DOI: 10.3745/KIPSTD.2009.16-D.2.213

XML을 이용한 UML 기반 시험 산출물의 추적

서 광 익[†] · 최 은 만[††]

요 약

추적성 개념은 모델 중심의 개발에서만이 아니라 테스트를 위하여 매우 중요한 요소이다. 어떤 기능을 중심으로 높은 추상수준에서부터 낮은 수준의 프로그램까지 수직적인 추적은 모델로부터 테스트와 디버깅 단계에 이르기까지 시스템을 잘 관리할 수 있게 한다. 또한 테스트 단계에서 발견한 결함에 대한 오류를 추적할 때 발생하는 단계별 추상 수준의 분리를 추적가능성으로 극복하게 한다. 이 논문에서는 XML을 기반으로 모델과 시험사례 그리고 소스코드를 수직적으로 추적하여 더욱 엄격한 테스트가 되는 방법을 제안한다. 실험을 통하여 추적이란 개념이 어떻게 작동하고 오류 부분을 찾아내는지 보이며 구체적인 사례를 이용하여 방법을 소개한다.

키워드 : 시스템 테스트, 추적가능성, UML 테스트, 통합 테스트, 오류 추적가능성

Traceability of UML Based Test Artifacts Using XML

Seo Kwang Ik[†] · Choi Eun Man[††]

ABSTRACT

Traceability has been held as an important factor in testing activities as well as model driven development. Vertical traceability affords us opportunities to improve manageability from models and test cases to code in testing and debugging phase. Traceability also overcomes difficulties by the separation between abstraction levels when we trace errors from models to source code after test. To support a rigorous test this paper proposes XML based traceability which vertically trace from model and test case to source code. This paper explains how the traceability works and finds out error spots Through experiments using a concrete example.

Keywords : System Test, Traceability, UML Test, Integration Test, Error Traceability

1. 서 론

시스템을 효율적으로 구현하기 위해 여러 추상 수준의 모델을 제시하고 이를 바탕으로 점차 구체적인 소스코드를 구현하는 것을 모델기반(Model-driven) 개발 방법이라 한다. 그리고 개발단계에서 생산한 모델을 기준으로 시험 사례를 명세하고 검사하는 방법을 모델기반 시험(Model-based Testing)이라 한다[1, 2].

모델기반 개발은 개발 과정에서 다양한 산출물과 소소 코드가 생성된다. 이 때 산출물의 종류와 양이 너무 많다면 각 산출물 간의 관계를 파악하기 어려울 수 있다. 특히 객체지향 개발에서는 요구사항 분석과 개념 설계 그리고 상세 설계 단계에서 발생하는 산출물이 다양하고, 이를 바탕으로 소소 코드를 개발하기 때문에 개발 단계의 산출물과 코드의 결합력이 약하다[3]. 이러한 약점을 보완하기 위한 방법으로 산출물 추적을 통한 시스템 개선과 같은 연구가 필요하다[4]. 이와 마찬가지로 모델 기반 시험 방법도 개발 단계에서 산출한 모델을 이용하여 시험 사례를 작성하기 때문에 시험 사례 간의 결합력뿐 아니라 시험 사례와 소소 코드간의 결합력이 약할 수밖에 없다. 이를 극복하기 위해서는 개발 단계의 추적뿐 아니라 시험 단계에서 발생하는 산출물의 추적에 관한 방법이 필수적이다.

추적성에 관한 최근 연구로는 추적 링크를 찾아내고 구현하는 것이 있다. Pinheiro와 Ramesh 그리고 Antoniol의 연구는 코드를 포함한 각종 요구명세서 간의 추적 관계를 찾아낼 수 있는 모델이나 도구를 제안했다[5, 6, 7]. 그리고 시각적인 표현을 통해 추적관계를 직관적으로 이해할 수 있는 방법과 도구에 관한 연구가 있다[8, 9, 10]. Zhou 등은 ENVISION을 개발하여 시각적인 표현 사례를 발표했다. 또한 추적관계가 변경됐을 때 이를 유지하거나 복구하는 연구가 있는데 Grechanik 등은 사용사례 다이어그램에서 소소 코드까지의 관계를 구현하고 유지보수하는 방법을 제안했고 Lucia 등은 유지보수 기능을 갖춘 산출물 관리 시스템을 제안하기도 했다[3, 11]. 그리고 추적의 자동화에 대한 연구로는 Huang과 Neumuller 등이

† 준 회 원 : 동국대학교 컴퓨터공학과 박사수료
†† 정 회 원 : 동국대학교 컴퓨터공학과 교수
논문접수 : 2008년 11월 11일
수 정 일 : 1차 2008년 12월 22일, 2차 2009년 1월 12일
심사완료 : 2009년 1월 12일

실례를 통해 추적 자동화를 구현하는 방법을 보였다[12, 13].

본 연구에서는 추적관계를 유지보수하거나 자동으로 찾아내는 방법 보다는 추적 대상을 식별한 후 추적성을 구현하고 시각적으로 표현하는 프로토타입에 대해 기술한다. Pinheiro와 Ramesh, Antoniol와 같은 기존의 연구는 추적의 대상을 대부분 개발단계에서 작성하는 산출물로 간주한다. 하지만 본 연구는 시험단계에서 작성하는 산출물까지 포함한다. 특히 Pinheiro의 연구와 같이 객체지향기반 개발 산출물을 기반으로 하되 UML 기반으로 작성한 산출물을 추적대상으로 한다. 그리고 산출물 표현 및 추적 방법으로는 XML 기반 언어를 사용한다. XML은 문서와 데이터를 공유할 수 있는 표준으로 널리 쓰인다. 실제로 워드프로세서, 스프레드시트, UML 명세 도구, 컴포넌트 명세 도구 등과 같은 다양한 응용프로그램이 산출 결과를 XML 형태로 저장하거나 읽을 수 있도록 지원하고 있다. 따라서 여러 이해관계자들은 XML 기반 명세를 이해하고 사용자 환경에 맞도록 적용할 수 있다. 또한 XML 파생 언어 중 XPoint는 명세 요소 간의 링크를 설정하는 수단을 제공하고 있어 추적 관계를 적용하는데 용이하다[14]. 그리고 XML의 사용 편이성을 제공하고 있는 다양한 기존 연구와 도구가 이미 개발되어 있을 뿐 아니라 XML을 이용한 UML 표현 방법이 OMG의 표준으로 제안되었다.

이에 본 논문에서는 객체지향 관점에서 주로 사용하고 있는 UML 모델 기반 시험 사례들 간의 추적 방법을 XML 기반 파생 언어로 실현한다. 그리고 추적을 통해 오류를 발생시키고 있는 소소 코드의 위치를 확인해 본다.

