스트는 각 구성원 함수나 자료를 노드로 하고, 이들간의 메시지를 변으로 하는 클래스 그래프를 사용하며, 테스트 기록은 {mi, (TISi, test?), (TIPi, test?)}로 나타내어진다.

서브 클래스 테스트는 슈퍼 클래스 P의 기록, HISTORY(P)와 P의 클래스 그래프 G(P)와 수정자 M을 입력으로 받아 서브클래스 R의 기록을 출력으로 내며 테스트 알고리즘은 수정자 M의 개개의 속성 A에 따라 다음과 같이 필요한 테스트를 결정한다.

새로 추가된 구성원 함수는 P에 정의되어 있지 않으므로 완전히 테스트하여야 한다. 또한, 새 구성원 함수도 기존의 구성원 함수에 메시지를 보내거나 자료를 사용할 수 있으므로 A를 G(R)에 통합시키고 테스트케이스를 작성해서 'test?'를 'Y'로 하여 HISTORY(R)에 추가한다. A 자료 속성인 경우는 클래스 내부 테스트에서 G(R)에 추가될 때 A와 상호 작용하는 구성원 함수들을 테스트된다.

반복되는 구성원 함수는 이미 P에서 테스트되었으므로 다시 테스트할 필요가 없다. 다만 클래스 내부 테스트에서 새 구성원 함수와 재정의 함수와의 상호작용만 테스트하면 되나 이는 구성원들이 통합될 때 수행되었으므로 다시 테스트할 필요가 없다. 따라서 A가 클래스내의 다른 구성원 함수들과 같은 자료를 사용하는 경우만 테스트하면 된다.

재정의 속성의 경우는 모두 다시 테스트해야 한다. 클래스 내부 테스트의 경우도 마찬가지이다. 따라서 'test?' 필드를 'Y'로 하여 HISTORY(R)을 변경한다. A가 자료 구성원인 경우는 A를 사용하는 구성원 함수들의 통합 테스트 시에 테스트되므로 여기서는 따로 테스트할 필요 없다.

이 점진적 테스트 방법을 앞으로 제시하려는 모델의 계승관계에 적용하여 서브 클래스의 속성을 여기서 제시된 것과 같이 여러 가지 속성으로 나누고 속성간의 상호작용을 점검해서 재 테스트가 필요한 상호 작용들을 찾아내는데 사용하려 한다.

Ⅳ. 계승관계에서의 변경에 영향받는 구성원 함수 분석

변경된 속성이 서브 클래스에 계승될 때 서브 클래스에서 재정의 되면, 슈퍼 클래스에 발생된 변경이 재 정의된 만큼 서브 클래스에는 영향을 미치지 않는다.[3] 따라서 서브 클래스로 변경이 계승되면서 재정의 된 부분만큼 변경의 영향이 줄어든다고 할 수 있겠다.

이 논문에서는 변경이 일어난 클래스를 슈퍼 클래스라고 하고, 바로 차 하위 클래스를 서브 클래스라 칭하고 각 슈퍼 클래스와 서브 클래스의 수정 테스트 시 필요한 테스트를 알아보기로 한다.

1 슈퍼 클래스 테스트

슈퍼 클래스의 변경은 자료 구성원이나 구성원 함수에 일어날 수 있다. 슈퍼 클래스의 테스트는 변경된 구성원에 대한 구성원 개별 테스트와 변경된 구성원의 통합 테스트로 이루어지며 새로 작성된 구성원 함수를 테스트하는 방법과 동일하다.

2. 서브 클래스 테스트

2..1 구성원 개별 테스트

서브 클래스의 테스트는 각 구성원들의 구성원 테스트와 각 구성원간의 통합 테스트로 나누어진다.

변경은 계승되는 속성과 재정의 속성에서만 일어나게 된다. 왜냐하면, 새로운 속성은 슈퍼 클래스에는 없는 속성을 서브 클래스에서 새로이 정의한 것이어서 슈퍼클래스에 일어난 변경과는 관계가 없기 때문이다. 또한 재정의 된 속성에 발생한 변경은 서브 클래스에서 재정의 되기 때문에 변경이 일어난 곳이나 일어나지 않는 곳이나 서브 클래스에서는 차이가 없다. 따라서 수정 테스트 시에 서브 클래스를 규정 짓는 속성은 새로운 속성, 재정의 속성, 변경이 일어난 계승되는 속성, 변경 없이 계승되는 속성으로 나눌 수 있다.

