韓國 컴퓨터情報學會 論文誌
第10卷 第5號, 2005. 11.

2005-10-5-1-7

메시지 경로를 이용한 클래스 연합관계에서의 수정 테스트 방법

방 정 원*

Retesting Method of Classes in Associated Relation using Message Path

Jung-Won Bang*

요 약

소프트웨어의 재사용이란 소프트웨어 시스템을 구축할 때 모든 것을 새로 만드는 것이 아니라 기존의 소프트웨어를 이용하여 소프트웨어 시스템을 만드는 과정을 말한다. 객체 지향 개념에서 동적 바인딩, 다형 현상 등은 수행 가능한 경로의 숫자를 현저히 증가시켜 수행 경로를 찾기 위한 소스 코드의 정적 분석은 거의 도움이 되지 않는다. 이러한 개념들은 수정 테스트에도 새로운 문제를 던져 주고 있다. 전통적인 테스트 방법은 상속 관계, 메시지 보냄, 실례 관계와 같은 복잡한 관계들에 대하여는 다루지 않으므로 객체 지향 프로그램의 수정 테스트에 그대로 적용하기 어렵다. 이 논문에서는 새로 대두된 클래스 관계 중에서도 특히 메시지 보냄 관계에 있는 클래스 연합 관계에서, 특정 메서드를 변경함으로서 영향 받는 메서드만을 찾아내어 수정 테스트 되어야 할 메서드 및 클래스를 줄이는 방법에 대하여 연구하였다.

Abstract

Reuse of software is the process of developing software system which does not develop all modules newly but construct system by using existing software modules. Object oriented concept introduces dynamic binding and polymorphism, which increases number of executable paths. In this case static analysis of source code is not useful to find executable path. These concepts occur new problems in retesting. Traditional testing method can not cover complex relations such as inheritance, message passing and instance of and can not apply on object oriented retesting. we propose the method to reduce the number of methods and classes to be retested in associated class relation.

▸ Keyword : 수정테스트(retesting), 메시지보냄(message passing), 클래스 연합 관계(associated class relation)

• 제1저자 : 방정원
• 접수일 : 2005.10.10, 심사완료일 : 2005.11.09
* 청강문화산업대학 컴퓨터소프트웨어과 교수

Ⅰ. 서 론

소프트웨어의 재사용이란 소프트웨어 시스템을 구축할 때 모든 것을 새로 만드는 것이 아니라, 기존의 소프트웨어를 이용하여 소프트웨어 시스템을 만드는 과정을 말한다. 객체 지향 개념은 클래스, 상속, 추상화, 동적 바인딩, 다형성 등의 새로운 개념을 소개하였고, 이로 인해 클래스와 그들의 속성 간에는 복잡한 관계가 형성 되었다. 따라서 각 컴포넌트들 간에는 복잡한 관계가 형성되었고, 이로 인해 소프트웨어의 오류 발생 기회는 더욱 증가 되었다. 예를 들어 상속 관계에서의 하위 레벨은 상속된 속성들의 새로운 내용이 되므로 재 테스트되어야 한다. 왜냐하면 상위 레벨에서의 올바른 행위가 하위 레벨에서도 올바른 행위를 보장하지 않기 때문이다. 또한 동적 바인딩과 함께 다형 현상 등은 수행 가능한 경로의 숫자를 현저히 증가 시켰다.

따라서 수행 경로를 찾기 위한 소스 코드의 정적 분석은 거의 도움이 되지 않는다. 이러한 개념들은 수정 테스트에도 새로운 문제들을 던져 주고 있다. 첫 번째로 전통적인 방법은 상속관계, 메시지 보냄, 실례 관계와 같은 복잡한 관계들에 대하여는 다루고 있지 않다. 두 번째로 전통적인 방법들은 제어 흐름 모델에 기초를 두고 있으나 클래스 객체는 다양하게 변화하는 상태 의존적인 행위를 갖고 있다. 세 번째로는 전통적인 접근 방법은 불려지는 모듈을 시뮬레이트하기 위해 테스트 스텁을 사용하나 객체 지향 프로그램에서는 많은 관련된 클래스들을 이해해야 하고 그것들의 구성원 함수들을 이해해야하고 어떻게 이러한 구성원 함수들이 불리는 지를 이해해야 하기 때문에 테스트 스텁을 사용하는 것이 어렵고 비용이 많이 든다. 따라서 전통적인 방법들을 객체 지향 프로그램의 수정 테스트에 그대로 적용하기는 어렵다.

