프레임워크기반 웹 어플리케이션을 위한 BizUnit 테스트 코드 생성

(A BizUnit Test Code Generation for Framework-based Web Application)

이 은 영 † 최 병 주 †† 송 화 정 ††† 황 상 철 ††††
(Eunyoung Lee) (Byoungju Choi) (Hwajung Song) (SangCheol Hwang)

요 약 웹 어플리케이션의 활용과 그 시장이 압도적으로 성장하면서 그 기능이 확대 심화되고 있다. 오늘날 나날이 높아지는 소프트웨어의 고품질 요구와 맞물려 웹 어플리케이션의 테스트에 대한 관심도 급증하고 있다. 웹 어플리케이션은 개발환경적인 면에서 프레임워크 기반으로 개발되고 있는 추세로 그 프레임워크의 영역이 확장될수록 전체 웹 어플리케이션의 각 모듈은 이질적인 파일들의 조합으로 구성되고 있다. 반제품의 형태로 제공되는 프레임워크가 전체 개발 대상의 구조를 제어한다는 점에서 웹 어플리케이션만의 특성을 갖게 된다. 본 논문에서는 웹 어플리케이션의 실행 단위로써 의미를 가지는 최소 단위로 웹 비즈니스 로직을 정의하고, 이에 대한 BizUnit 테스트 코드를 자동생성하는 방안을 제안하며, BizUnit을 통해 효과적으로 웹 어플리케이션을 테스트하는 것을 분석한다.

키워드 : 웹 어플리케이션, 프레임워크, 웹 어플리케이션 테스트, JUnit

Abstract With greater utilization of web application and growth of its market, the function of such application is being expanded With today's increasing demand for high quality software, we see a sharp rise in the interest for web application test. The current trend is that web applications are being developed based on a framework of developmental environment. Hence, as the scope of that framework expands, each module of the entire web application is being configured by the combination of heterogeneous files. The distinctive characteristics of a web application are based on the fact that the framework that is provided in partially completed form controls the structure of all objects of development. The present study defines the web business logic as a minimal unit which has meaning as a unit of application for the web application. In this paper, we define web business logic as a least meaningful execution unit of web applications, propose how to generate automatically the BizUnit test code for it, and analyze the effectiveness of testing of the web application via the BizUnit test codes.

Key words : Web Application, Framework, Web Application Test, JUnit

· 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원 사업의 연구결과로 수행되었음(NIPA-2009-(C1090-0903-0004))

† 학생회원 : 이화여자대학교 컴퓨터공학
ddonggurami@ewhain.net
†† 종신회원 : 이화여자대학교 컴퓨터정보통신공학과 교수
bjchoi@ewha.ac.kr
††† 학생회원 : 이화여자대학교 컴퓨터정보통신공학과
hwajungsong@ewhain.net
†††† 정 회 원 : SAMSUNG SDS Engineering Methodology Team 책임
hwangsc@samsung.com

논문접수 : 2009년 1월 6일
심사완료 : 2009년 9월 26일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제12호(2009.12)

1. 서 론

자바는 수년에 걸쳐 엔터프라이즈 환경에 맞는 어플리케이션을 개발하기 위한 대표적인 플랫폼으로 자리잡았으며 그 중심에는 웹 어플리케이션(Web Application)이 있다고 말할 수 있다. 오늘날 웹 어플리케이션은 인터넷 상에서 고객과 기업을 연결하는 중요한 도구로써 그 시장이 압도적으로 성장하여 블로그, 온라인 상점, 인터넷 뱅킹, 기업간 서비스 등의 모든 분야에서 널리 사용되고 있다. 급증하는 사용량에 따라 웹 어플리케이션의 고품질에 대한 고객의 요구 또한 증가하고 있다.

객체 지향 언어로써 개발속도를 단축시키고 높은 이식성으로 다양한 플랫폼에 적용될 수 있기 때문에 자바는 웹 어플리케이션를 개발하는 중심 언어가 되었다. 또한

웹 어플리케이션은 프레임워크를 기반으로 개발되고 있다. 프레임워크[1,2]란 유사한 어플리케이션들의 개발을 위해 공통적으로 개발되는 부분을 재사용하기 위해 반제품의 형태로 모듈화하여 제공하는 것을 뜻하며 웹 어플리케이션을 위한 프레임워크들 역시 자바로 구현된다.

프로그래밍언어나 플랫폼에 따라 다양한 버전으로 제공되는 테스트 프레임워크(Test Framework)를 xUnit이라 하며 그 중 JUnit은 자바를 위한 것으로 자바 어플리케이션을 테스트하기 위한 테스트 코드 작성을 지원하는 산업표준으로 자리잡았다[3,4]. 그러므로 웹 어플리케이션을 테스트하는 경우 JUnit을 이용한 테스트 자동화 도구가 좋은 해결책이 될 수 있을 것이다. 현재의 JUnit테스트 자동화 도구[11,12,13]는 웹 어플리케이션을 구성하는 클래스의 메소드 단위에 대한 JUnit테스트 코드를 생성한다.

프레임워크를 기반으로 개발을 하는 경우가 일반화되어감에 따라 기존의 프레임워크를 사용하지 않고 순수한 객체만으로 직접 모든 모듈을 개발하는 방법론을 POJO(Plain Old Java Object) 기반 개발이라 부르게 되었다. POJO기반 개발과 달리, 프레임워크 기반 개발로 인해 실제 개발하는 부분은 비교적 간단해지는 반면 웹 어플리케이션의 구성요소가 프레임워크의 제어를 통해 서로 상호작용하는 부분의 비중이 커지게 된다. 개발한 부분만의 단위 테스트 보다는 개발영역, 프레임워크 영역 및 데이터베이스 사이의 상호작용을 테스트하는 것이 보다 중요하다. 그러나 대다수의 JUnit 테스트 자동화 도구가 자동 생성하는 JUnit 테스트 코드는 웹 어플리케이션의 클래스의 메소드 하나 하나를 단위로 하는 단위테스트 용도로 한정되기 때문에 웹 어플리케이션을 테스트하기 위한 도구로써 충분하지 못함을 알 수 있다. 이에 따라 웹 어플리케이션의 도메인 특성을 잘 반영한 테스트 방안이 필요하다.

본 논문에서는 웹 어플리케이션을 테스트하는 최소 단위인 웹 비즈니스 로직(Web Business Logic)을 제안하며, JUnit 기반의 BizUnit을 통해 효과적으로 웹 어플리케이션을 테스트하는 것을 보이고자 한다. 또한 웹 어플리케이션으로부터 웹 비즈니스 로직 추출과 BizUnit 생성을 자동화한 웹 비즈니스 로직 테스트 지원 도구인 Testopia_biz를 소개한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 기술하고, 3장에서는 웹 어플리케이션 테스트를 위한 비즈니스 로직 및 BizUnit 테스트 코드에 대해 기술하고, 4장에서는 웹 비즈니스 로직 테스트 자동화를 기술하고, 테스트 자동화 도구인 Testopia_biz를 소개한다. 5장에서는 샘플 웹 어플리케이션을 테스트한 사례와 6장에서 분석해 기술하고, 7장에서는 결론 및 향후 과제를 기술한다.

2. 관련 연구

xUnit 은 다양한 프로그래밍언어나 플렛폼에 따라 구성 단위에 대해 환경 설정, 테스트 메소드와 같은 테스트 구조를 제시하는 테스트 프레임워크로써 JUnit, HttpUnit, HtmlUnit, XmlUnit, DBUnit 등과 같은 다양한 종류가 있다[3]. 이 중에 JUnit은 자바 언어를 위한 것으로 자바 프로그램을 테스트하는 가장 일반적인 방법으로 자리매김 하고 있다[5].

JUnit테스트 코드는 그림 1(b)에서처럼 크게 테스트 환경설정과 테스트 메소드로 구성되며 테스트 메소드는 테스트 데이터 입력, 테스트 대상 실행, 테스트 결과 판정 등의 3가지 구조를 갖는다.