새로운 속성의 구성원 개별 테스트는 수정 전 테스트의 서브 클래스 테스트에서 수행되어졌기 때문에 여기서는 다시 테스트할 필요가 없다. 재정의 속성의 구성원 개별 테스트도 수정 전 테스트의 서브 클래스 테스트에서 수행되어 졌고, 변경 없이 계승되는 속성의 경우도 마찬가지이므로 여기서는 다시 테스트할 필요가 없다. 변경된 반복되는 속성의 경우는 변경 후 수정 테스트의 베이스 클래스 테스트 시에 수행되어 졌으므로 재 테스트할 필요가 없다. 따라서 서브 클래스의 각 구성원들의 구성원 개별 테스트는 수정 전 테스트와 수정 테스트의 슈퍼 클래스 테스트에서 수행되었기 때문에 모두 재 수행할 필요가 없다. 따라서 서브 클래스 테스트 시에 구성원 개별함수는 속성에 관계없이 모두 재 테스트 할 필요가 없다. 이를 속성별로 정의하면 다음과 같다.

표 1. 구성 개별 테스트 필요 여부

	구성원 개별테스트
새로운 속성	필요 없음
재정의 속성	필요 없음
변경된 반복속성	필요 없음
변경없는 반복속성	필요 없음

위의 표에서 보았듯이 새로운 속성과 재정의 속성에 대한 테스트도 수행하지 않아도 되므로 필요한 테스트의 수를 줄일 수 있다.

2..2 구성원 통합 테스트

구성원 통합 테스트는 각 구성원 함수간의 상호 작용이므로 위에서 정의한 속성 특성별로 필요한 테스트와 필요치 않은 테스트를 살펴보기로 한다.

새로운 속성과 재정의 속성간의 상호 작용에 관한 통합 테스트 케이스는 변경 전 수정 전 테스트의 서브 클래스 테스트 시에 수행되었으므로 재 수행할 필요가 없다. 새로운 속성과 새로운 속성간의 상호작용에 관한 통합 테스트 케이스도 수정 전 서브 클래스 테스트 시에 수행되었으므로 재 수행할 필요가 없다. 변경된 계승된 속성과의 상호 작용은 테스트가 필요하나 변경된 계승된 속성과의 상호작용을 점검할 때로 미루기로 한다.

재정의 속성과 재정의 속성간의 상호 작용에 관한 통합 테스트 케이스는 수정 전 테스트의 서브 클래스 테스트 시에 수행되었으므로 재 수행할 필요가 없다. 새로운 속성과의 상호 작용에 관한 통합 테스트 케이스도 수정 전 테스트의 서브 클래스 테스트 시에 수행되었고, 변경

없는 계승되는 속성과의 상호 작용도 수정 전 테스트 시에 수행되었으므로 재 수행할 필요가 없다. 변경된 계승되는 속성과의 상호 작용은 테스트가 필요하나 새로운 속성과 마찬가지로 변경된 계승되는 속성의 상호작용을 점검할 때로 미루기로 한다.

변경이 일어나지 않은 계승되는 속성간의 상호 작용은 모두 이미 수행되었으므로 재 수행할 필요가 없다. 재정의 속성과 새로운 속성과의 상호 작용에 관한 통합 테스트 케이스는 수정 전 테스트의 서브 클래스 테스트 시에 수행되었고, 변경 없는 계승되는 속성의 상호 작용도 수정 전 슈퍼 클래스 테스트 시에 수행되었다. 변경된 계승되는 속성과의 상호 작용은 변경이 일어난 슈퍼 클래스의 수정 테스트 시에 수행하였기 때문에 재 수행할 필요가 없다.

변경된 계승되는 속성간의 상호 작용은 한정된 테스트가 필요하다. 재정의 속성과의 상호 작용에 관한 통합 테스트는 테스트 케이스를 새로이 작성하여 테스트를 수행하여야 한다. 이는 재정의 속성의 상호 작용을 점검할 때 변경된 반복되는 속성 점검 시로 미루어 두었던 테스트와 일치한다. 새로운 속성과의 상호 작용도 처음 테스트되는 것이므로 여기서 테스트되어야 한다. 이것도 새로운 속성의 상호 작용을 점검할 때 변경이 일어난 계승되는 속성 점검 시로 미루어 두었다. 그러나 변경 없이 계승되는 속성과의 상호 작용과 변경이 일어난 계승되는 속성과의 상호 작용은 수정 테스트 시 슈퍼 클래스를 테스트 할 때 테스트되었으므로 여기서는 재 수행할 필요가 없다.

위에서 살펴본 바를 속성별로 표로 표시하면 다음과 같다.

표 2. 구성원간의 상호 작용 테스트 필요 여부

	신규 속성	재정의 속성	계승되는 속성(변경부분)	계승되는 속성(비변경 부분)
신규 속성	X	X	O	X
재정의 속성	X	X	O	X
계승되는 속성(변경부분)	O	O	X	X
계승되는 속성(비변경 부분)	X	X	X	X

이 표는 방향성을 고려하지 않았기 때문에 대각선 대칭으로 나타나며, 우리가 살펴본 바대로 새로운 속성과 재정의 속성에서 변경이 일어난 계승되는 속성 테스트 시로 미루었던 것처럼, 변경이 일어난 계승되는 속성과 대칭 되어 나타난다. 따라서 서브 클래스의 클래스 내부 테스트에서는 변경이 일어난 계승되는 속성의 재정의 속성과 새로운 속성과의 상호 작용에 관한 테스트만을 수행하면 됨을 알 수 있다. 이를 점진적 테스트 방법과 비교하면 다음과 같다.