2. XML을 이용한 추적

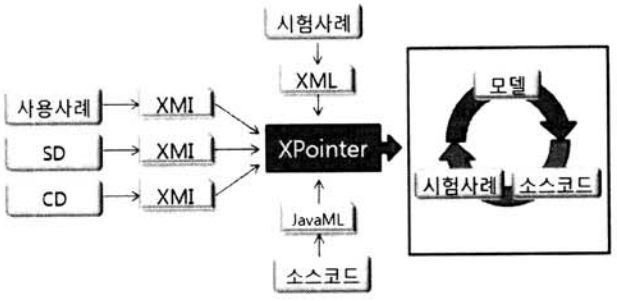
2.1 XML을 이용한 추적 사례

XML을 이용한 추적은 Matetic 등이 XPath를 이용하여 서로 다른 수준의 모델의 추적 방법을 제안했다[15]. 이 방법은 로컬 영역에서 계층별로 흩어져 있는 산출물의 추적을 잘 표현하고 있다. 또한 'function/block/if/condition'과 함께 어떤 기능에 속한 if 조건에 대한 정보를 살펴 볼 수 있는 방법도 있다. 하지만 물리적으로 분산되어있는 산출물과 소스코드 사이의 관련성이 결여되어 있다. Bellos 등은 XLink를 이용하여 생산자부터 고객까지 어떤 경로로 식품이 전달이 되었는지 추적하는 시스템을 제안했다[16]. 유통 과정을 XML 형태로 등록하면 원거리에 있는 관계자들이 인터넷을 통해 확인할 수 있는 방법이다. 이러한 방법은 XML의 장점을 이용하여 정보를 공개하는데 있어 편의성을 제공하고 있지만 추적의 대상과 종류가 다를 뿐 아니라 추적 기능도 단순하다. Alves-Foss도 XLink를 이용하여 모델과 소스코드의 관계를 보이는 방법을 제안했다[17]. 연구 [17]은 객체지향기반 개발에서 발생하는 초기 모델에서부터 소스 코드까지 추적하고 있다. 그러나 소스코드추적 단계에서는 코드 내에 있는 요소들의 추적보다는 클래스 수준에서 관련 있는 상위 수준의 산출물을 언급하는 정도이다. 또한 XLink를 XMI 명세로 다시 정의한 다음 XSL 스타일시트로 변환 보여주므로 추적관계를 파악하기 위해서는 테이블에 표현된 제목과 셀의 내용을 해석해야 한다. 그러나 Lumb는 XPointer를 이용하여 지구역학에 사용되는 데이터와 주석문의 링크를 제공하고 직관적으로 추적 관계를 파악할 수 있는 사용자 인터페이스를 제안했다[14].

본 논문에서는 연구 [17]과 같이 객체지향 기반의 산출물을 추적하면서 연구 [14]와 같이 물리적으로 서로 다른 위치에 저장되어 있는 산출물을 추적할 수 있도록 한다. 하지만 추적의 범위는 개발단계의 소스코드까지가 아니라 시험 단계의 시험사례가지를 포함한다. 그리고 이러한 추적 관계를 연구 [15]과 같이 직관적으로 파악할 수 있는 사용자 인터페이스를 제안한다. 그리고 이를 통해 시험단계에서 발견한 오류의 위치를 찾아내기 위해 소스코드까지 도달하는 과정을 보인다.

2.2 추적을 위한 XML 기반 언어

본 연구에서는 수직적 시험에서 제안했던 관점과 같이 사용사례로부터 기능 시험사례를, 순서도에서 통합 시험사례를, 클래스 다이어그램에서 단위 시험사례를 작성하는 것을 가정한다[18]. 이러한 산출물들은 (그림 1)과 같이 데이터 모델들을 서로 교환하기 위한 개발 도구와 저장소간의 표준으로 쓰이는 XML 기반 언어로 변환되어야 한다. 사용사례나 순서도 그리고 클래스 다이어그램은 Together나 Rational Rose와 같은 자동화 도구에 의해 XMI(XML Metadata Interchange)로 변환이 가능하다. 그리고 표 형태의 시험사례는 XML로 표현한다. 표의 형식이나 내용 또한 XML로 자동변환 할 수 있는 MS 오피스 제품과 같은 상용도구가 있다. XML 기반 Java 소스코드의 표현은 JavaML을 이용한다. JavaML은 자바 코드를 XML로 기술한 마크업 표현으로 Badros의 방법[19]과 Evan Mamas의 방법[20] 두 가지가 있다. 이 중 JavaML 표현을 하는데 있어 공개소프트웨어로 사용에 제한이 없는 Badros의 방법을 채택했다. 하지만 XMI나 JavaML은 모델과 소스코드 간의 추적 관계를 나타낼 수 없다. 따라서 추적성을 표현하는 방법으로는 XLink 또는 XPointer를 이용한다. XLink은 XML로 표현한 문서들의 링크를 작성하는데 사용한다. HTML과는 다르게 여러 개의 문서들이 동시에 연관되는 링크들 및 연관된 리소스를 다양한 방향으로 탐색할 수 있는 링크가 가능하다. 그러나 링크의 대상은 XML 문서 단위로만 가능하고 문서 내에 있는 단어와 같은 요소들의 접근에는 한계가 있다. 따라서 소스코드 내부의 문장이나 특정 변수와 같은 수준까지는 추적할 수 없다. 하지만 XPointer는 XLink을 확장하여 문서를 추적하면서 동시에 HTML의 링커와 같이 XML 문서 안에 선언된 요소들까지도 추적할 수 있도록 지원한다. 즉 XML 문서에 들어 있는 문서 조각을 가리킬 수 있는 기능을 제공한다. 예를 들면 하위 폴더에 <chap1> ... xxx ... </chap1>와 같은 요소를 포함하고 있는 book.xml이 있고 문서가 있다고 하자. XPointer를 정의하는 문서에서는 <subset

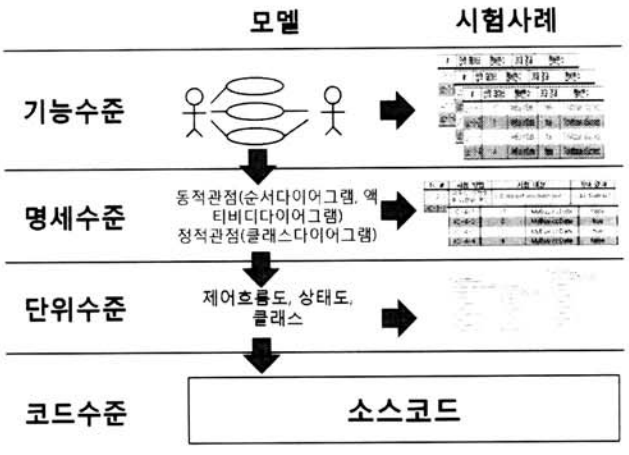


(그림 1) XML을 이용한 추적 방법

xlink:type="simple" xlink:href="book.xml#XPointer(//chap1) ... >와 같이 정의하여 xlink를 이용하여 book.xml를 접근하고, XPointer를 이용하여 book.xml 내에 있는 <chap1>.. </chap1>을 접근하게 된다. 따라서 (그림 1)과 같이 XML을 이용하여 UML과 시험사례의 엘리먼트를 정의한 후 소스코드의 변수나 조건문 등과 같은 구문들을 접근하기 위해 XPointer를 사용한다. 그리고 궁극적으로는 XPointer를 이용하여 소스코드를 변환한 JavaML 내의 변수나 조건문들을 추적하게 되는데, (그림 11, 12)와 같이 JavaML는 소소코드의 위치 정보를 모두 포함하여 표현하므로 오류를 추적하는 과정에서 정확히 소스코드의 위치를 파악할 수 있게 된다. (그림 1)은 이러한 흐름을 개괄적으로 보여준다. 모든 산출물을 XMI 또는 XML로 표현한다. UML이나 시험사례 그리고 JavaML은 자동으로 변환이 가능하다. 그리고 XPointer를 이용하여 문서를 포함한 문서 내의 요소들과 소스코드의 위치를 추적한다.