JUnit으로 수행할수 있는 단위테스트는 클래스의 메소드 단위 테스트가 있을 수 있다. 예를 들어, 그림

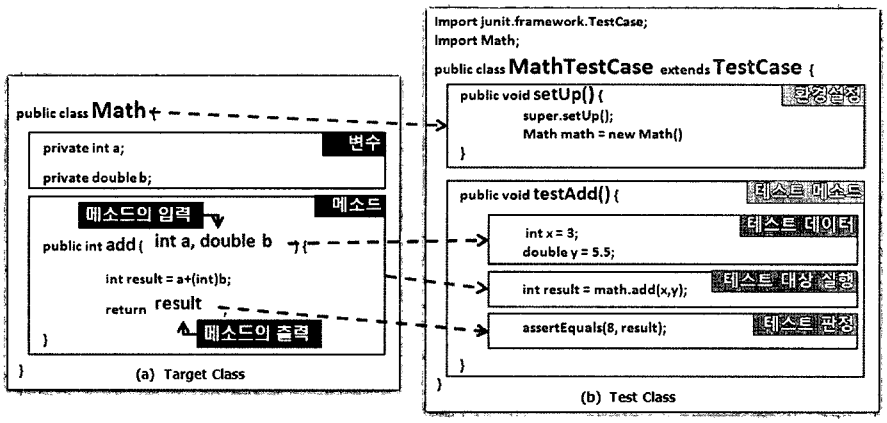


그림 1 클래스-메소드 단위 JUnit 테스트 코드 예

1(b) 예제와 같이 그림 1(a)의 테스트 타겟 클래스의 메소드에 대해 메소드 입력을 위한 데이터를 입력하고 메소드를 실행시킨다. 메소드의 출력에 대한 기대값 비교와 같이 수행결과를 자동으로 검증하면 메소드 단위의 테스트를 수행했다고 볼 수 있다. 만약 테스트 대상 실행에서 어플리케이션 전체를 구동하도록 한다면 전체 어플리케이션이 테스트 대상이 될 수 있으며 환경설정 영역에서 어플리케이션이 상호작용하는 다른 시스템과 연동할 수 있다면 다른 시스템과의 통합테스트도 수행할 수 있게 된다. 이와 같이 JUnit으로 테스트를 할 수 있는 단위는 최소한 클래스의 메소드부터 전체 시스템까지 다양하게 확장할 수 있다.

각 도메인의 특성과 수행하고 있는 테스트 레벨에 따라 JUnit을 활용할 수 있기 때문에 가장 폭넓게 개발자들에게 사용되고 있지만 테스트 도구는 개발자의 단위 테스트 레벨의 클래스 메소드 단위 테스트를 위해 개발된 경우가 가장 많다. 현재 많은xUnit 테스트 자동화 도구가 개발되었다. 특히 대표적인 JUnit 테스트 도구로는 IMB Rational사의 CodePro[11], Agita사의 Agita-One[12], ParaSoft사의 JTest[13]가 있다. 이들은 모두 JUnit 테스트 코드를 자동 생성하며, 개발환경인 Eclipse에 플러그인 도구로 지원되며 자동 생성된 테스트 코드를 실행하여 테스트 판정을 할 수 있도록 한다.

JUnit 테스트코드를 자동생성한다는 의미는 테스트 메소드와 테스트 메소드를 구성하는 테스트 데이터, 테스트 대상 실행, 결과 판정문을 자동생성하는 것으로 요약된다. 이들 도구는 테스트 코드에 필요한 테스트 데이터 및 테스트 메소드를 자동생성한다. 즉 테스트 대상의 클래스에 정의된 모든 메소드에 대한 테스트 메소드를 자동생성한다. 문제는 테스트 메소드에 필요한 테스트 데이터와 판정에 필요한 결과값을 얼마나 자동생성할 수 있는가에 달려있다.

자동화 도구가 생성할 수 있는 테스트 데이터의 종류는 POJO 기반의 기본(Primitive) 객체이며, 소스 정적 분석에 의해 판단될 수 있는 정보를 최대한 활용하지만 의미없는 임의의 값을 무작위로 생성한다. 예를 들어, String 객체에 대해서는 "abc", "testdata001"과 같은 데이터를 생성하고 Integer 객체에 대해서는 1, -100과 같은 데이터를 생성한다. 반면에 라이브러리로 제공되는 객체에 대해서는 자동화 도구가 객체에 대한 정보가 없기때문에 테스트 데이터를 생성할 수 없다. 예를들어 프레임워크가 제공하는 클래스 등에 대해서 자동화 도구는 테스트 데이터를 생성하지 못한다.

테스트 판정문은 소스 코드에 대한 정적분석을 통해 메소드 반환 변수 값이나 메소드 내부에서 사용된 변수 값 등이 기준이 되어 하나 이상 생성된다. 그러나 복잡한 로직을 구현할수록 테스트 중요도는 높아지는 반면 자동화 도구가 로직을 이해하여 판정문의 기준값을 알아내기는 더욱 어려워지기 때문에 판정문의 수준은 낮은 편으로 볼 수 있다.

메소드 내에서 다른 클래스의 함수를 호출하는 경우, 그 클래스의 함수가 소스상에 존재하는 경우를 제외하고 각 도구가 제공하는 스텁으로 호출한 실제 클래스를 대신하여 독립적인 테스트를 수행할 수 있도록 도와준다. 즉, 이들 자동화 도구는 클래스의 메소드 단위의 독립적인 테스트 수행을 위해 필요한 테스트 데이터에 관해 임의의 값을 생성하여 JUnit 테스트 코드를 생성함으로써 클래스 단위 테스트 용도의 도구이다. 문제는 웹 어플리케이션의 경우, 라이브러리 형태로 제공되는 프레임워크를 웹 어플리케이션의 클래스 파일에서 사용한다. 이때에 위의 테스트 도구들은 서블릿이나 리퀘스트와 같은 웹 프레임워크 레이어에 해당하는 객체를 라이브러리 객체로 인식하기 때문에 프레임영역과 웹 어플리케이션 개발영역의 상호작용을 테스트하지 못하게 된다.

3. 웹 어플리케이션의 비즈니스 로직

급변하는 비즈니스 환경과 함께 웹 어플리케이션 규모와 복잡도는 점점 증가하여 이에 대한 해결책으로써 프레임워크[1,2,6]가 대두되었다. 프레임워크가 제공하는 검증된 서비스와 공통 모듈을 재사용하여 개발함으로써 높은 개발 생산성을 가져오게 되었다. 프레임워크 기반으로 개발되는 웹 어플리케이션이 갖게 되는 특성을 파악하기 위해 본 논문에서는 오픈 프레임워크인 애니 프레임(Anyframe)[6]을 택하여 웹 어플리케이션을 분석한다.

3.1 애니 프레임을 통해 본 웹 어플리케이션 구조

애니 프레임은 업무용 프로그램 개발을 효과적으로 진행하기 위해 기본 아키텍처, 기술 공통 서비스, 템플릿 등을 제공하는 자바 기반의 어플리케이션 프레임워크로써 업계 표준으로 활용되는 스트러츠(Struts), 스프링(Spring), 하이버네이트(Hibernate), 아이바티스(iBatis) 등의 다양한 오픈 소스를 활용하여 구성되어 있다[6]. 아래 그림 2는 애니 프레임 기반으로 개발되는 웹 어플리케이션의 구조를 개념적으로 표현한 것이다. MVC 모델의 3계층 아키텍처를 따라 프레젠테이션 계층(Presentation Layer) - 비즈니스 계층(Business Layer) - 퍼시스턴스 계층(Persistence Layer)으로 구조화되어 있다. 개발 영역인 하얀색 부분은 프레임워크가 참조하는 XML설정 파일와 JSP 형식의 웹 페이지 파일, 변수 객체나 실제 웹 어플리케이션이 제공하는 기능을 구현하는 Java 클래스 파일로 구성되며 표 1과 같이 정리할 수 있다.

• **설정파일(XML)**은 XML 형식으로 개발되며 프리젠

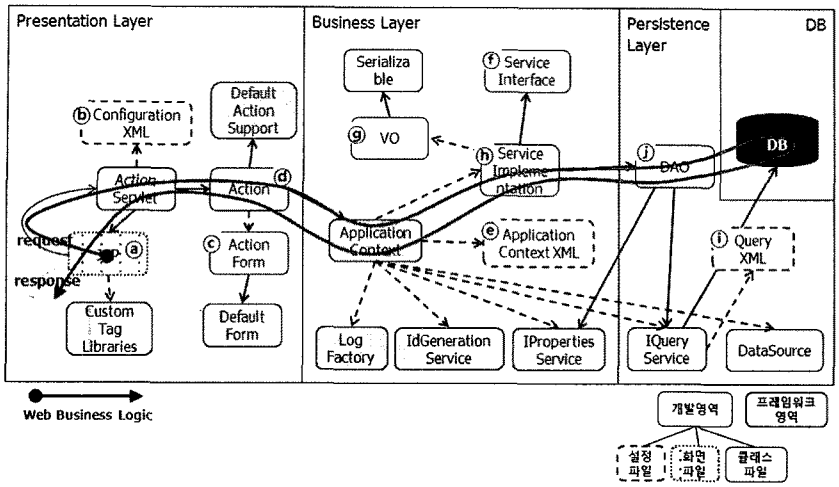


그림 2 애니 프레임 기반 웹 어플리케이션의 구조

표 1 파일 종류에 따른 실제 웹 어플리케이션 파일 대상

파일 종류		웹 어플리케이션 구조에서 실제 대상
설정 파일(XML)		Configuration XML, Application Context XML, Query XML
클래스 파일(JAVA)	기능 구현클래스	Action, Service Interface, Service Implementation, DAO
	변수 클래스	Action Form, VO
화면 파일(JSP)		JSP