수정 테스트의 목적은 수정된 프로그램이 여전히 요구사항을 충족한다는 것을 보여주는데 크게 다음과 같이 나누어 볼 수 있다[1].

1) 어떤 컴포넌트의 변경으로 인해 영향 받는 컴포넌트들을 어떻게 찾아낼 것인가?
2) 영향 받는 컴포넌트들을 어떤 순서로 테스트할 것인가?
3) 이 컴포넌트들을 테스트하는데 사용되는 점검 항목은 무엇인가?
4) 기존의 테스트 케이스를 어떻게 선택해서 재사용할 것인가?

특히 수정 테스트에 드는 시간과 노력을 절약하기 위해서는 수정에 의해 영향 받는 부분들만을 찾아내는 것이 중요하다.

이 논문에서는 특히 클래스가 연합 관계에 있는 경우 특정 메서드를 변경함으로서 영향 받는 메서드만을 찾아내어 수정 테스트 되어야 할 메서드 및 클래스를 줄이는 방법에 대하여 다루고자 한다.

Ⅱ. 메서드 경로 추적에 의한 통합 테스트 방법

객체 지향 소프트웨어의 사건 지향적인 성향에 초점을 맞추어 Paul C. Jorgensen 과 Carl Erickson은 클래스, 클러스터로 대별되는 테스트와는 다른 통합 테스트 방법을 제안 하였다[2]. 사건 지향적인 성향에 맞추어보면 테스트를 다음과 같은 단계로 나누어 볼 수 있다.

메서드
메시지 종료(message quiescence)
사건종료(event quiescence)
쓰레드 테스트

쓰레드 상호 작용 테스트

여기서 메서드는 단위 테스트에, 메시지 종료와 사건 종료는 통합 테스트에, 쓰레드 테스트와 쓰레드 상호 작용 테스트는 전통적인 시스템 테스트에 해당된다.

우선 단위 테스트에 해당되는 메서드 테스트는 전통적인 방법들을 그대로 적용할 수 있다.

이렇게 테스트된 메서드를 노드로 하고 메서드들 간에 메시지를 변으로 하면 (그림 1)과 같은 방향성 그래프 형태의 객체 네트워크를 구성하게 된다.

그러나 각 객체는 독립된 각각의 그래프로 나타내지지 않는다.

이 객체 네트워크에는 메시지에 의해 연결되어 수행되는 일련의 메서드들을 볼 수 있는데 이것을 MM(Method/Message) 경로로 정의한다. 다음의 그림에서 객체 1의 메서드 1에서 시작해서 메서드 3에서 끝나는 MM 경로 1과 객체 1의 메서드 2에서 시작해서 객체 2의 메서드 2를 거쳐 객체 3의 메서드 1에서 끝나는 MM 경로 2, MM 경로 2의 객체 2의 메서드 2에서 파생된 객체 3의 메서드 1에서 끝나는 MM 경로 3 등이 나타나 있다.

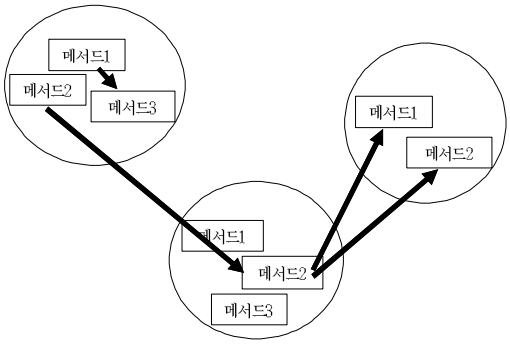

그림 1. 객체 네트워크의 예
Fig. 1. Example of object network

즉 MM 경로는 메서드에서 시작해서 더 이상 자신의 메시지를 내지 않는 메서드에서 끝나게 되는데 이것을 메시지 종료점이라 한다. 이러한 MM 경로는 서로 연결된 메서드들의 메시지들로 이루어지며, 위의 그림에서 본 것과 같은 다른 MM 경로와 상호 교차하기도 하고 또 다른 MM경로를 파생시키기도 한다.