3. UML 기반 시험 추적

UML(Unified Modeling Language) 모델 기반 시험에서는 개발단계에 따라 사용사례 다이어그램(use-case diagram)이나 클래스다이어그램(class diagram), 순서다이어그램(sequence diagram), 상태도(state diagram)등을 생성한다. 그리고 이들을 변형하거나 시험에 필요한 의미를 추출하여 다양한 시험 사례를 설계하고 시험한다[21]. L. Naslavsky는 기능을 구현하고 있는 순서도에서 여러 경로를 조합하여 시험 사례를 작성하고 그 과정에서 추적 관계를 찾아낸다[22]. 또한 P. Zielczynski는 사용사례에서 시나리오를 중심으로 시험사례를 추출하고 추적 관계를 형성한다[23]. 이와 같은 모델과 시험 사례간의 연관성 파악은 시험 작업의 불필요한 중복을 예방하면서 전체 시험 공정을 파악할 수 있게 한다. 하지만 모델과 시험사례의 추상 수준이 기능 시험이나 통합 시험 수준에 머무르고 있고 단위 시험사례나 소스코드로의 추적성은 결핍되어 있다. 만약 기능 시험을 하다가 오류를 발견하면 그 기능을 중심으로 한 통합 시험이 필요하다. 마찬가지로 통합 시험에서 오류가 발견되면 그에 대한 단위 시험이 필요하다. 이때 만약 개발단계 뿐 아니라 시험단계에서 생성된 산출물 간의 추적 정보가 부족하다면 오류를 발견했을 때 수준별 산출물의 추적에



(그림 2) 추적 관계 개요

많은 노력이 필요할 것이다. 따라서 본 논문에서는 상위 수준의 시험 단계에서 오류를 발견했을 때 후속 시험 사례를 정확하게 선택할 수 있을 뿐 아니라 나아가 소스 코드로 접근할수록 정확한 오류 위치를 파악할 수 있는 방법을 제안한다.

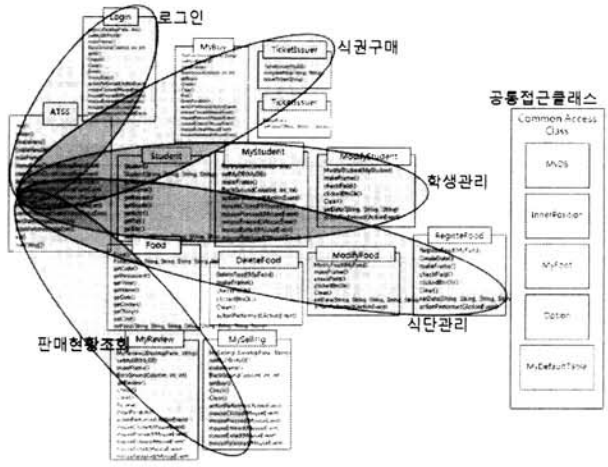
(그림 2)는 본 논문에서 다루고 있는 추적관계에 대한 전체적인 밑그림이다. 크게 모델간의 추적 관계와 모델에서 파생되는 동일 추상 수준의 시험사례간의 추적 관계로 나눌 수 있다. 모델은 추상 수준별로 사용사례와 순서다이어그램 그리고 클래스다어어그램과 소스코드 순으로 추적된다. 그리고 기능 시험 사례는 사용사례로부터, 통합 시험사례는 순서다이어그램으로부터, 단위 시험사례는 클래스다이어그램으로부터 추적된다. 따라서 추적의 방향은 모델을 중심으로 수준별 시험사례와 소스코드까지 이어진다.

4. 추적성 구현

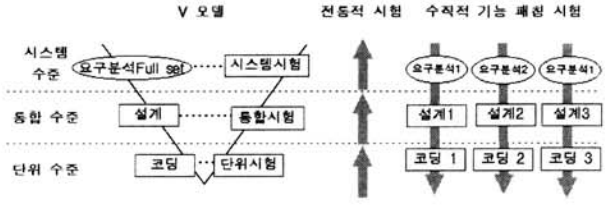
4.1 추적 시나리오

본 논문에서 제안한 추적성을 실현하기 위해 식권구매 시스템(Automatic Ticket Sales System)을 이용한다. (그림 3)은 ATSS의 주요 기능과 이를 구현하고 있는 클래스이다. 크게 로그인과 식권구매, 학생관리, 식단관리, 편매현황조회 기능이 있고 이들은 하나의 사용사례가 된다. (그림 3)을 보면 특정 기능을 구현하기 위해서 모든 클래스가 참여하는 것이 아니라는 것을 알 수 있다. 따라서 특정 기능을 수정하거나 추가했다면 시스템 전체 또는 기능을 구현하고 있는 모든 변수와 메소드를 시험하는 것보다 실제로 구현에 참여하고 있는 멤버들을 중심으로 시험 범위를 좁히는 것이 타당하다.

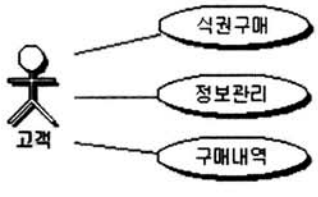
시스템을 구현한 이후에 기능 시험 단계에서 오류를 찾기 위한 과정은 (그림 4)의 하향식인 수직적 기능 시험과 같다[18]. 이는 전통적 시험 절차와 다르게 수직적인 기능시험을 하는 것이다. 본 논문에서는 ATSS의 식권구매 기능에 속한 MyBuy 클래스에 오류를 심어놓고 추적을 통해 소스코드의 어느 지점에 오류가 있는지 확인해 본다. 수직적 시험과 추



(그림 3) ATSS 기능별 구현 클래스



(그림 4) 수직적 시험 절차



(그림 5) ATSS 사용사례

적은 식권구매 기능을 먼저 시험하면서 시작한다. 그리고 오류를 발견하게 되면 순서다이어그램을 이용한 통합시험을 하고, 그 다음으로 단위시험을 함으로 점점 하위 레벨로 이동한다.