테이션 계층은 Configuration XML 파일, 비즈니스 계층은 Application Context XML 파일, 퍼시스턴스 계층은 Query XML 파일 등으로 웹 어플리케이션에서 배포 명세서인 Web.xml에 각각 정의된다. 그림 2의 Configuration XML 파일은 화면에서 발생하는 액션에 대해 발생한 화면 이름, 발생한 이벤트 이름, 사용된 액션 폼, 실제 처리하는 클래스 이름, 클래스가 이벤트를 처리한 후 반환되는 화면이나 이벤트 정보를 담고 있다. 이러한 정보들은 이것은 프레젠테이션 계층을 제어하는 애니프레임의 액션 서블릿(Action Servlet)의 리퀘스트 프로세서(Request Processor)가 참조하여 그림 2의 JSP 화면과 Action 클래스와 ActionForm 클래스를 연결시켜준다. Application Context XML은 웹 어플리케이션의 실제 기능을 구현한 클래스들의 상관관계나 구조를 Beans로 정의하여 비즈니스 계층에 존재하는 모든 클래스에 대한 정보를 갖고 있다. 퍼시스턴스 계층을 위한 Query XML에는 웹 어플리케이션의 데이터베이스의 테이블이나 쿼리를 비즈니스 계층의 클래스의 변수나 메소드에 연결을 위한 세부 정보를 갖고 있다.

- **클래스 파일(JAVA)**은 웹 어플리케이션의 실제 기능을 구현한 부분으로써 Action, Service Interface, Service Implementation, DAO 클래스 파일과 Action Form, VO 변수 객체로 구성한다. 웹 어플리케이션의 실제 기능은 기본적으로 이들 클래스 파일에서 구현하게 되며, 이들은 애니프레임이 제공하는 모듈과 연결된다. 그림2에서 화살표로 표시된 부분이 실제 구현한 클래스와 프레임워크가 제공하는 모듈 사이의 상호 관계를 나타낸다.
- **화면 파일(JSP)**은 HTML, PHP, MiPlatform, JSP 등과 같이 웹 페이지를 구현하는 많은 기술이 개발되고 있어서 각 프레임워크에서 다양하게 이 기술들을 사용하고 있으며 애니 프레임의 경우, JSP 파일로 웹 페이지를 구현하고 있다. 화면파일에서 실제 사용자에게 데이터 값을 입력받는 필드와 기능에 따른 이벤트를 구현하고 화면에서 바로 처리되어 보여질 간단한 함수는 자바 스크립트로 코딩하게 된다. 각 화면에서 발생하는 이벤트가 웹 어플리케이션을 동작시키는 것으로 볼 수 있다. 이러한 화면에서 발생하는 이벤트는 일정한 URL 패턴이 Web.xml에 정의되고 이벤트는 프레젠테이션 계층의 서블릿이 처리한다.

3.2 웹 비즈니스 로직

웹 어플리케이션은 화면을 통해 서비스를 제공하며 그림 2를 예로 앞 절에서 설명하였듯이 웹 어플리케이션의 실제 기능을 구현한 모듈은 프레임워크의 제어를 통해 서로 상호적으로 작용하여 실행된다. 웹 어플리케

이션의 모듈을 각각 테스트하는 단위테스트 (즉, 클래스 단위테스트)로는 웹 어플리케이션의 모든 결함들을 찾아내기 어려우며 프레임워크를 포함한 각 구성 요소들과의 상호작용을 테스트하는 것이 매우 중요하다.

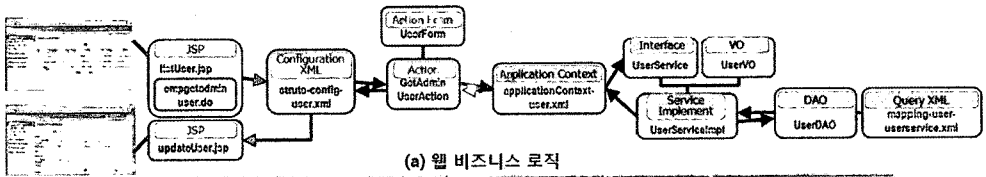
웹 어플리케이션은 일반적인 소프트웨어와는 달리 사용자가 어플리케이션을 접하는 환경은 JSP화면에 해당하는 웹 페이지로 국한되며, 개발자가 사전에 사용자의 입력을 제한해 어플리케이션이 정확한 동작을 하는 흐름만을 선택하도록 제약을 두고 있다. 그림 2에서 JSP 화면으로 들어온 사용자 요청은 프레임워크에서 제공하는 라이브러리 계층과 함께 연동하며, 프레젠테이션 계층과 비즈니스 계층, 퍼시스턴스 계층을 거쳐 데이터베이스를 경유하고 다시 JSP화면으로 돌아오는 일련의 실행 흐름을 갖는다. 이렇게 JSP 화면에서 시작하여 다시 JSP 화면으로 돌아오기까지 실행되는 실행 흐름을 웹 비즈니스 로직이라고 정의한다.

비즈니스 로직을 구체적으로 예를 들면, 그림 3(a) listUser화면에서 empgetadminuser.do 이벤트가 발생하여 액션 서블릿이 struts-config-user.xml 파일을 참조하여 userForm클래스에 화면 데이터를 저장하고, 이벤트 처리를 위해 GetAdminUserAction 클래스를 호출한다. GetAdminUserAction 클래스는 ApplicationContext를 이용하여 userService bean을 호출한다. ApplicatoinContext는 applicationContext-user.xml에 정의된 비즈니스 계층의 UserServiceImpl클래스에게 처리를 넘긴다. UserServiceImpl의 getUser 메소드는 UserDAO의 getUser 메소드를 호출하여 이 때 퍼시스턴스 계층의 IQueryService를 통해 DB에서 화면에서 입력된 데이터와 매치하는 키를 가지는 테이블 값을 가져온다. IQueryServie는 mapping-user-userservice.xml 설정 파일을 참조한다. DB에서 가져온 값은 순차적으로 반환되어 UpdateUser 화면으로 돌아온다.

3.3 BizUnit

본 논문에서는 웹 비즈니스 로직을 테스트 할 수 있도록 JUnit 형식의 테스트 코드로써 BizUnit을 제안한다. 먼저 BizUnit 테스트 코드를 예로 설명하면, 그림 3(b)이 웹 비즈니스 로직의 예로 들었던 그림 3(a)의 BizUnit 테스트 코드이다. 즉, listUser화면의 empgetadminuser.do 이벤트로 시작하여 updateUser화면으로 돌아오는 웹 비즈니스 로직 "BizUnitGetAdminUser"을 테스트 할 수 있도록 하는 BizUnit 테스트 코드 "BizUnitGetAdminUserTestCase"이다.

BizUnit은 비즈니스 로직을 테스트할 수 있도록 하는 테스트메소드를 가지며, 실행환경설정, 테스트데이터, 테



(a) 웹 비즈니스 로직

```
import servletunit.struts.MockStrutsTestCase;
public class BizUnitGetAdminUserTestCase extends MockStrutsTestCase {
    private final String webappPath = "./src/webapp";
    private final String CONFIG_LOCATIONS = "classpath*:/config/spring/applicationContext-*.xml";
    private ConfigurableWebApplicationContext WEB_APP_CTXT = null;
    public void setUp() throws Exception {                                  프레임워크 환경설정
        super.setUp();                                                      프레젠테이션 계층 설정 파일
        setInitParameter("validating", "false");
        setContextDirectory(new File(webappPath).getCanonicalFile());
        setConfigFile("/config/struts/struts-config.xml");
        setConfigFile("/config/struts/struts-config-user.xml");
        if (WEB_APP_CTXT == null) {                                         비즈니스 계층 설정 파일
            ContextLoader ctxLoader = new ContextLoader();
            context.setInitParameter(ContextLoader.CONTEXT_CLASS_PARAM, XmlWebApplicationContext.class.getName());
            context.setInitParameter(ContextLoader.CONFIG_LOCATION_PARAM, CONFIG_LOCATIONS);
            ctxLoader.initWebApplicationContext(context);
            WEB_APP_CTXT = (ConfigurableWebApplicationContext) WebApplicationContextUtils.getRequiredWebApplicationContext(context);
        } else {
            WEB_APP_CTXT.setServletContext(context);
            context.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE,WEB_APP_CTXT);
        }
    }
    public void testGetAdminUserAction() throws Exception {
        addRequestParameter("userId", "test");          테스트 입력          테스트 데이터
        setRequestPathInfo("/empgetadminuser.do");                          테스트 대상 수행
        actionPerform();
        verifyNoActionErrors();                                             테스트 판정
        verifyForward("success");
        assertEquals("Error", this.getActualForward(), "/sample/user/updateUser.jsp");  개발자 입력
    }
```

(b) BizUnit

그림 3 웹 비즈니스 로직과 BizUnit

스트대상수행, 테스트 판정으로 구성한다. 그림1(b)의 클래스 단위 테스트에 대한 JUnit의 경우 TestCase를 상속받는 것과 달리, BizUnit은 MockStrutsTestCase[7]를 상속받는다. MockStrutsTestCase는 JUnit의 스트러츠 확장형태로 웹 어플리케이션을 실행할 수 있도록 관련 프레임워크의 객체를 생성하고 설정파일을 정의하는 등의 환경 설정을 지원한다.