표 3. 점진적 테스트 방법과 구성원 함수별 테스트 방법의 비교

	점진적 테스트	구성원 함수별 구분 방법
신규 속성	전체 테스트	테스트 불필요
재정의 속성	전체 또는 일부 테스트	테스트 불필요
계승되는 속성(변경 부분)	부분 테스트 필요	부분 테스트 필요
계승되는 속성(비변경 부분)	부분 테스트 필요	테스트 불필요

여기에 제시된 방법을 사용하면, 점진적 테스트보다 수정 전 테스트에서 수행되었던 부분을 추가로 가려내어 테스트 필요 여부를 결정 짓기 때문에 필요한 테스트가 줄어듦을 알 수 있다.

슈퍼 클래스에서의 변경이 재정의 되는 부분에서만 일어나면, 변경이 일어난 계승되는 속성이 없으므로, 위의 표에서 변경이 일어난 계승되는 속성 행과 열이 없어지는 경우이므로 이 때는 서브 클래스의 통합 테스트가 전혀 필요 없음을 알 수 있다.

따라서 서브 클래스가 연속되는 경우는 슈퍼 클래스에서 변경이 일어난 속성이 모두 재 정의되면, 그 후에 연속된 서브 클래스들은 재 테스트가 필요 없음을 알 수 있다.

이를 알고리즘으로 표현하면 다음과 같다.

```
For each changed attribute A of
                        super class
  IF A does not exist in modifier of
                        subclass
     // for recursive-changed attribute
  THEN
     interaction test of A with new
                        attribute
     interaction test of A with redefined
                        attribute
  END IF
END
```

V. 결론 및 향후 연구과제

이 논문에서는 기 개발 된 프로그램을 스펙은 변경하지 않고 구현 내용만을 변경하였을 때, 그 변경이 다른 클래스나 구성원 함수에 미치는 영향을 분석하여, 변경 시 재 테스트 되어야하는 범위를 최소화하는 방안을 구성원 함수레벨에서 제안하였다.

수정테스트에서 변경의 영향이 미치는 최소한의 범위를 찾아내는 것은 무척 중요하다. 왜냐하면 이것은 비용과 테스트에 소요되는 시간의 문제와 직결되기 때문이다. 계승관계에서 본 바와 같이 이 논문에서 제시된 방법을 사용하면, 서브 클래스의 구성원 함수 단위 테스트가 필요 없고, 구성원 통합테스트의 경우에도 일부만을 수행하게 되므로 테스트해야 할 테스트 케이스의 수가 현저히 줄어듦을 알 수 있다.

여기서 제시한 방법은 계승관계에 한하여 연구하였는데 앞으로 이외의 집합관계, 연합 관계등에 변경이 일어났을 때에 대한 분석도 연구되어야 하며, 본 논문의 시작에서 스펙은 변하지 않고 코드만이 변경되었을 때를 가정하였는데, 스펙이 변경된 경우에 대한 연구도 진행되어 수정 테스트의 비용을 줄일 수 있는 방안이 연구되어야 한다.

참고문헌

[1] David C. Kung et al, "Class firewall, test order, and regression testing of object-oriented programs", JOOP, pp51-65, May 1995

[2] D.E.Perry and G.E.Kaiser, "adequate testing and object-oriented programming", JOOP, vol2, pp13-19, January/February 1990

[3] M.J.Harrold, J.D.Mcgregor and K. J. Fitzpatrick, "Incremental testing of object-or class structure", Proceedings of the international conference of Software Engine Washington D.C., 1992

[4] Gail C. Murphy, Paul Towsend and Pok Sze Wong, "Experiences with cluster and Class Testing", Communication of the ACM, vol 37, no. 9, pp39-47, September 1994

[5] Sankar S. and Hayes R., "ADL-An Inteface definition language for specifying and testing software", Proceedings of the Workshop on Interface Definition Language, 4, pp13-21, 1994

[6] Hoffman D.M., " A case study in class testing", Proceedings of CASCON '93. IBM, Toronto, pp472-482, 1993

저 자 소 개

방 정 원

1984 : 서울대학교,계산통계학과(학사)

1997 : 한국과학기술원, 정보 및 통신공학과(석사)

1984 - 1992 : 한국 IBM 제품개발부

1992 - 1995 : 한국 ISIS 제품개발부

1997 - : 청강문화산업대학 컴퓨터소프트웨어과 조교수