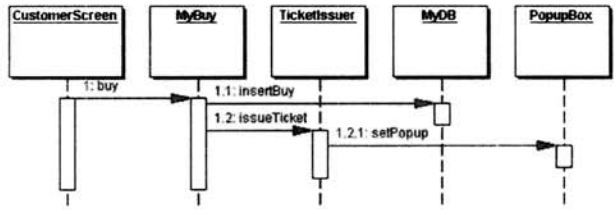
4.2 추적성 구현

먼저 UML과 시험사례, Java 소스코드를 각각 XMI와 XML 그리고 JavaML로 변환한 다음 XPointer를 이용해 추적관계를 구현한다.

(그림 5)는 ATSS의 사용사례이다. 본 논문에서는 식권구매에 대한 시험으로 추적 관계를 확인한다. 식권구매 기능을 이용하기 위해서는 1000000부터 9999999까지의 학번(loginID)과 식권구매 선택(TransCode), 그리고 원하는 날짜(ccDate)와 메뉴(menuCode)를 선택해야 한다. 유효한 데이터가 입력되고 기능이 명세대로 작동한다면 그 결과는 TicketIssuer 클래스의 insertSuccess의 값은 true가 된다. 이러한 명세를 기반으로 (그림 6)과 같이 시험사례를 XML로 표현한다. 라인 3, 11, 19, 27은 입력데이터를 정의하고 있고 라인 35는

```
1:<F-TC>
2:      <TCNo>F-TC1</TCNo>
3:      <InputData>
4:              <dataNo>1</dataNo>
5:              <attribute>loginID</attribute>
6:              <type>integer</type>
7:              <min>1000000</min>
8:              <max>9999999</max>
9:              <class>MyBuy</class>
10:     </InputData>
11:     <InputData>
12:             <dataNo>2</dataNo>
13:             <attribute>TransCode</attribute>
14:             <type>integer</type>
15:             <min>2</min>
16:             <max>2</max>
17:             <class>MyBuy</class>
18:     </InputData>
19:     <InputData>
20:             <dataNo>3</dataNo>
21:             <attribute>ccDate</attribute>
22:             <type>integer</type>
23:             <min>0</min>
24:             <max>2</max>
25:             <class>MyBuy</class>
26:     </InputData>
27:     <InputData>
28:             <dataNo>4</dataNo>
29:             <attribute>menuCode</attribute>
30:             <type>integer</type>
31:             <min>0</min>
32:             <max>2</max>
33:             <class>MyBuy</class>
34:     </InputData>
35:     <ExpectedResult>
36:             <Expt-ResultNo>1</Expt-ResultNo>
37:             <attribute>issertSuccess</attribute>
38:             <type>boolean</type>
39:             <value>true</value>
40:             <class>TicketIssuer</class>
41:     </ExpectedResult>
42:     <PassFail>
43:     </PassFail>
44:</F-TC>
```

(그림 6) 식권구매 기능 시험사례



(그림 7) 식권구매 순서다이어그램

시험 결과 불리언(boolean) 타입의 true를 기댓값으로 정의하고 있다.

만약 (그림 6)의 시험 결과 오류를 발견했다면 식권구매 기능을 이루고 있는 모듈에 대한 통합시험을 한다. 통합 시험사례는 하나의 기능을 실현하고 있는 순서다이어그램 상의 경로인 MM-Path(Method-Message Path)를 기준으로 한다[24]. (그림 7)은 식권구매 기능을 실현하고 있는 순서다이어그램의 일부분이다. MyBuy 클래스의 buy()를 통해 loginID에 1000000에서 9999999 사이의 값을 입력하면 MyDB객체의 record는 string 값이 들어가고 issueTicket()에 의해 TicketIssuer 객체의 insertSuccess는 true로 바뀐다. (그림 8)은 이에 대한 통합 시험사례의 XML 표현이다. (그림 8)의 라인 3은 시험데이터에 대한 정의이고 라인 18과 25는 시험 예측결과를 정의했다.

실패한 기능시험사례의 영역을 이루는 하위수준인 통합 시

```
<I-TC>
    <TCNo>I-TC4</TCNo>
    <InputData>
        <dataNo>1</dataNo>
        <attribute>loginID</attribute>
        <type>integer</type>
        <min>1000000</min>
        <max>9999999</max>
        <class>MyBuy</class>
    </InputData>
    <InputData>
        <dataNo>2</dataNo>
        <attribute>insertSuccess</attribute>
        <type>boolean</type>
        <value>false</value>
        <class>TicketIssure</class>
    </InputData>
    <ExpectedResult>
        <Expt-ResultNo>1</Expt-ResultNo>
        <attribute>insertSuccess</attribute>
        <type>boolean</type>
        <value>true</value>
        <class>TicketIssuer</class>
    </ExpectedResult>
    <ExpectedResult>
        <Expt-ResultNo>2</Expt-ResultNo>
        <attribute>record</attribute>
        <type>string</type>
        <value>string</value>
        <class>MyDB</class>
    </ExpectedResult>
</I-TC>
```

(그림 8) 식권구매 통합 시험사례

```
<U-tc>
    <TCNo>U-TC3</TCNo>
    <InputData>
        <dataNo>1</dataNo>
        <attribute>intData</attribute>
        <type>integer</type>
        <min>0</min>
        <max>2</max>
        <class>MyBuy</class>
        <method>getBuy</method>
    </InputData>
    <ExpectedResult>
        <Expt-ResultNo>1</Expt-ResultNo>
        <attribute>ccDate</attribute>
        <type>date</type>
        <min>08-Jan-11</min>
        <max>08-Jan-13</max>
        <class>MyBuy</class>
        <method>getBuy</method>
    </ExpectedResult>
</U-tc>
```

(그림 9) MyBuy 객체의 단위 시험사례

험사례를 통해 실패한 MM-Path를 이루고 있는 클래스를 중심으로 단위 시험을 할 수 있다. (그림 7)의 MM-Path를 구성하고 있는 클래스는 MyBuy, PopupBox, MyDB, TicketIssuer이다. 통합시험에서 MM-Path에 참여하고 있는 멤버 변수를 중심으로 각 클래스의 단위 시험사례를 설계한다. (그림 9)는 오류가 심어진 MuBuy 객체에 대한 시험사례의 일부이다. (그림 9)의 라인 3은 intData에는 정수로 0이나 1 또는 2만 입력이 가능하다. 그리고 결과는 날짜 타입의 08-Jan-11이나 08-Jan-12 또는 08-Jan-13이 출력되어야 한다.