• BizUnit 테스트코드 구성요소

① **실행환경설정**: setup()은 해당 웹 비즈니스 로직 테스트 메소드를 실행시키기 위해 필요한 환경설정 변수 및 함수를 초기화한다.

② **테스트메소드**: 웹 비즈니스 로직을 테스트하는 테스트 메소드이다. 예를 들어 그림 3(a)의 경우, listUser 화면의 empgetadminuser 이벤트에 대한 실행부분은 GetAdminUserAction 액션 클래스에 해당하므로 testGetAdminUserAction()가 테스트 메소드이다.

③ **테스트데이터**: 비즈니스 로직을 시작하는 화면 입력 테스트 데이터 값이다. 화면 입력 변수는 그림 3(a)의 UserForm 객체에 변수명과 그 타입이 정의되어 있다. MockStrutsTestCase 클래스의 addRequestParameter() 함수를 이용하여 화면에 대한 테스트 데이터 값을 입력한다. 예를 들어 그림 3(b)의 경우, listUser 화면에 대한 userId라는 변수에 "test"라는 값을 입력함을 의미한다.

④ **테스트대상수행**: 웹 비즈니스 로직을 실행한다. MockStrutsTestCase 클래스의 setRequestPathInfo()함수의 파라메터로 이벤트 이름을 입력하고 actionPerform()으로 그 이벤트가 실행하게한다. 예를 들어, 그림 3(b)의 경우 empgetadminuser.do를 실행한 예이다.

⑤ **테스트판정**: MockStrutsTestCase 클래스의 판정 메소드인 verifyNoActionErrors(), verifyForward(), assertEquals ()등을 사용하여 테스트 판정을 한다. 예를들어, 그림 3(b)의 경우 웹 비즈니스 로직인 "BizUnitGetAdminUserTestCase"의 결과화면이 UpdateUser.jsp 화면인지 assertEquals 메소드를 통해 확인한 예이다.

웹 비즈니스 로직을 단위로 웹 어플리케이션을 테스트하는 것은 웹 어플리케이션의 실제 실행단위를 만족하고, 프레임워크의 영역을 테스트에 포함하여 프레임워크와의 상호작용을 볼 수 있다는 장점이 있다. 또한 웹 비즈니스 로직은 화면을 기준으로 테스트 단위가 결정되기 때문에 테스트 데이터는 사용자가 화면에 실제 입력하는 데이터를 그대로 활용할 수 있다. 개발자의 단위 테스트 레벨에서 실행 단위와 사용자 인터페이스인 화면을 고려하기 때문에 점진적인 테스트 확장이 용이하다.

4. BizUnit테스트 자동화: 테스토피아_비즈 (Testopia_biz)

테스토피아_비즈는 웹 어플리케이션에 대한 BizUnit을 자동생성하는 테스트 도구이며, 현재 애니 프레임에 맞춰 개발하였으며, 자바 개발환경인 이클립스(Eclipse)의 플러그인 형태로 개발 하였다. 테스토피아_비즈는 다음과 같은 모듈로 구성한다.

- Analyzer: 웹 어플리케이션 정적 분석 모듈
- Find Web Business Logic: 모든 웹 비즈니스 로직 추출 모듈
- BizUnit Generator: 웹 비즈니스 로직을 테스트 하기 위한 BizUnit 자동 생성 모듈

(1) 웹 어플리케이션 정적 분석 모듈: 테스토피아_비즈는 웹 어플리케이션의 소스 디렉토리 정보를 입력받아 화면 파일, 3계층의 설정 파일, 클래스 파일로부터 웹 비즈니스 로직을 추출하기 위해 필요한 정보를 추출한다. Request 객체를 활용하여 웹 페이지 속성, 파라메터, 링크 등 페이지 정보를 분석하도록 개발 하였다. 오픈 라이브러리인 Java Source Reflection[8]을 사용하여 클래스 파일을 분석하고 설정파

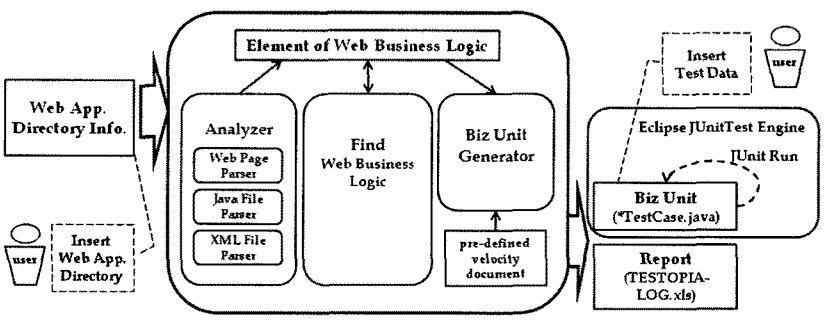


그림 4 테스토피아_비즈의 구조 및 사용 환경

일인 XML 파일 분석을 위해 DOM[9]을 활용하여 개발하였다.

(2) **웹 비즈니스 로직 추출 모듈**: 정적 분석 모듈에서 추출된 정보들을 토대로 웹 어플리케이션에 존재하는 모든 웹 비즈니스 로직을 추출한다. 표 2는 웹 비즈니스로직에 관하여 추출한 주요 정보와 이에 대한 웹어플리케이션 구조에서의 해당 구성요소 생성 과정을 나타낸다. 그림 3(a)의 웹 비즈니스 로직에 대하여 추출한 정보의 예도 함께 나타낸다.

(3) **BizUnit 자동 생성 모듈**: 웹 비즈니스 로직을 테스트 하기위한 BizUnit 코드와 웹 비즈니스 로직에 대한 로그 파일을 자동 생성하는 모듈이다. BizUnit 테스트 코드는 앞서 추출한 웹 비즈니스 정보를 토대로 BizUnit 테스트 코드로 변환하는데, 표 2는 웹 비즈니스 구성 정보에 대해 BizUnit 테스트 코드 구성요소에 어떻게 매핑되는지 나타낸다. 자동 생성되는 테스트 코드의 Template부분을 오픈 소스 템플릿 엔진인 Apache Velocity[10]를 활용하여 개발하였다.

자동 생성된 BizUnit 테스트 코드는 이클립스의 JUnit의 테스트 엔진으로 실행하여 테스트를 수행할 수 있다. 그림 5는 실제 테스토피아_비즈를 실행한 화면이

표 2 BizUnit 생성 과정

추출 정보	그림 2 웹 어플리케이션 구조의 모듈	그림 3(a) 웹 비즈니스 로직 예	BizUnit 구성요소	그림 3(b) BizUnit 예
JSP	모든ⓐJSP 화면 파일 추출	listUser.jsp	**④ 테스트 대상 수행**	setRequestPathInfo("/empgetadminuser.do"); actionPerform();
Event	ⓐJSP 화면의 모든 이벤트 추출. JSP 의 각 이벤트 마다 비즈니스 로직 하나가 생성됨.	getadminuser.do		

Action Form	ⓑConfiguration XML 에서 실제 액션 폼 클래스인 ⓒActionForm 클래스 추출	UserForm	**③테스트 데이터**
			액션 폼 클래스의 화면 데이터 구성 속성에 대한 입력 데이터
				//그림 3(a) 시작화면(listUser.jsp)의 필드("userId") 값("test")
				addRequestParameter("userId", "test");

Action Class	ⓑConfiguration XML 에서 실제 액션 클래스 파일인 ⓓAction 클래스 추출	GetAdminUserAction	**BizUnit 테스트 클래스 및 ② 테스트 메소드 이름**	public class BizUnitGetAdminUserTestCase extends MockStrutsTestCase{ public viod testGetAdminUserAction()