이러한 MM 경로는 객체 지향 소프트웨어의 통합 테스트를 할 수 있는 방법을 제공해 준다. 이러한 MM 경로는 시스템 수준의 입력을 통해서 일어나게 되고, MM 경로의 연속은 시스템에 어떤 반응을 냄으로써 끝나게 된다. 이에 시스템 입력으로 시작되어 MM 경로를 거쳐 시스템에 대한 반응으로 끝나는 또 하나의 개념이 대두된다.

이것을 ASF(Atomic System Function)으로 정의하고 있는데, 이것은 입력이 일어나는 사건과 그것에 의해 발생되는 여러 개의 MM 경로와 MM 경로가 종료될 때 일어나는 사건으로 이루어져 있으며 출력 사건으로는 공 값을 포함해 여러 값을 가질 수 있다[3].

위의 객체 네트워크에서 살펴보면, 사건 A의 시작점에서 시작해서 MM 경로 1을 거쳐 사건 A의 종료 점에서 끝나는 ASF A와 사건 B의 시작점에서 시작해서 사건 B의 종료 점에서 끝나는 ASF B를 볼 수 있다. 이것은 시스템에 대한 자극과 반응으로도 볼 수 있는데, 특히 이러한 방법은 사건적인 성격을 갖는 시스템에 적당하다. 예를 들어, 잘못 작성된 GUI 소프트웨어의 경우 어떠한 입력에 대하여는 아무런 반응을 난타내지 않는 경우가 있다. 이런 경우 ASF의 출력 사건은 공 값이 된다.

이러한 ASF는 시스템 레벨에서 볼 수 있는 기본적인 함수이므로 통합 테스트와 시스템 테스트가 만나는 점이라고도 볼 수 있다.

한 개의 객체는 여러 개의 쓰레드를 포함하고 쓰레드는 여러 개의 객체를 포함할 수 있다. 비슷하게 객체는 여러 개의 ASF를 포함하고, 한 개의 ASF는 여러 개의 객체를 포함할 수 있다. 위의 두 관계를 통해 객체들이 통합될 수 있고, 더 나아가서는 통합이 구조보다는 행위에 중점을 두어 이루어진다는 것을 알 수 있다[5].

구조적 테스트의 단점 중의 하나로 수행 불가능한 경로를 들 수 있는데, 위의 구성 요소들은 구조적으로는 가능하나 수행되지 않는 경로들에 관한 문제들은 피할 수 있음을 보여 준다[6].

계승관계의 경우 스펙은 변경되지 않고 구현 내용만을 변경한 경우, 그 변경이 다른 클래스나 구성원 함수에 미치는 영향은 구성원 함수 레벨 단위로 분석하여 재 테스트 되어야 하는 범위를 줄일 수 있다[7].

연합 관계의 경우 두 개의 클래스의 모든 구성원들이 서로 작용하는 것이 아니고 특정 구성원만이 관련되므로 여기서 사용된 MM 경로의 개념을 적용시켜 영향 받는 메서드만을 찾아내고자 한다.

III. 연합관계에서 영향 받는 메서드

연합 관계는 서로 독립적인 두 클래스 간에 다음과 같은 3가지 경우에 클래스 C1이 클래스 C2에 연합 관계에 있다고 한다.

- 자료 종속: 클래스 C1이 클래스 C2의 데이터를 사용할 때
- 메시지 보냄: 클래스 C2의 구성원 함수가 클래스 C1의 구성원 함수에 의해 불릴 때
- 객체 매개변수 보냄 : 클래스 C2의 객체가 클래스 C1의 한 구성원 함수의 매개변수로 사용될 때

3.1 자료종속 관계

자료 종속은 클래스 C1의 메서드가 C2의 메서드를 사용할 때 발생한다. 이것을 그래프로 나타내면 다음과 같다.