시험데이터 0,1,2를 사용해 시험해보지만 결과는 08-Jan-11는 출력되지 않고 08-Jan-12 또는 08-Jan-13만 출력되었다. 따라서 예상한 시험결과 값이 출력되지 않기 때문에 오류가 있음을 감지하고 관련된 소스코드를 추적한다. (그림 10)은 실제 소스코드이다. intDate에 입력이 가능한 정수의 범위는 0-2이지만 실제 코드는 라인 167에 1, 라인 171에 2, 라인

```
162     public void getBuy(){
163         code = tfCode.getText();
164         //선택한 식단의 번호를 code에 할당
165         int intDate = cbDate.getSelectedIndex();
166
167         if (intDate == 1)
168         {
169             ccDate = "08-Jan-11";
170         }
171         else if (intDate == 2)
172         {
173             ccDate = "08-Jan-12";
174         }
175         else if (intDate == 3)
176         {
177             ccDate = "08-Jan-13";
178         }
179     }
```

(그림 10) MyBuy.getBuy()

175에 3 중 하나가 입력되도록 프로그래밍 되어있다. 즉 0-2가 아닌 1-3의 값을 입력하고 있으므로 비정상적으로 작동한 것이다. 지금까지 기능 시험사례에서 오류를 감지하면 관련된 하위 시험사례를 통해 소스코드까지 추적하고 있음을 보였다. 이러한 추적이 XML을 통해 가능하도록 JavaML로 표현하면 (그림 11)과 같다.

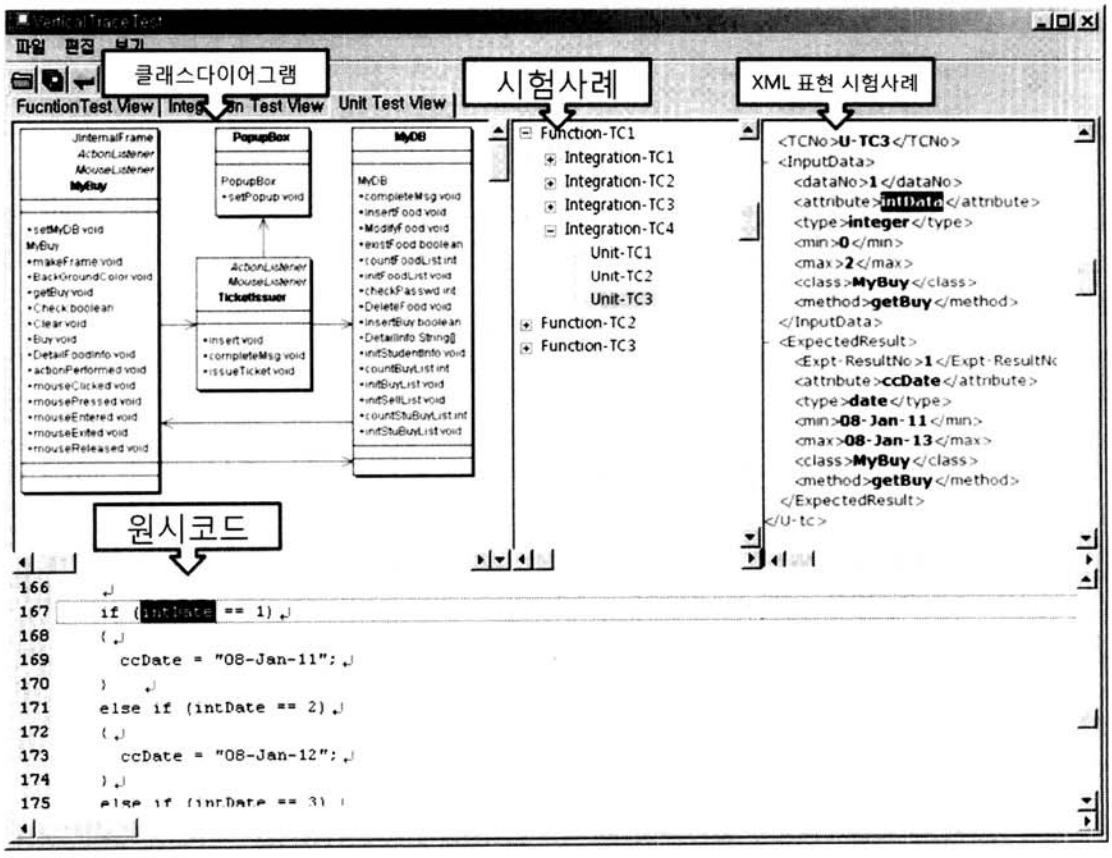
(그림 11)은 MyBuy.getBuy()를 JavaML로 표현한 일부분이다. 31번째 라인에서 메소드 getBuy()의 정의가 시작된다. 이때 소스코드 내에 있는 메소드 getBuy()의 위치와 가시성 등이 모두 나와 있다. 그리고 37번째 라인에서 if문 블록이 시작되고 있는데 첫 번째 검사 조건이 있는 if문의 위치가 168번 라인의 9번째 칼럼이고 해당 블록이 끝나는 위치는 170번째 라인의 9번째 칼럼임을 알 수 있다. 또한 (그림 10)에서 보는 바와 같이 각 조건에 의해 intDate의 값이 1에서 3까지 입력되는데, (그림 11)의 JavaML에서도 37번째 라인에서 1, 48번째 라인에서2, 57번째 라인에서 3이 입력되고 있다. 따라서 구체적인 코드의 위치까지 접근이 가능하다.

(그림 12)의 VTT(VerticalTraceTest) 프로토타입은 UML 모델과 시험사례 그리고 소스코드를 모두 추적할 수 있도록 지원한다. VTT는 크게 사용사례 중심의 기능시험 관점과 순서다이어그램 중심의 통합시험 관점 그리고 클래스다이어그램 중심의 단위시험 관점으로 나뉜다. 그 중 (그림 12)는 단위시험 관점을 표현하고 있다. 통합시험의 MM-Path를 이루고 있는 클래스 다이어그램과 단위 시험사례의 리스트가 보인다. 그리고 그 중 Unit-TC3의 XML 표현이 오른쪽 상단에 보인다. 그리고 하단의 윈도우는 시험사례와 관련 있는 원시코드를 보여준다. XML 표현 시험사례는 유효한 입력값이 0-2까지이지만 실제 코드는 1-2 임을 알 수 있다.