Action_ Mapping	ⓑConfiguration XML 의 화면과 액션 및 액션 폼 클래스 파일 사이의 매핑 정보	struts-configuser.xml	**①프레임워크 환경설정**
			- 프리젠테이션 계층
				private final String webappPath = "./src/webapp";
				setContextDirectory((new File(webappPath)).getCanonicalFile());
				setConfigFile("/config/struts/struts-config-user.xml");

Bean_ Mapping	ⓔApplicationContext 로부터 프레젠테이션 계층과 비즈니스 계층의 상호작용에 관련된 액션 클래스와 빈 클래스 추출	applicationContext-user.xml	**- 프리젠테이션과 매핑되는 비즈니스 계층의 Bean 클래스**	private final String CONFIG_LOCATIONS = "classpath*:/config/spring/applicationContext-*.xml";
				ContextLoader ctxLoader = new ContextLoader();
				context.setInitParameter(ContextLoader.CONFIG_LOCATION_PARAM,CONFIG_LOCATIONS);
				ctxLoader.initWebApplicationContext(context);

BeanClass**	ⓕService Interface, ⓗService Implimentation 로부터 실제 클래스 파일 추출	UserServiceImpl	** Bean, VO, DAO 클래스 정보는 BizUnit 테스트 코드 구성에는 쓰이지 않으나, 해당 비즈니스 로직에 관련된 클래스 파악을 위한 로그 정보에 쓰임	
VOClass**	ⓖVO 로부터 실제 변수 클래스 파일 추출	UserVO		
DAO Class**	ⓙDAO 으로부터 DB 접근 클래스로써 추출	UserDAO		
DB_ Mapping	ⓘ Query XML 에서 퍼시스턴스 계층과 매핑되는 DAO 클래스와 DB 정보 추출	mapping-user-userservice.xml	**- 퍼시스턴스 계층과 과 매핑되는 DAO 및 DB**	

Return	ⓐJSP 와 ⓑConfiguration XML 으로부터 처리된 결과값에 대한 정보 추출	struts-config-user.xml, updateUser.jsp	**⑤테스트 판정**	verifyNoActionErrors ();
				// struts-config-user.xml 의 Forward name 값(success)
				verifyForward("success");
				assertEquals("Error", this.getActualForward(),"/sample/user/updateUser.jsp")

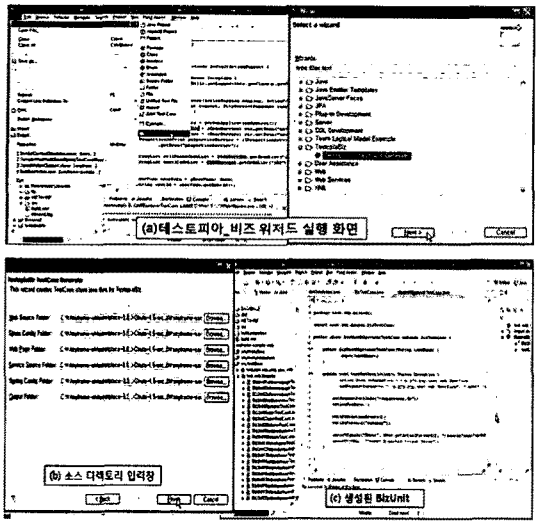


그림 5 이클립스에서 테스토피아_비즈를 통해 테스트 코드를 생성하는 화면

다. 그림 5(a)와 같이 이클립스의 파일 메뉴에서 테스토피아_비즈를 실행하면 사용자 입력을 위한 그림 5(b) 화면이 팝업된다. 이 입력 창에 웹 어플리케이션의 소스 디렉토리 정보와 BizUnit 테스트 코드를 저장할 결과 디렉토리 정보를 입력한다. 테스토피아_비즈는 입력된 웹 어플리케이션에 대한 웹 비즈니스 로직을 추출하여 테스트 코드를 생성하여 결과 디렉토리에 생성된 BizUnit 테스트 코드들을 저장한다. 그림 5(c)의 왼쪽은 BizUnit 테스트 코드 리스트이며 오른쪽은 테스트 코드 하나의 내부 코드 모습이다.

5. 적 용

테스트 대상인 e-Market Place는 스트러츠(Struts), 스프링(Spring), 하이버네이트(Hibernate), 아이바티스(iBatis) 등의 다양한 오픈 소스를 합쳐서 만든 오픈 프레임워크로 Anyframe JAVA의 공식 홈페이지에서 제공하고 있는 샘플프로젝트(Anyframe Samples 3.0.0-jdk 1.5-Struts)이다. 일반 사용자와 관리자가 사용자 등록(User), 상품 구매(Purchase), 상품 판매(Sale) 등과 같은 서비스를 각 메뉴를 통해 이용할 수 있다. 또한 사용자 등록하기 및 수정하기, 목록 보기, 세부 정보 보기 등의 세부 메뉴로 구성되어 있는 웹어플리케이션이다. 대부분의 SI(Software Integration) 프로젝트에서 개발하는 전형적인 웹 어플리케이션 기능을 갖고 있다.

e-Market Place 는 총 36개의 화면과 95개의 클래스와 12개의 설정파일 문서로 구현되어 있다. 테스토피아_비즈를 이용하여 e-Market Place를 분석한 결과 총 40개의 BizUnit테스트 코드를 생성하였다. 그림 6은 e-Market Place의 화면과 화면에서 발생한 이벤트를 가지고 화면 플로우 차트를 그린 것으로 테스토피아_비즈가 찾은 웹 비즈니스 로직이 실선 화살표이고, 화살표 위의 번호는 생성된 BizUnit의 번호이다. 화면 파일인

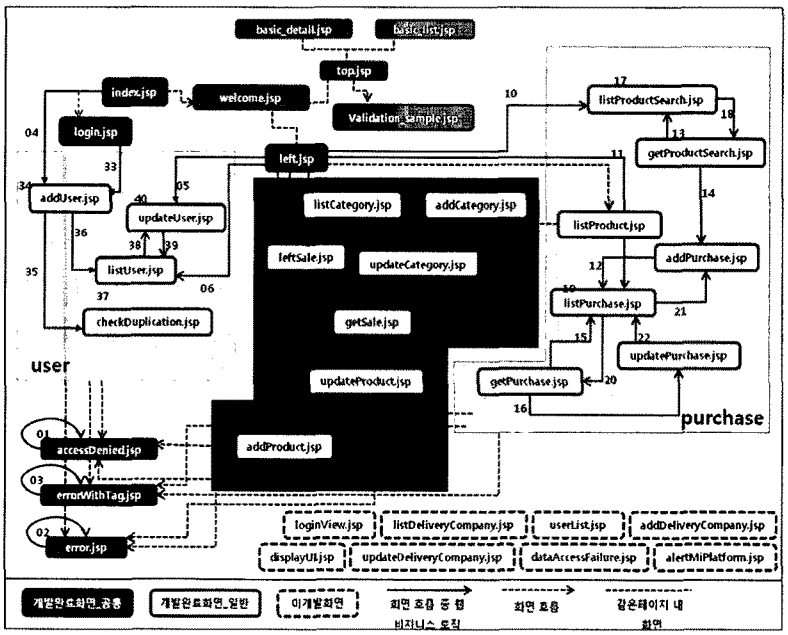


그림 6 e-Market Place에 대한 웹 비즈니스 로직과 해당 BizUnit

경우 총 19개가 비즈유닛에 포함되었으며 포함되지 않은 17개의 화면은 아직 구현이 덜 된 부분으로 각각 처리 클래스가 없는 경우나 자바 스트립스함수로 이벤트를 처리하는 경우로 확인할 수 있었다. 38번 BizUnit이 3장의 그림 3의 웹 비즈니스 로직에 대한 BizUnit에 해당한다.

테스토피아_비즈로 생성된 BizUnit 테스트 코드는 이클립스가 제공하는 JUnit의 테스트 엔진으로 실행이 가능하며 오류를 추적하여 할 수 있도록 개발 되었다. 그림7 은 사용자 로그인과 관련된 BizUnit 테스트 코드를 실행한 화면이다. 테스토피아_비즈로 생성된 테스트 코드는 이클립스의 JUnit으로 실행 할 수 있으며 (a)는 실제 실행하는 모습이다. 테스트 코드에 데이터를 입력하지 않고 실행을 하거나, 잘못된 데이터 값을 입력하고 실행하면 (b)의 그림과 같이 오류가 난 것을 확인할 수 있다. 정상적인 데이터를 입력하고 실행하면 (c)와 같이 테스트가 성공적으로 이루어졌음을 확인할 수 있다.

