TDD기반의 모바일 단위 테스트 방법 및 개발에 관한 연구

채현철*, 황선명*, 김철홍**
*대전대학교 컴퓨터공학과
**한국전자통신연구원, 임베디드 S/W 연구단
e-mail:chaehc@gmail.com

A Study for Mobile Unit Test Method & Development based-on T.D.D

Hyeon-Cheol Chae*, Sun-Myung Hwang*
Chul-Hong Kim**
*Dept of Computer Engineering, DaeJeon University
**Embedded S/W Research Division, ETRI

요 약

모바일의 소프트웨어 테스팅은 매우 중요하다. 그러나 모바일 특성상 일반 어플리케이션의 테스팅과 다른 점이 있으며, 짧은 생명주기 등으로 빠른 출시를 목적으로 테스팅을 소홀히 하는 경우가 있다. 이렇게 테스팅을 소홀히 함으로써 추가적인 유지보수등의 비용으로 기업의 손실로 직결될 수 밖에 없다. 하지만 현재 모바일의 단위 테스트 방법 및 테스트 개발 방법이 미미한 실정이다. 본 논문에서는 TDD방법을 적용하여 모바일의 테스트 방법 및 개발 절차에 대하여 제시하고자 한다.

1. 서론

오늘날 모바일 어플리케이션의 중요성이 증대 되고 수요가 점차 증가함에 따라 사회 전반에서 모바일 소프트웨어의 중요성은 매우 중요한 위치를 차지하고 있다. 모바일 소프트웨어의 중요성이 증가함에 따라 소프트웨어 테스트 또한 기업의 손익을 결정하는 중요한 요소로 자리 잡게 되었다. 하지만 모바일의 테스트 방법 및 테스트 도구는 상당히 미미한 실정이다.

따라서, 본 논문에서는 시중에 배포된 테스팅 도구들을 분석하고 다중 모바일 플랫폼 환경에 적용 할 수 있는 단위 테스트 절차 및 테스트 개발 방법을 제안 하고자 한다.

2. 관련연구

2.1 단위 테스트

테스트의 목적은 모듈이 제대로 구현되었는지 시험하는 것으로 모듈이 어떤 작업을 수행하는지를 나타낸 설계 명세를 근거로 외부 관점의 테스트를 할 수도 있고 프로그램의 내부를 살펴서 논리 흐름을 체크하는 화이트 박스 형태를 취할 수도 있다. 벤처기업의 경우 테스트는 대부분 비공식적으로 다루어 프로그램 개발자에게 일임하는 경우가 많다. 프로젝트의 공식적인 절차로 다룰 경우는 테스트 계획에 무엇을 테스트하며, 어떤 결과를 예상하고 있는지 작성하여 설계자와 프로그래머가 사인한 후 테스트 케이스를 수행시킨다. 테스트의 완료 시점은 코드 커버리지를 사용하는 경우가 많다. 원시코드 문장이 한번 이상, 또는 분기에 대한 조건이 모두 수행되었는지 확인하여 테스트가 완료되었는지 확인한다. 아래 <표 1>은 모바일과 데스크탑의 테스트를 비교 분석 한 것이다.[2]

<표 1> 모바일/데스크탑 어플리케이션 테스트 비교

요소	모바일	데스크탑
테스트 기록	GUI 이벤트 기록을 위한 기능을 제공하지 못해, 사용자 인터랙션에 대한 에뮬레이션이 불가능	GUI 이벤트 기록을 위한 java.awt.Robot 제공
배치 (deployment) 자동화	디바이스 상에 배치와 테스트의 지루한 수작업	배치가 완전히 자동화 되어 있고, 테스트도 비교적 자동되어 용이함
테스트 환경 차이점	디바이스 간에 차이점이 있고(JVM의 차이, 메모리와 가용 자원의 차이 등), 가능한 모든 디바이스 상에서 테스트가 요구됨	개발과 배치/테스트가 동일하고, 특정 OS에 의존하는 경우가 거의 없음
API	API가 표준화가 아직 미성숙	API가 성숙되어 있고, 다양한 벤더로부터 JVM 이식가능

2.2 T.D.D

최근 몇 년 동안 새로운 소프트웨어 개발 기법인 XP(eXtreme Programming) 기법이 주목을 받아왔다. XP 기법에서는 테스트 주도 개발을 기본으로 짝 프로그래밍, 리팩토링 등의 기법을 통해 생산성을 향상,개발이 가능하

본 연구는 정보통신부 및 정보통신연구진흥원의 IT신성장동력 핵심기술개발사업의 일환으로 수행하였음. [2007-S032-01, 다중플랫폼지원 모바일 응용 S/W 개발환경 기술 개발]

도록 하고 있다. 이 중에 TDD(Test Driven Development : 테스트 주도 개발)는 새로운 코드를 추가하기 전에 먼저 테스트를 작성하고, 이러한 테스트를 바탕으로 프로그램을 작성하게 된다. 또한 과감한 리팩토링 과정을 통해 지속적인 디자인을 가능하게 한다.