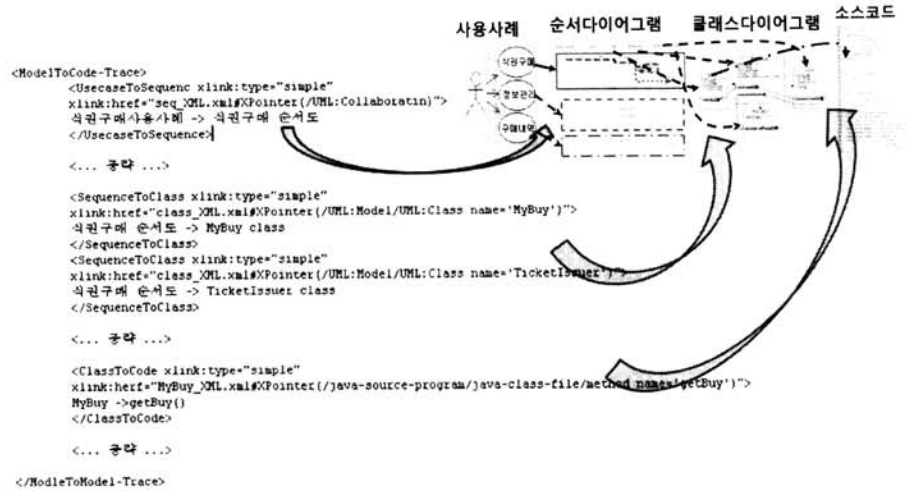
이러한 추적을 구현하기 위해 (그림 13) 그리고 (그림 16)과 같이 XPointer를 사용하여 모델간의 추적성과 모델과 시험사례간의 추적관계의 정의가 필요하다. (그림 13)은 식권구매사용사례에서 식권구매 순서다이어그램까지 그리고 순서다이어그램에서 클래스 다이어그램으로 마지막으로 클래스다이어그램에서 소스코드까지의 추적이다. <UsecaseToSequence xlink:type="simple" xlink:href="seq_XMI.xml#XPointer(/UML:

```
<method name="getBuy" visibility="public" id="MyBuy:mth-5" line="162" col="5" end-line="179" end-col="5">
<type name="void" provitive="true"/>
<block line="162" cole"26" end-line"179" end-com="2" comment="//선택한 식단의 번호를 code에 할당">

... 중략...

<if><test><binary-expr op="=="><var-ref name="intDate"/><literal-number kind="integer" value="1"/></binary-expr></test>
<true-case>

    <block line="168" cole"9" end-line"170" end-com"9">
    <assignment-expr op="=">
        <lvalue><avr-set name="ccDate"/></lvalue><literal-number kind="string" value="08-Jan-11"/>
    </assignment-expr>
    </block>
</true-case>

<false-case>
<if><test><binary-expr op="=="><var-ref name="intDate"/><literal-number kind="integer" value="2"/></binary-expr></test>
    <true-case>
    <block line="172" cole"9" end-line"174" end-com"9">
    <assignment-expr op="=">
        <lvalue><avr-set name="ccDate"/></lvalue><literal-number kind="string" value="08-Jan-12"/>
    </assignment-expr>
    </block>
    </true-case>
    <false-case>
    <if><test><binary-expr op="=="><var-ref name="intDate"/><literal-number kind="integer" value="3"/></binary-expr></test>
        <true-case>
        <block line="176" cole"9" end-line"178" end-com"9">
        <assignment-expr op="=">
            <lvalue><avr-set name="ccDate"/></lvalue><literal-number kind="string" value="08-Jan-13"/>
        </assignment-expr>
        </block>
        </true-case>
    </if>
    </false-case>
</if>
</false>
```

(그림 11) MyBuy.getBuy()의 JavaML



(그림 12) VTT의 단위시험 관점



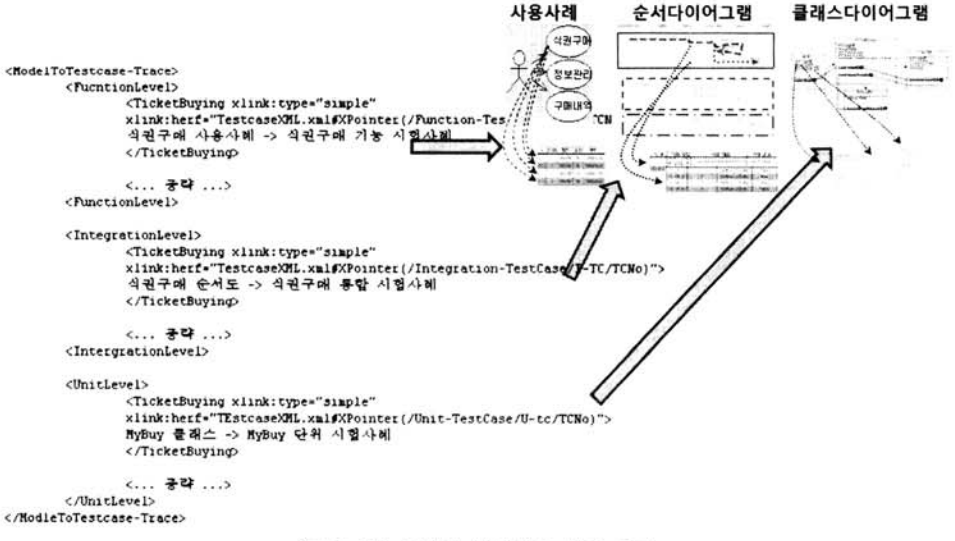
(그림 13) 모델에서 원시코드까지의 추적

```
17      <UML:Collaboration xmi.id = 'S.3'
18          name = 'Collaboration' visibility = 'public' isSpecification = 'false' isAbstract = 'false'>
19       <UML:Namespace.ownedElement>
20        <UML:ClassifierRole xmi.id = 'G.1'
21            name = 'CustomerScreen' visibility = 'package' isSpecification = 'false'
22            isAbstract = 'false' isActive = 'false'>
23         <UML:ClassifierRole.multiplicity>
24          <UML:Multiplicity>
25           <UML:Multiplicity.range>
26            <UML:MultiplicityRange lower = '1' upper = '1'/>
27           </UML:Multiplicity.range>
28          </UML:Multiplicity>
```

(그림 14) 순서다이어그램 XMI 일부(seq_XMI.xml)

```
14  <XMI.content>
15   <UML:Model xmi.id = 'S.1' name = 'Project' visibility = 'public'>
16    <UML:Namespace.ownedElement>
17     <UML:Class xmi.id = 'S.5'
18          name = 'MyBuy' visibility = 'public' isSpecification = 'false'
19          isAbstract = 'false' isActive = 'false'>
20      <UML:ModelElement.namespace>
21       <Foundation.Core.Namespace xmi.idref = 'S.4'/>
22      </UML:ModelElement.namespace>
23      <UML:Classifier.feature>
24       <UML:Attribute xmi.id = 'S.6'
```

(그림 15) 클래스다이어그램 XMI의 일부분(class_XMI.xml)



(그림 16) 모델과 시험사례 간의 추적

Collaboration)">는 순서다이어그램을 XML 형태로 정의하고 있는 XMI 파일에서 <Collaboration>이라고 명명한 순서다이어그램을 추적하고 있다.

(그림 14)와 같이 순서다이어그램이 XMI로 정의되어 있고 17번째 라인에서부터 식별자 S.3을 가진 순서다이어그램이 seq_XMI.xml 파일에 정의되어 있다. 따라서 (그림 13)의 XPointer에서 seq_XMI.xml 코드상의 요소를 접근할 수 있다.

(그림 15)는 클래스다이어그램을 XMI로 표현한 class_XMI.xml 파일의 일부분이다. 17번째 라인 이하에서는 식별자 S.5를 가진 MyBuy가 있는데 (그림 13)의 <SequenceTo Class xlink:type="simple" xlink:href="class_XML.xml#XPointer (/UML:Class name='MyBuy')">에서 멤버 클래스 MyBuy를 추적하고 있다. 즉 순서다이어그램에 참여하고 있는 MyBuy를 추적하고 있는 것이다.