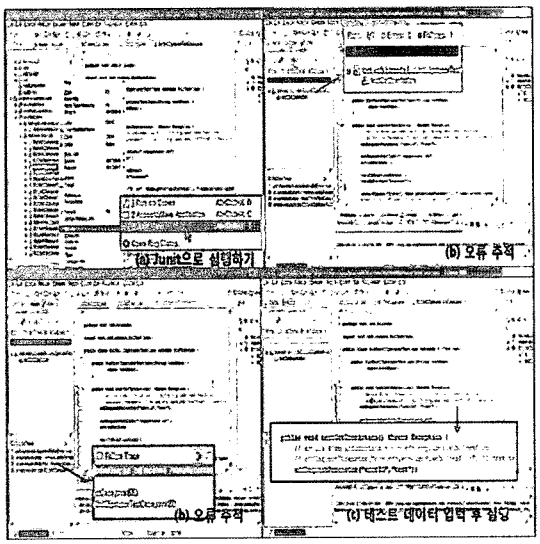


그림 7 BizUnit 실행 및 오류 추적

6. 분 석

e-Market Place 웹 어플리케이션을 대상으로 BizUnit 테스트 코드 생성하고, BizUnit 테스트 코드를 실행함을 통해 JUnit 테스트 코드에 비교하여 BizUnit 테스트 코드 생성의 의미를 분석하면 다음과 같다.

- Web 기반 테스트 도구와의 비교
- 클래스 단위가 아닌 웹 어플리케이션을 구성하는 실행 가능한 개발 단위를 테스트 대상으로 한다.
- 사용자 화면 입 출력 값이 테스트 케이스이다.
- 웹 어플리케이션의 특징인 계층간 상호작용에 의해 발생하는 결함을 테스트할 수 있다.

(1) 웹 어플리케이션 테스트를 위한 타 도구와의 비교

프로그래밍언어나 플랫폼에 따라 다양한 버전으로 제공되는 테스트 프레임워크(Test Framework)를 xUnit이라 하며 JUnit[4], HttpUnit[16], DBUnit[17] 등과 같은 다양한 종류가 있다. 아래 표 3은 이들 테스트 프레임워크를 비교한 것이다.

이들 어느 것도 BizUnit을 테스트 단위로 하고 있지 않으며, 이 가운데 java 를 대상으로 하는 것은 JUnit이다. 웹어플리케이션 테스트를 위해 현재 상용적으로 사용하고 있는 대표적인 JUnit 테스트 도구로는 IMB Rational사의 CodePro[11], Agita사의 AgitaOne[12], ParaSoft사의 JTest[13]가 있다. 이들은 모두 JUnit 테스트 코드를 자동 생성하며, 개발환경인 Eclipse에 플러그인 도구로 지원되며 자동 생성된 테스트 코드를 실행하여 테스트 판정을 할 수 있도록 한다. 이 가운데 JUnit 테스트코드 생성 도구로써 가장 대표적인AgitarOne에 의한 본 논문에서 제시하는 BizUnit 생성하는 테스토피아_비즈 테스트 도구의 비교를 표 4로 요약하였다.

테스토피아_비즈로 생성한 BizUnit 테스트 코드는 실제 의미있는 의미있는 웹비지니스 로직 단위로 테스팅한다. 또한 객체의 의존하여 임의의 테스트 데이터를 생성하는 AgitarOne과 달리 테스트데이터를 직접 생성하

표 3 웹어플리케이션 테스트 도구 비교

비교 항목	JUnit	HttpUnit	DBUnit
테스트 대상	· 클래스의 메소드 단위 테스트 · 외부 호출 함수나 라이브러리 객체에 대한 스텁 생성	· 서버 측의 자바 코드, 서블릿 EJB, 태그 라이브러리 및 필터 등의 단위 테스트 · 컨테이너와 상호작용을 위한 통합테스트	· DataBase의 DAO 클래스 · DataBase에 연결하여 단위테스트
결함 발견 종류	· 메소드 내부에 존재하는 결함	· redirector 서블릿에 요청후 응답된 내용과 비교 후 웹 계층에 존재하는 결함	· DataBase 내부의 존재하는 결함 및 DataBase 내의 Data 오류

표 4 JUnit 자동화도구와 BizUnit 자동화도구 비교

비교 항목	JUnit 테스트 코드(도구: AgitarOne)	BizUnit 테스트 코드(도구: Testopia_biz)
테스트 대상	· 클래스의 메소드 단위 테스트 · 외부 호출 함수나 라이브러리 객체에 대한 스텁 생성	· 웹 비즈니스 로직 단위 테스트 · 스텁 생성하지 않고, 실제 객체를 사용
테스트 데이터 생성	· 메소드의 입력인 파라메터 객체에 대해 형에 의존한 임의의 값 생성	· 웹 비즈니스 로직의 입력인 액션 폼에 대한 정보를 주며, 직접 데이터를 생성하지는 않음.
테스트 판정	· 메소드의 출력인 반환 객체나 메소드 내부에 사용된 객체에 대한 판정문 생성	· 웹 비즈니스 로직의 올바른 수행완료에 대한 판정문 생성
결함 발견 종류	· 메소드 내부에 존재하는 결함	· 웹 비즈니스 로직에 포함되는 각 구성요소들에 대한 결함 · 각 구성요소들의 상호작용이나, 프레임워크와의 상호작용에 대한 결함

지 않지만 실제 사용자 입출력 화면 정보를 제공함으로서 정보에 맞춰서 다양한 값을 테스트 해 볼 수 있다. 또한 테스토파아_비즈로 생성한 BizUnit의 경우, 스텁 생성 없이 실제 객체를 사용하기 때문에 각 구성요소들의 결함뿐 아니라 프레임워크와의 상호작용까지 테스트 할 수 있다. 이 특징을 좀 더 상세히 분석하면 아래와 같다.

(2) 클래스 단위가 아닌 웹 어플리케이션을 구성하는 실행 가능한 개발 단위를 테스트 대상으로 한다.

JUnit테스트 코드를 생성하는 자동화 도구들은 e-Market Place에 대하여 각각의 클래스에 대한 JUnit 테스트 코드를 생성한다. 그런데 앞서 웹 어플리케이션의 특징에서 기술하였듯이 클래스 각각의 하나 보다는 이들이 프레임워크와 상호작용하는 부분이 많은 비중을 차지하기 때문에 클래스 하나에 대한 JUnit 테스트 코드를 단독 실행하게 하기 위하여는 스텁을 생성할 수 밖에 없다.

BizUnit은 프레임영역의 서블릿이나 컨텍스트를 그대로 사용하기 때문에 설정 파일까지 포함되어 스텁이 따로 필요없다. 예를들어, 그림 8은 그림 3(a)의 listUser 화면에서 UpdateUser 화면으로 empgetadminuser.do 이벤트가 발생한 것에 대한 웹 비즈니스 로직에 대해 실제 파일들과 호출, 반환 흐름을 표시한 것이다. 그림 4의 예제의 GetAdminUserAction, UserServiceImpl, UserDAO에 대해 JUnit 기반 테스트 자동화 도구가 자동으로 생성한 코드를 실행한 경우 그림 8의 (a), (b), (c)와 같이 밝은 실행 영역과 어두운 스텁 영역으로 나눠진다. 그러나 BizUnit은 그림 3(b)를 실행시킨 것으로 (d)와 같이 스텁없이 관련된 파일들이 실행된다.

(3) 사용자 화면 입출력 값이 테스트 케이스이다.

테스트케이스의 의미 있는 입출력 데이터 값을 자동생성하는 것은 매우 어렵다. 앞서 2장에서 언급했듯이 테스트대상 코드의 정적 분석을 통하여 테스트케이스를 자동생성을 일부 실현하고 있지만 쓸모없는 데이터 값

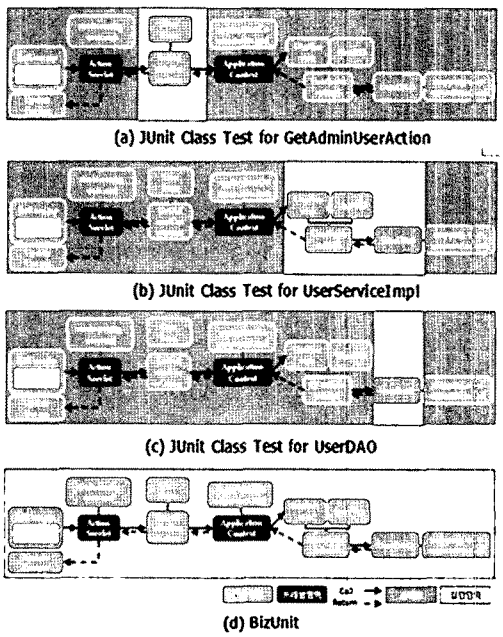


그림 8 JUnit 클래스 테스트 코드(a, b, c)와 BizUnit(d)의 스텁 사용 비교

이 함께 생성되는 문제점이 있다. Junit 테스트 코드 자동생성 도구들도 테스트케이스를 자동생성한다. 즉, 테스트 대상 객체 클래스를 포함하여 사용되는 모든 변수 객체에 대해 임의의 값을 가지는 인스턴스를 생성한다. 이 때에 생성되는 테스트 데이터는 객체 형에 의존하지만 비교적 종류별로 다양한 값을 생성한다. 그러나 테스트대상 코드의 컴파일레이션 타임에 정적분석된 정보를 활용함으로 실제 의미있는 테스트 입력 데이터 값과는 거리가 있다. 한편, 웹 어플리케이션의 경우 프레임워크가 제공하는 객체는 라이브러리 형태로 존재하기 때문에 이에 대한 객체를 생성하지 않는다. 즉, 프레임워크의 객체가 메소드의 파라메터로 사용된 경우 스텁으로

대체된 테스트를 수행하게 되며 결과적으로 각 계층의 모든 클래스에 대해 의미 없는 테스트가 수행되게 된다.