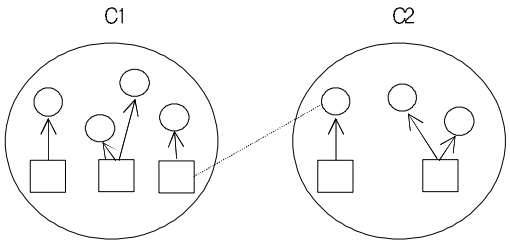

그림 2. 자료 종속의 객체 그래프
Fig. 2.Object graph of data depedency

여기서 원은 멤버 변수를 네모는 메서드를 나타내고, 화살표는 메서드가 멤버 변수를 사용함을 표시한다.

클래스를 이렇게 나타내 보면 클래스는 서로 연결되지 않은 여러 개의 서브 그래프로 이루어진다. 위의 그림에서 클래스 C1, C2의 그래프 중 서로 연결 되지 않은 서브 그래프를 좌로부터 순서대로 C1.1, C1.2, C1.3, C2.1, C2.2 로 명명하기로 한다. 이러한 경우 크게, C2.2에 변경이 발생했을 때 즉, 변경이 발생한 C2의 서브 그래프와 C2를 사용하는 C1의 서브그래프 C1.3이 서로 연결되지 않았을 때와 C1이 사용하는 C2의 서브그래프에 변경이 발생했을 때, 즉 C2.1에 변경이 발생했을 때로 나눌 수 있다.

C1이 사용하는 자료가 속한 C2의 서브 그래프 C2.1에 변경이 발생했을 때는 C2의 변경으로 인해 C1이 영향을 받으므로 C1의 재 테스트가 필요하나, C1이 사용하는 자료가 속한 C2의 서브 그래프와 C2의 변경이 발생한 서브 그래프가 서로 연결되지 않는다면, 즉 C2.2에 변경이 발생했다면, C2의 변경이 C1에 영향을 미치지 않으므로 C2가 변경되었다 할지라도 C1은 재 테스트할 필요가 없다.

1개 이상의 연합관계가 지속될 때도 같이 적용될 수 있는데, 이것을 그래프로 나타내면 다음과 같다.

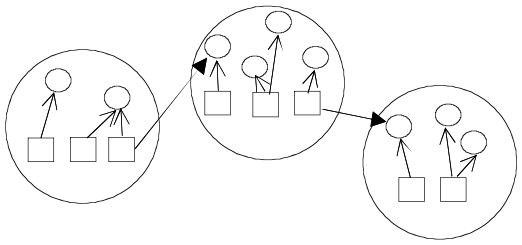

그림 3. 1개 이상의 연합 관계가 연결될 때
Fig. 3. connected associated relation

그래프에서 C1은 2개의 서브 그래프로 이루어져 있고 이를 차례대로 C1.1, C1.2, C1.3 이라 하자. 클래스 C2는 3개의 서브 그래프로 이루어져 있으며 이를 각각 C2.1, C2.2, C2.3이라 하자. 클래스 C3은 2개의 서브 그래프로 이루어져 있으며 이를 C3.1, C3.2라 하자. 이 그래프의 경우 C1은 C2에 종속되어 있고, C2는 C3에 종속되어 있다.

C3.2에 변경이 일어난 경우 변경이 일어난 C3의 서브 그래프와 C2가 사용하는 자료가 속한 서브 그래프가 서로 연결되어 있지 않으므로 C3이 변경되어도 C2에는 영향을 미치지 않아 C2를 재 테스트할 필요가 없다.

C3.1에 변경이 일어난 경우 C2가 사용하는 자료가 속한 서브 그래프에 변경이 일어났으므로 이 변경은 C2에 영향을 미쳐 C2.3을 재 테스트해야 한다. 그러나 다음과 같은 경우는 C3의 어느 부분에서 변경이 일어났는가에 관계없이 C1은 재 테스트 하지 않아도 된다.

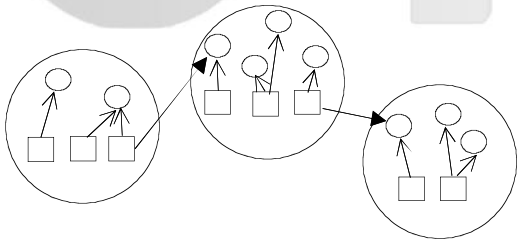

그림 4. 1개 이상의 연합 관계가 지속될 때
Fig. 4. Seperated associated relation

그러나 클래스 결합도가 높아 클래스 그래프가 각각의 서로 연결되지 않은 서브 그래프로 나타나지 않고 한 개의 그래프로 나타나는 경우를 고려하여 보자.