이와 마찬가지로 클래스다이어그램에서 (그림 13)의 <Class ToCode xlink:type="simple" xlink:herf="MyBuy_XML.xml# XPointer(/java-source-program/java-class-file/method name ='getBuy')">를 통해 MyBuy_XML.xml 파일의 getBuy method를 참조할 수 있다. 즉 XPointer를 통해 (그림 11)에 보인 메소드 getBuy()를 추적하면 getBuy()에 대한 위치 정보와 메소드 안에 정의된 지역 변수와 조건 검사 등에 대한 정보를 정확히 알 수 있게 된다. 따라서 모델 기반으로 설계한 시험사례를 이용해 시험을 시행하다가 오류가 발견되면 오류가 발생한 소스코드의 위치를 정확히 찾을 수 있다.

지금까지는 모델을 기준으로 소스코드까지의 추적을 설명하였는데 이와 마찬가지로 모델과 시험사례의 추적을 제공해야 시험사례와 소스코드 간의 추적도 가능하다. (그림 16)은 모델과 시험사례간의 추적을 XPoint로 구현한 것이다. <TicketBuying xlink:type="simple" xlink:herf="TestcaseXML.xml#XPointer(/Function-TestCase/F-TC/TCNo)">에서는 기능 수준의 식권구매 사용사례에서 식권구매 기능 시험사례가 정의된 TestcaseXML.xml로의 추적을 제공하고 있다. <TicketBuying xlink:type="simple" xlink:herf="TestcaseXML.xml#XPointer (/Integration-TestCase/I-TC/TCNo)">에서는 통합 수준의 순서다이어그램에서 통합 시험사례로의 추적 과정이고, <TicketBuying xlink:type="simple" xlink:herf= "TEstcaseXML.xml #XPointer (/Unit-TestCase/U-tc/TCNo)"> 는 클래스다이어그램과 단위 시험사례간의 추적 과정이다. 이와 같이 XPoint를 이용하여 모델을 중심으로 시험에서 발생되는 시험사례와 소스코드간의 추적을 완성한다.

4.3 XML 기반 추적 방법의 비교

XML을 이용하여 추적 방법을 제안한 기존 연구 방법은 2.1절에서 소개한 바와 같이 XPointer를 이용한 Lumb의 연구[14]와 XPath를 이용한 Janathan의 연구[15] 그리고 XLink를 이용한 Lucia의 연구[16]와 XPointer를 이용한 Alves-Foss[18]의 연구가 대표적이다. 하지만 Lumb의 연구와 Bello의 연구는 추적의 대상이 소프트웨어 개발을 위한 산출물이 아니므로 비교에서 제외한다. 이 외에 본 논문에서 참고하고 있는 기존 연구들 중 산출물 추적과 그 결과를 시각적으로 보여주는 연구들을 대상으로 추적의 범위와 가시성을 <표 1>에서 비교해 본다.

추적의 범위는 산출물의 대상이 개발과 시험 단계를 기준으로 어느 수준까지 미치고 있는지를 말한다. 가시성은 여러 산출물들의 관계를 쉽게 파악할 수 있도록 해주는 역할을 하므로 추적 방법과 이를 효과적으로 표현하는 기능이 있는지 비교했다. 추적 대상은 수준별로 시스템모델, 통합모델, 단위모델과 소스코드 그리고 시험사례가 있다. 그리고 소스코드와 문장요소를 구별하여 추적이 소스코드의 위치와 이름에 대한 정보만 제공하고 있는지, 아니면 소스코드를 구성하고 있는 변수와 같은 요소까지 접근이 가능한지 구별하여 비교했다. Grechnik의 연구는 통합 모델이나 단위 모델 산출물은 고려하지 않고 시스템 모델과 관련있는 소스코드 사이의 관계를 추적할 수 있도록 지원한다. Sengupta나 Pinheiro 그리고 Ramesh의 연구는 시스템 모델에서 단위 모델까지 추적한다. Jonathan의 연구는 단위 모델과 소스코드 사이의 추적 방법은 있지만 이를 가시적으로 표현하는 방법은 결여되어 있다. Zhou와 Alves-Foss의 방법은 소스코드를 구성하고 있는 메소드나 변수들도 접근이 가능하다. 이러한 연구들은 모두 개발단계의 산출물만 고려하고 있다. 따라서 시험단계에서 생산되는 다양한 산출물을 개발단계의 산출물과 연계하기 어렵다. 그러나 본 논문에서 제안한 방법은 개발 단계의 산출물과 관련 있는 시험사례의 추적이 가능하고 또한 시험 수행 후 발견한 오류를 소스코드를 내에서 추적하여 정확한 위치를 찾을 수 있도록 지원하고 있다.

〈표 1〉 XML 기반 추적 방법 비교

	추적 범위						가시성					
	시스템 모델	통합 모델	단위 모델	소스 코드	문장 요소	시험 사례	시스템 모델	통합 모델	단위 모델	소스 코드	문장 요소	시험 사례
VerticalTrace	○	○	○	○	○	○	○	○	○	○	○	○
Grechanik[3]	○	X	X	○	X	X	○	X	X	○	X	X
Sengupta[4]	○	○	○	X	X	X	○	○	○	X	X	X
Pinheiro[5]	○	○	○	X	X	X	○	○	○	X	X	X
Ramesh[6]	○	○	○	X	X	X	○	○	○	X	X	X
Zhou[8]	X	X	○	○	○	X	X	X	○	○	○	X
Jonathan[15]	X	X	○	○	X	X	X	X	X	X	X	X
Alves-Foss[17]	X	X	○	○	○	X	X	X	○	○	○	X

5. 결 론

본 논문에서는 모델기반 시험을 할 때 수준별로 발생하는 개발 산출물과 시험사례간의 추적성을 구현하는 방법에 대해 서술했다. 추적성을 구현하는 방법은 기존의 연구와 도구가 있지만 본 연구에서는 변환이 용이하고 호환성이 뛰어난 문서 공유 표준인 XML을 사용했다. 또한 XPoint는 XML을 기반으로 한 추적 관계를 표현할 수 있는 표준이다. 이러한 XML 기반으로 모델간의 추적관계를 UML 모델의 사용사례에서 시작해서 순서다이어그램과 클래스 다이어그램 그리고 소스코드까지 구현했다. 그리고 수준별로 사용사례와 기능 시험사례, 순서다이어그램과 통합시험사례, 클래스다이어그램과 단위시험사례를 연결했다. 이러한 방법은 기존 연구들에 비해 더 넓은 추적성을 제공하고 있다. 그리고 추적관계와 더불어 추적의 대상을 직관적으로 쉽게 파악할 수 있는 세 가지 관점의 프로토타입을 제공했다. 따라서 모델을 기반으로 시험하는 과정에서 오류를 발견하면 하위 수준의 시험사례로 내려가면서 구체적인 오류 발생 지점까지 안내할 수 있다.