BizUnit의 테스트 대상이 되는 단위는 사용자 화면에서 출발하여 사용자 화면으로 돌아오는 실행 흐름을 갖는 비즈니스 로직이다. 따라서 사용자 화면 입출력이 테스트대상이 된다. 테스토피아_비즈는 웹 비즈니스 로직에 의해 테스트코드를 생성한다. 동등분할값(Equivalance Partition), 경계값(Boundary Value Analysis), Pair-Wise 등 다양한 블랙박스 테스트 도구를 활용하여 결정한 테스트데이터를 입력할 수 있다. 이는 의미있는 데이터를 입력하여 테스트코드를 실행 할 수 있도록 한다.

(4) 웹 어플리케이션의 특징인 계층간 상호작용에 의해 발생하는 결함을 테스트할 수 있다.

웹 어플리케이션의 각 구성요소들은 프레임워크가 참조할 수 있는 특정한 형태로 단순화되어 개발되며 이 구성요소들은 프레임워크의 제어받는다. 그러므로 웹 어플리케이션의 테스트는 프레임워크의 제어여부와 개발 구성요소들 간의 상호작용에 보다 집중되어야 할 것이다. 웹 어플리케이션에서 발생가능한 결함을 그림 2의 웹 어플리케이션 구성에서의 위치에 따른 결함 종류를 기준[14,15]으로 표 4와 같이 정리할 수 있다.

표 4 웹 어플리케이션 결함

결함 위치	결함 종류	판정	
		JUnit	Biz Unit
	JV1 자바 스크립트 함수 에러	N	Y*
	JV2 화면 필드 값 저장 에러	N	Y
	JV3 이벤트 발생 에러	N	Y
	XV1 이벤트 처리 정보 에러	N	Y
	XV2 중복 처리 에러	N	Y
	XV3 처리 속성 에러	N	Y
	CV1 화면 필드 값과 액션 폼 변수의 타입 불일치	N	Y
	CV2 액션 클래스 함수 로직 에러	Y	Y
	CV3 액션 클래스 외부 함수 호출 에러	N	Y
	XB1 자바빈 정의 에러	N	Y
	XB2 중복 처리 에러	N	Y
	XB3 관계 속성 에러	N	Y
	CB1 인터페이스클래스 정의 에러	Y	Y
	CB2 액션 폼의 변수와의 타입 불일치	N	Y
	CB4 비즈니스 클래스 함수 로직 에러	Y	Y
	CB5 비즈니스 클래스 외부 함수 호출 에러	N	Y
	XP1 DB 테이블 정의 에러	N	Y
	XP2 쿼리문 에러	N	Y
	XP3 매핑 정의 에러	N	Y
	CP1 DB접근 클래스 함수 로직 에러	Y	Y
	CP2 쿼리 호출 메소드 에러	Y	Y

J(Jsp), X(Xml), C(Class), V(Presentation Layer), B(Business Layer), P(Persistence Layer)
N: 발견못함, Y: 발견함, Y*: BizUnit에 포함된 자바 스크립트 경우

다계층 웹에서 발생할 수 있는 결함은 클라이언트(사용자화면)에서의 사용자 입력 에러, 서버(웹 어플리케이션)와 클라이언트의 상호작용 에러, 서버 내부 에러, 구성요소사이의 데이터 불일치 에러, DB관련 에러 등으로 분류할 수 있다[15]. 표 4는 이 분류에 의해 웹 어플리케이션의 구성요소별로 결함을 구체화한 것이다. 예를 들어, 서버와 클라이언트의 상호작용은 서블릿이 관여하는 것으로 서블릿이 참조 하는 Ⓑ에서 발생하는 XV1이벤트 처리 정보 에러가 발생할 수 있다. 그림 3의 (a)에서 언급한 웹 비즈니스 로직처럼 사용자가 사용자 등록을 진행하면 JSP 화면과 Action 클래스가 액션매핑 라이브러리를 통해 연결정보에 해당하는 ConfigurationXML 파일을 찾는다. ConfigurationXML 파일은 사용자 등록 이벤트인 "empadduser"를 처리하는 클래스인 AddUserAction 클래스를 찾아 연결을 해야 한다. 판정을 위한 실험에서는 Ⓑ의 이벤트 처리 정보 에러를 발생하기 위하여 AddUserViewAction 클래스와 연결을 하였다. AgitaOne으로 테스트코드를 자동생성 하였고, BizUnit 테스트 코드는 본 논문에서 개발한 테스토피아_비즈로 자동생성 하였다. 생성된 테스트 코드들을 비교함으로써 이벤트 처리 에러를 찾을 수 있는 지 비교하였다.

먼저 사용자의 등록 액션이 발생과 관련된 클래스는 GetAdminUerAction, UserVO, UserServiceImpl, UserServiceImpl, UserDAO, UserService 총 6개와 관련이 되어 있다. AgitaOne의 자동 생성한 테스트 코드는 클래스를 단위로 생성하기 때문에 GetAdminUerAction, UserVO, UserServiceImpl만 자동 생성했다. 폼클래스(UserForm)와 객체 클래스는 (UserDAO)는 내부에 accessor()와 mutator() 메소드만 존재하고 객체 변수 정의 되어있기 때문에 테스트 코드를 만들지 않았다. UserService 클래스의 경우 인터페이스 클래스이기 때문에 테스트코드를 생성하지 않았다. 아래 그림 9는 AgitarOne으로 생성한 Action 클래스의 테스트코드의 일부이다. 코드를 살펴 보면 에서 제공하는 스텁의 일종인 Mockingbird 스텁을 통해서 4개의 빈 객체를 생성한 것을 볼수 있다. 빈 객체를 생성해서 메소드를 콜하고 그 결과에 대해서 스텁으로 생성한 빈 객체와 비교하기 때문에 실험에서는 PASS가 된다. 결과적으로 Action에 대해 코드를 정확히 테스트했다고 볼 수 없다.

그림 10은 테스토피아_비즈로 생성된 테스트코드이다. BizUnit38GetAdminUser 테스트 코드를 살펴보면 테스트 대상에 대한 객체 타입 및 정보를 제공함을 볼 수 있다. 또한 프레임워크에서 사용하는 객체와 설정파일 문서와 웹 페이지 문서까지 테스트 대상으로 포함시킨다. 액션매핑 라이브러리를 통해 상호 연결정보에 해당

(a)For Persistence Layer Class

```
package com.sds.emp.sale.web;

import com.agitar.lib.junit.AgitarTestCase;[]
public class ProductDAOAgitarTest extends AgitarTestCase {
public Class getTargetClass()  {
        return ProductDAO.class;
    }
public void testAddProductThrowsErrorWithAggressiveMocks() throws Throwable {
        ProductDAO productDAO = (ProductDAO) Mockingbird.getProxyObject(ProductDAO.class, true);
        Mockingbird.enterRecordingMode();
        Mockingbird.replaceObjectForRecording(Error.class, "<init>(java.lang.String)", Mockingbird.getProxyObject(Error.class));
        Mockingbird.enterTestMode(ProductDAO.class);
        try {
            productDAO.addProduct(null);
            fail("Expected Error to be thrown");
        } catch (Error ex) {
            assertTrue("Test call resulted in expected outcome", true);
        }
}
//중략//
}
```