이것을 그래프로 나타내 보면 다음과 같다.

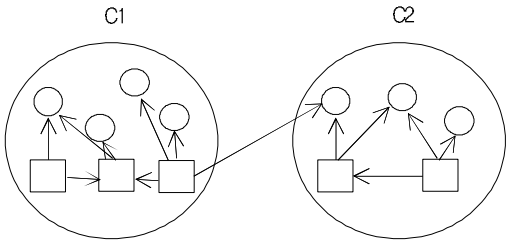

그림 5. 클래스 결합도가 높은 경우의 객체 그래프
Fig. 5. Object graph of tightly coupled relation

위의 클래스 관계에서 나타날 수 있는 경우는 다음과 같다.

a. C1의 구성원이 사용하는 자료가 변경된 경우
b. C1의 구성원이 사용하지 않는 자료가 변경된 경우
c. C1의 구성원이 직접 사용하지 않으나 C1이 사용하는 자료를 사용하는 구성원이 사용하는 자료가 변경된 경우
d. C1의 구성원이 사용하는 자료를 사용하는 메서드가 변경된 경우
e. C1의 구성원이 사용하는 자료를 사용하지 않는 메서드가 변경된 경우

위 그래프에서 각 클래스의 멤버 변수를 각각 d11, d12, d13, d14, d21, d22, d23라 명명하자.

a의 경우 C2에서는 m21 메서드의 개별 테스트와 m21 통합 테스트가 필요하다.

b의 경우 C2에서는 m22 메서드의 개별 테스트와 m22 통합 테스트가 필요하다. 그러나 C1이 종속적인 멤버 변수가 영향을 받지 않았기 때문에 C1은 테스트할 필요가 없다.

c의 경우 C2에서는 m21,m22 메서드 개별 테스트와 m21,m22 통합 테스트가 필요하다. C1에서는 d21이 직접 변경되지는 않았으나 d21을 사용하는 구성원 m21이 영향을 받음으로 m21의 인터페이스에도 영향을 미쳐 d21이 영향을 받게 되므로 d21의 인터페이스 즉 m13의 개별 테스트와 m13 통합 테스트가 필요하다.

d의 경우 C2에서는 m21 메서드의 개별 테스트와 m21 통합 테스트가 필요하다. C1에서는 m13의 변경으로 인해 d21이 영향 받았으므로 d21의 인터페이스 테스트 즉 m13-d21과 m13 개별 테스트, m13통합 테스트가 필요하다.

e의 경우는 C2에서는 m22 메서드 개별 테스트와 m22 통합 테스트가 필요하나, d21이 영향을 받지 않았으므로 C1의 테스트가 필요 없다.

3.2 메시지 보냄 관계

메시지 보냄 관계는 클래스의 C1의 메서드가 클래스 C2의 멤버 변수를 사용하는 것이 아니라 메서드를 사용한다는 것이 자료 종속과 다르다. 이것을 그래프로 나타내면 다음과 같다.

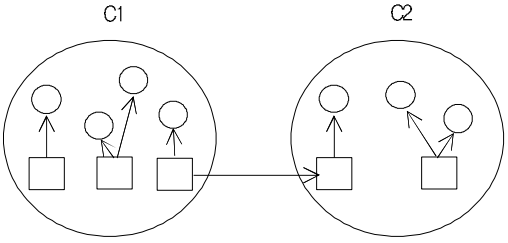

그림 6. 메시지 보냄 관계의 객체 그래프
Fig. 6. Object graph of message passing

따라서 자료 종속에 나타난 모든 그래프의 화살표가 메서드를 향하고 있다는 점만 다르므로 자료 종속에서 사용한 방법을 그대로 사용할 수 있다.

클래스 결합도가 높은 클래스간의 관계에 있어서는 다음과 같이 나눌 수 가 있다.