XML 기반 추적 작업은 여러 산출물을 포함하고 있고 이들을 XML로 변환하고, 변환된 XML의 추적관계를 완성해야 한다. 즉 추적에 참여하는 대상도 많고 관계도 다양하다. 이러한 추적성을 수작업으로 하는 것은 한계가 있다. 따라서 향후 자동화를 위한 연구가 필요할 것이다. 또한 산출물의 변경이 발생할 경우 변경된 추적관계를 유지보수하기 위한 관리 방법 연구도 추가로 필요하다.

참 고 문 헌

[1] Pretschner. A, Lotzbeyer. H, and Philipps. J, "Model based testing in evolutionary software development," Proc. of 12th Workshop on Rapid System Prototyping, 2001, pp.155-160.

[2] L. Briand and Y. Labiche, "A UML-based approach to system testing," Proc. of 4th Conf. on The Unified Modeling Language, Modeling Language, Concepts, and Tools, 2001, pp.194-208.

[3] M. Grechanik, K. S. McKinley, and D. E. Perry, "Recovering And Using Use-Case-Diagram-To-Source-Code Traceability Links," Proc. of 6th ISEC/FSE'07, 2007, pp.95-104.

[4] Sengupta. S, Kanjilal.A, and Bhattacharya. S, "Requirement Traceability in Software Development Process: An Empirical Approach," Proc. of the 19th IEEE/IFIP Symposium on Rapid System Prototyping, 2008, pp.105-111.

[5] F. Pinheiro, and J. Goguen, "An Object-oriented Tool for Tracing Requirements," IEEE Software, Vol.13, No.2, 1996, pp.52-64.

[6] B. Ramesh and M. Jarke, "Toward Reference Models for Requirements Traceability," IEEE Transactions on Software Engineering, Vol.27, No.1, 2001, pp.58-93.

[7] G. Antoniol, G. Canfora, G. Casazza, A. DeLucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," IEEE Transactions on Software Engineering, Vol.28, No.10, 2002, pp.970-983.

[8] X. Zhou, Z. Huo, Y. Huang, and J. Xu, "Facilitating Software Traceability Understanding with ENVISION," Annual IEEE International Computer Software and Applications Conference, 2008, pp.295-302.

[9] A. Marcus, X. Xie, and D. Poshyvanyk, "When and How to Visualize Traceability Links?," Pro. of 3rd international workshop on Traceability in emerging forms of software engineering, 2005, pp.56-61.

[10] Maletic, J. I., Munson, E., Marcus, A., and Nguyen, T., "Combining Traceability Link Recovery with Conformance Analysis via a Formal Hypertext Model," Proc. of 2nd International Workshop on Traceability in Emerging Forms of Software Engineering, Montreal, Canada, October 7th, 2003, 2003, pp.47-54.

[11] A. D. Lucia, F. Fasano, R. Oliveto, and G. Torotora, "Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods," ACM Transactions on Software Engineering and Methodology, Vol.16, No.4, Article13, 2007, pp.13-48.

[12] J. C. Huang, R. Settimi, E. Romanova, B. Berenbach, and S Clark, "Best Practices for Automated Traceability," 2007, IEEE Computer, Vol.40, Issue6, 2007, pp.27-35.

[13] C. Neumuller, and P. Grunbacher, "Automating Software Traceability in Very Small Companies: A Case Study, and Lessons Learne," IEEE International Conference on Automated Software Engineering, 2006, pp.145-156.

[14] L, I. Lumb, J. I. Lederman, J. R. Freemantle, and K. D. Aldridge, "Semantically Enabling the Global Geodynamics Project: Incorporating Feature-Based Annotations via XML Pointer Language(XPointer)," International Symposium on High Performance Computing Systems and Application, 2007, pp.21-27.

[15] J. I. Matetic, M. L. Collard, and B. Simoes, "An XML Based Approach to Support the Evolution of Model-to-Model Traceability Links," Proc. International workshop on Traceability in emerging forms of software engineering, 2005, pp.67-72.

[16] L. L. Bello, O. Mirabella, and N. Torrisi, "A General Approach to Model Traceability System in Food anufacturing Chains," Conf. Emmerging Technologies and Factory Automation, 2005, pp.19-22.

[17] J. Alves-Foss, D. C. de Leon, and P. Oman, "Experiments in the Use of XML to Enhance Traceability Between Object-Oriented Design Specifications and Source Code," Proc. Hawaii International Conference on System Sciences, 2002, pp.3959-3966.

[18] K. I. Seo and E. M. Choi, "Rigorous Vertical Software System Testing in IDE," Proc. Software Engineering

Research, Management & Application, 2007, pp.847-854.
[19] G. Badros, "JavaML: A Markup Language for Java Source Code," University of Washington, 2000, Web Site: http://www.cs.washington.edu/research/constraints/web/badros-javaml-www9.pdf
[20] E. Mamas, and K. Kontogiannis, "Towards Portable Source Code Representation Using XML," Proc. Reverse Engineering Working Conference, 2000, pp.172-182.
[21] K. Seo, and E. M. Choi, "Comparision of five black-box testing methods for object-oriented software," Proc. of the 4th ACIS Internation Conf. on Software Enginerring Research, Management & Applications, 2006, pp.213-222.
[22] L. Naslavsky, H. Ziv, and D. J. Richardson, "Towards Traceability of Model-based Testing Artifacts," Proc. internatinal workshop on Advances in Model-based Testing, 2007, pp.105-114.
[23] P. Zielczynski, "Traceability from Use Cases to Test Cases," IBM Rational Technical Library, Web page: http://www.ibm.com/developerworks/rational/library/04/r-3217/
[24] P. C. Jorgensen, C. Erickson," Object-Oriented Integration Testing," Communicatin of ACM, Vol.37, No.9, 1994, pp.30-38.

서 광 익

e-mail : bradseo@dongguk.edu
2002년 동국대학교 컴퓨터공학과(학사)
2004년 동국대학교 컴퓨터공학과(공학석사)
2006년 동국대학교 컴퓨터공학과(박사수료)
관심분야 : 소프트웨어 테스트, 소프트웨어 품질, 프로세스, Traceability

최 은 만

e-mail : emchoi@dgu.ac.kr
1982년 동국대학교 전산학과(학사)
1985년 한국과학기술원 전산학과(공학석사)
1993년 일리노이 공대 전산학과(공학박사)
1985년~1988년 한국표준연구소 연구원
1988년~1989년 데이콤 주임연구원
2000년, 2007년 콜로라도 주립대 전산학과 방문교수
1993년~현 재 동국대학교 컴퓨터공학과 교수
관심분야 : 객체지향 설계, 소프트웨어 테스트, 프로세스와 메트릭, Program Comprehension, AOP