(b) For Presentation Layer Class

```
package com.sds.emp.sale.web;

import com.agitar.lib.junit.AgitarTestCase;
import com.agitar.lib.mockingbird.Mockingbird;
   //코드 중략//
public class AddProductActionAgitarTest extends AgitarTestCase {
   public Class getTargetClass()  {
        return AddProductAction.class;
   }
   public void testProcessWithAggressiveMocks() throws Throwable {
        AddProductAction addProductAction = (AddProductAction) Mockingbird.getProxyObject(AddProductAction.class, true);
        ActionMapping actionMapping = (ActionMapping)   Mockingbird.getProxyObject(ActionMapping.class);
        ProductForm productForm = (ProductForm) Mockingbird.getProxyObject(ProductForm.class);
        HttpServletRequest httpServletRequest = (HttpServletRequest) Mockingbird.getProxyObject(HttpServletRequest.class);
       //코드 중략//
        Mockingbird.enterRecordingMode();
        Mockingbird.setReturnValue(webApplicationContext.getBean("productService"), productService);
        Mockingbird.setReturnValue(httpServletRequest.getSession(), httpSession);
        Mockingbird.setReturnValue(httpSession.getAttribute("userId"), null);
         //코드 중략//
        Mockingbird.enterTestMode(AddProductAction.class);
        ActionForward result = addProductAction.process(actionMapping, productForm, httpServletRequest, null);
        assertNull("result", result);
    }
```

그림 9 AgitarOne으로 자동생성된 테스트 코드

```
public abstract class BizTestCase extends MockStrutsTestCase {

 public void setUp() throws Exception {
  super.setUp();

  setInitParameter("validating", "false");
  setContextDirectory((new File(webappPath)).getCanonicalFile());
  setConfigFile("/config/struts/struts-config.xml");
  setConfigFile("/config/struts/struts-config-demo.xml");
  setConfigFile("/config/struts/struts-config-purchase.xml");
  setConfigFile("/config/struts/struts-config-sale.xml");
  setConfigFile("/config/struts/struts-config-user.xml");

 }
}
```

```
package test.web.bizcode;
import test.web.common.BizTestCase;
public class BizUnit38getadminuserTestCase extends BizTestCase {
 public BizUnit40getadminuserTestCase(String testName) {
          super(testName);
 }
 public void testGetAdminUserAction() throws Exception {
  // action form information = com.sds.emp.user.web.UserForm
  // addRequestParameter("com.sds.emp.user.web.UserForm", "Insert test data");

  setRequestPathInfo("/empgetadminuser.do");
  actionPerform();

  verifyNoActionErrors();
  verifyForward("success");

  assertEquals("Error", this.getActualForward(), "/sample/user/updateUser.jsp");
  //assertNotNull("Insert Expected Output Value");
 }

}
```

그림 10 테스토피아_비즈로 자동생성된 테스트 코드

하는 ConfigurationXML 파일의 결함을 찾아준다.

이와 동일한 방법으로 표 4의 결함 리스트에 따라 *e-Market Place*에 그 종류별로 결함을 삽입하여 자동 생성된 JUnit 테스트 코드와 BizUnit 테스트 코드가 결함을 찾을 수 있는지 실험하였고, 그 결과가 표 4의 판정이다. 본 실험 결과로 알 수 있듯이 웹 어플리케이션의 경우, 메소드 단위의 테스트보다 각 구성요소들이나 프레임워크와의 상호작용을 테스트하는 것이 더 필요한데, 웹 비즈니스 로직 단위인 BizUnit 테스트코드로 테스트를 수행하는 것이 클래스 메소드 단위의 JUnit 테스트 코드보다 실제로 많은 결함들을 발견할 수 있었다.

7. 결론 및 향후 과제

프레임워크기반 웹 어플리케이션 개발은 라이브러리 형태로 프레임워크 객체를 제공하여 개발하고자 하는 기능을 보다 쉽게 개발할 수 있도록 도와준다. 이는 웹 어플리케이션의 각 구성요소들이 프레임워크의 제어를 받으며 각 프레임워크와 상호작용하는 부분의 비중이 커지는 것을 의미한다. 본 논문에서는 웹 어플리케이션을 올바르게 테스트할 수 있도록 BizUnit 테스트 코드를 제안하였다. BizUnit은 하나의 웹 비즈니스 로직을 실행하여 테스트 할 수 있도록 하는 JUnit 테스트 코드이다. 웹 비즈니스 로직이란 웹 어플리케이션 테스트를 위해 본 논문에서 제안한 최소 단위로써, 사용자 화면에 해당하는JSP 화면에서 시작하여 웹 프레임워크에서 제공하는 라이브러리 계층과 함께 연동하며, 프레젠테이션 계층과 비즈니스 계층, 퍼시스턴스 계층을 거쳐 데이터베이스를 경유하고 다시 JSP화면으로 돌아오는 실행 흐름이다.

기존의 자바 프로그램을 위한 JUnit는 웹 어플리케이션의 모듈을 각각 테스트하는 단위테스트(즉, 클래스 단위테스트)이며, 따라서 이를 자동화한 기존의 도구로는 웹 어플리케이션의 모든 결함들을 찾아내기 어렵다. 본 논문에서 개발한 BizUnit 테스트코드 자동생성 도구인 테스토피아_비즈는 삼성 애니프레임 구조하에 개발 하는 웹 어플리케이션과 그 프레임워크와의 상호작용에서 발생할 수 있는 종류의 결함을 찾아낼 수 있음을 사례연구를 통하여 확인할 수 있었다. 또한 웹 비즈니스 로직의 시작점이 사용자 화면이기 때문에 사용자 화면에서 입력하는 그대로를 BizUnit 테스트 데이터로 활용할 수 있었다. 이것은 일반 자바 어플리케이션에 대해 JUnit 기반의 단위 테스트 방식을 활용한 것에 대해 웹 어플리케이션을 위한 BizUnit 기반의 테스트 수행을 제시함으로써 궁극적으로 TDD(Test Driven Development)에서도 활용될 수 있음을 뜻한다.

본 논문은 웹 어플리케이션 테스트를 위한 BizUnit 테스트 코드의 최초 제안와 그 자동화에 큰 의의가 있다. 향후 본 BizUnit 테스트 코드를 산업에서의 프로젝트 개발에 실제 적용할 계획이며, 이를 통해 웹 어플리케이션 테스트의 품질 향상에서의 기여를 분석할 예정이다.

참 고 문 헌

[1] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann, "Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2," Wiley & Sons, New York, 2000.

[2] John Arthur, Shiva Azadegan, "Spring Framework for Rapid Open Source J2EE Web Application Development: A Case Study," pp. 90-95, 2005.

[3] Gerard Meszaros, "xUnit Test Patterns," Addison-Wesley, Upper Saddle NJ, 2007.

[4] JUnit, http://www.junit.org

[5] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," In *Proc. of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI)*, 2005.

[6] Samsung SDS, "AnyFrame," http://www.anyframe-java.org/

[7] StrutsTestCase, http://strutstestcase.sourceforge.net/

[8] Java Source Reflection, http://ws.apache.org/jaxme/js/jparser.html

[9] W3C Document Object Model, http://www.w3.org/DOM

[10] *Apache Velocity*, http://velocity.apache.org/

[11] CodePro, http://download.instantiations.com/CodeProDoc/continuous/latest/docs/doc/features/junit/test_case_generation.html

[12] AgitarOne, http://www.agitar.com/solutions/products/agitarone.html

[13] JTest, http://www.parasoft.com/jsp/products/home.jsp?product=Jtest

[14] Filippo Ricca, Paolo Tonella, "Analysis and Testing of Web Applications," Proceedings of the 23rd IEEE International Conference on Software Engineering, *IEEE Computer Society*, pp.25-34, 2001, Toronto, Ontario.

[15] A. Marchetto, F. Ricca, P. Tonella, "Empirical Validation of a Web Fault Taxonomy and its usage for Fault Seeding," In the IEEE Int. Symposium on Web Site Evolution, 2007.

[16] HttpUnit test, http://httpunit.sourceforge.net/

[17] DBUnit test, http://dbunit.org/

이 은 영

2002년~2007년 덕성여대 컴퓨터과학과 학사. 2009년~현재 이화여대 컴퓨터공학과 석사과정. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 웹 어플리케이션 테스트

최 병 주

1979년~1983년 이화여대 수학과 학사 1986년~1987년 Purdue Univ. Computer Science 석사. 1987년~1990년 Purdue Univ. Computer Science 박사. 1991년~1992년 삼성종합기술원 1992년~1995년 용인대 전산통계학과 조교수 1995년~현재 이화여대 컴퓨터학과 교수. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 소프트웨어 및 데이터 품질 측정

송 화 정

2003년~2007년 이화여대 컴퓨터과학과 학사. 2007년~2009년 현재 이화여대 컴퓨터공학과 석사과정. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 웹 어플리케이션 테스트

황 상 철

1992년~1999년 경희대 산업공학과 학사. 1999년~2001년 경희대 산업공학과 석사. 2001년~현재 삼성 SDS SW Eng 팀 근무. 관심분야는 J2EE 개발 방법론 및 아키텍처, 소프트웨어 테스트 자동화