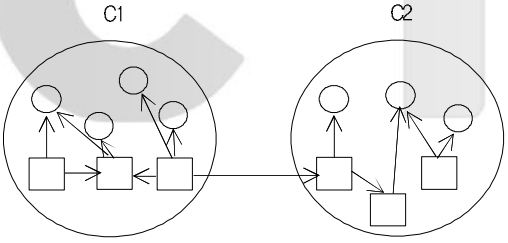

그림 7. 클래스 결합도가 높은 메시지 보냄 관계의 객체 그래프
Fig. 7. Object graph of message passing in tightly coupled classes

a. C1의 구성원이 불러들이는 메서드가 사용하는 자료가 변경된 경우
b. C1의 구성원이 불러들이는 메서드가 사용하지 않는 자료가 변경된 경우
c. C1의 구성원이 불러들이는 메서드가 직접 사용하지 않으나 그 메서드와 같은 통합 케이스에 속하는 자료가 변경된 경우

d. C1의 구성원이 불러들이는 메서드가 변경된 경우
e. C1의 구성원이 직접 불러들이지는 않으나, C1이 부르는 구성원과 같은 통합 테스트 케이스에 속하는 경우
f. C1의 구성원이 직접 불러들이지는 않고, C1이 부르는 구성원과 같은 통합 테스트 케이스에 속하지 않는 구성원이 변경된 경우

a의 경우 C2에서는 m21 메서드의 개별 테스트와 m21 통합 테스트가 필요하다. C1에서는 m21을 사용하는 구성원인 m13 통합 테스트가 필요하다.

b의 경우 C2에서는 m23 메서드의 개별 테스트와 m23 통합 테스트가 필요하다. 그러나 C1이 종속적인 멤버 변수가 영향을 받지 않았기 때문에 C1은 테스트할 필요가 없다.

c의 경우 C2에서는 m22 메서드 개별 테스트와 m22 통합 테스트가 필요하다. C1에서는 m21이 영향을 받음으로 m13 통합 테스트가 필요하다. 이것을 MM 경로를 이용한 객체 네트워크를 그려보면 m13-m21-m22가 같은 경로 안에 있으므로 m22의 변경에 의해 m13까지의 통합테스트가 필요함을 알 수 있다.

d의 경우 C2에서는 m21 메서드의 개별 테스트와 m21 통합 테스트가 필요하다. C1에서는 m13-d21관계와 m13 통합 테스트가 필요하다.

e의 경우는 C2에서는 m22 메서드 개별 테스트와 m22 통합 테스트가 필요하다. C1에서는 c의 경우와 같이 m22와 m13이 같은 통합 테스트 케이스에 속하므로 m21-m13 관계와 m13 통합 테스트가 필요하다.

f의 경우는 C2에서는 m23 메서드의 개별 테스트와 m23 통합 테스트가 필요하다. C1에서는 m23과 m13이 같은 통합 케이스에 속하지 않으므로 변경의 영향을 받지 않아 더 이상의 테스트가 필요 없다.

3.3 객체 매개 변수 보냄 관계

객체 매개 변수 보냄 관계는 클래스 C2의 객체가 클래스 C1의 한 구성원 함수의 매개변수로 사용될 때를 말한다. 이것을 자료 종속이나 메시지 보냄과 같은 방법으로 그래프로 나타내면 다음과 같다.

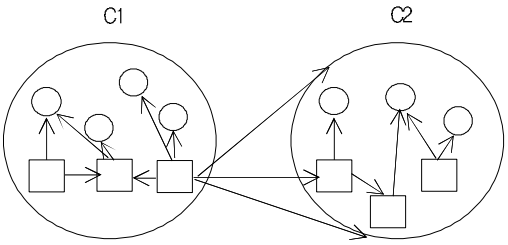

그림 8. 객체매개변수 보냄 관계의 객체 그래프
Fig. 8. Object graph of parameter passing

위의 그래프에서 알 수 있듯이 C2의 어느 부분의 변경도 m13에 영향을 미친다. 따라서 매개변수 보냄 관계에 있는 경우는 모든 변경의 경우에 C1에서는 m13의 개별 메서드 테스트와 m13 통합 테스트가 필요하다.

위의 3가지 경우의 테스트 알고리즘은 다음과 같다.

```
Setup set S {
 If (member dependency)  then
   {
    include the dependent member in S
    include all data in S which the dependent
                                member accesses
    include all member which interact with
                          the dependent member
    }
 end if
 If (data dependency) then
   {
    include dependent data
    include all member mi which access the
                               dependent data in S
    include all data in S which member mi
                               accesses
    }
end of set up set S
}
end if

If (change occurs on data or member in S)
  then
   {
    individual member test of mi which use
        data in C2 or message passing to C2
    interaction test of mi
    }
  else
   no test needed for C1
endif

end
```

Ⅳ. 결론

본 논문에서는 기 개발된 소프트웨어에 수정이 가해진 경우 그 변경이 다른 클래스나 메서드에 미치는 영향을 개체 그래프를 이용하여 분석하고 그 중 연합관계에 있는 객체 관계에서 변경에 의해 재 테스트 되어야 하는 부분에 대하여 메서드 레벨로 제시 하였다.

클래스 간의 관계는 상속 관계, 집합 관계, 연합 관계로 나누어 볼 수 있는데 이 중 연합 관계가 관계성이 가장 약하다고 볼 수 있다. 연합 관계는 관련된 클래스의 모든 메서드나 멤버 변수와 관련된 것이 아니라 특정 멤버 변수나 메서드와 관련 된다.

Paul C. Jorgensen과 Carl Erickson의 통합 테스트 방법에서도 보았듯이 클래스의 구성원 함수를 노드로 하고 그들 간의 메시지를 변으로 하는 객체 네트워크는 오히려 클래스 자체로는 그래프가 형성되지 않는 것을 보여준다.

따라서 대부분의 많은 그래프가 서브 그래프로 나뉘고 변경의 영향은 종속적인 서브 그래프에만 영향을 미치므로 테스트해야 하는 범위가 클래스 전체에서 서브 그래프로 한정되었다. 또한 클래스 레벨에서는 2개 이상의 연합 관계가 계속 연결될지라도 메서드 레벨에서는 서브 그래프가 연결되어 있지 않아 변경이 전혀 영향을 미치지 않는 경우도 제시하였다.

그러나 여기서 제시한 방법은 스펙은 변하지 않고 구현만 변한 경우에 한정 하였으며, 재사용의 초점이 되고 있는 테스트 케이스의 재사용에 관하여는 고려하지 않았다. 따라서 각 개체 관계에 따라 스펙이 변경된 경우의 재사용과 테스트 케이스의 재사용에 대한 연구가 진행 되어야 한다.

참고문헌

[1] Gali C. Murphy, Paul Townsend and Pok Sze Wong, “Experiences with Cluster and Class Testing”, Communications of the ACM, Vol. 37, No. 9, pp. 39-47, September 1994.

[2] Paul C. Jorgensen and Carl Erickson, “Object-oriented Integration Testing”, Communication of the ACM, Vol. 37, No. 9, pp. 30-38, September 1994.

[3] John D. McGregor and Timothy D. Korson, “Integrated Object-oriented testing and development process”, Communication of the ACM, Vol. 37, No. 9, pp. 59-77, September 1994.

[4] 임 근, “이벤트 추상화를 통한 정보관계 표현”, 한국 컴퓨터 정보학회 논문지, Vol. 7 No. 4, pp. 1-7, December 2002.

[5] Sankar, S. and hayes, R., “ADL-An Interface definition language for specifying nd testing software”, Proceedings of the Workshop on Inteface definition language, pp. 13-21, 1994.

[6] David C. Kung et al, “Class firewall, test order and regressing testing of object-oriented programs”, JOOP, pp. 51-65, May 1995.

[7] 방정원, “계승관계에서 구성원 함수 수준의 변경 영향 분석”, 한국 OA학회 논문지, Vol. 7, No 1, pp. 27-32, March 2002.

저 자 소 개

방 정 원

1997 한국과학기술원, 정보 및 통신공학과 석사

1984~1992 한국 IBM 제품개발부

1992~1995 한국 ISIS 제품개발부

1997~현재 청강문화산업대학 컴퓨터소프트웨어과 교수