그러나 전반적으로 모바일 업계에서는 TDD를 거의 쓰고 있지 않다. 그 이유는 첫째, 생명주기가 매우 짧은 모바일 어플리케이션에서 하나하나 코드를 테스트 해가며 개발할 시간이나 인력이 매우 부족하다. 둘째로, TDD는 기존 모바일 GUI테스트와 달리 처음부터 테스트 코드를 작성하여야 하고 TDD방법은 개인적인 수련이 필요 하기 때문에 지금에 와서 새롭게 배운다는 것은 개발자 입장으로서는 상당히 귀찮은 일이기 때문이다.

TDD방법을 적용할 경우, 초기의 결함을 매우 줄일 수 있을 뿐만 아니라 유지보수비용도 상당히 낮출 수 있다. TDD라는 단어 자체가 테스트를 포함하기 때문이다.

2.3 JUnit

JUnit은 J2SE 어플리케이션을 위한 테스트 프레임워크로 이클립스에 포함되어 있다. 그리고 현재 가장 널리 사용되는 테스트 도구로서 이해가 쉽고, 구현이 용이할 뿐만 아니라 오픈 소스로 테스트 주도형 개발을 지원한다.

특징은 xUnit이라고 하는 테스트 프레임워크를 Java로 구현한 것으로 JDK와 맞물려서 assertXXX 메소드를 이용하여 테스트 코드를 작성한다. 또한 테스트 오류 결과 확인 및 최적 코드 유추가 가능하고 테스트 및 테스트 클래스를 반환한다.이러한 구조는 테스트 방법 및 클래스의 이력을 추적할 수 있는 이점이 있다.

2.4 Mobile JUnit

Mobile JUnit은 Sony Ericsson이 제공하는 JUnit을 확장한 단위 테스트 도구이다. J2ME CLDC 플랫폼에서 단위 테스트를 할 수 있도록 하고, 간단한 코드 커버리지 도구를 제공한다. Sony Ericsson 모바일 폰 상에서만 테스트가 가능하다.

특징은 J2SE의 reflection 기능을 대신하여 helper class를 사용하여 테스트 메소드를 호출하고 JUnit 테스트 케이스와 거의 동일하며, JUnit에서 사용하는 모든 assertion들이 사용 가능하다. JUnit과 다른 점은 테스트 fixture를 초기화하는 방법이다.

테스트 클래스 구현에 setUp과 tearDown 메소드를 정의한다.

3. 단위 테스트 절차

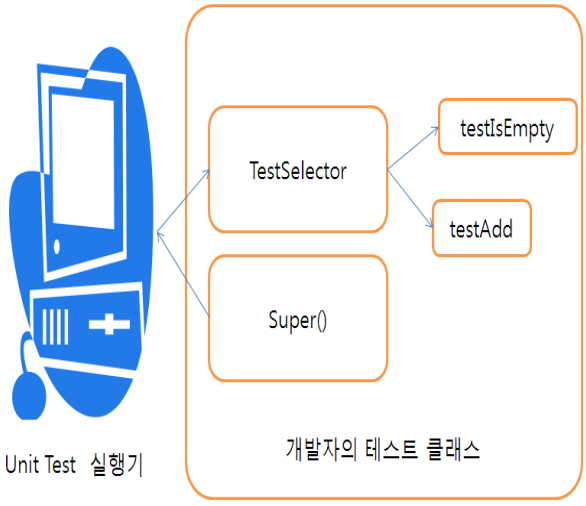
3.1 모바일 테스트 프레임워크의 구성

모바일 환경에서는 리플랙션 기능이 없기 때문에 일반적인 JUnit과 같이 리플랙션을 활용한 테스트 프레임워크를 적용하기 어려운 환경이다. 또한 모바일 개발 환경에서는 에뮬레이터와 연동되어 동작할 수 있는 기능이 필요하다. 이를 위해 별도의 모바일 환경용 테스트 프레임워크가 필요하다.

기존의 모바일 환경에서의 테스트 프레임워크 중에서 기능상으로는 Mobile JUnit이 가장 뛰어나지만 현재 지원하는 플랫폼은 소니에릭슨의 휴대폰과 에뮬레이터만 지원하므로 현재 연구중인 모바일 환경에서는 적용할 수 없다. 그러므로 오픈 소스 테스트 프레임워크 중에 기능이 뛰어난 JMUnit을 기반으로 개발할 예정이다.

테스트 프레임워크에서의 리플랙션 기능은 테스트 개발자에게 테스트 클래스의 손쉬운 추가 및 테스트 메소드 추가를 가능하게 한다. 리플랙션 기능이 없다면 테스트 결과를 수집하고 테스트를 수행하는데 있어서 테스트를 추가할 때마다 테스트 개발자에게 일일이 관련된 테스트를 수정해야하는 번거로움을 겪게 만든다.

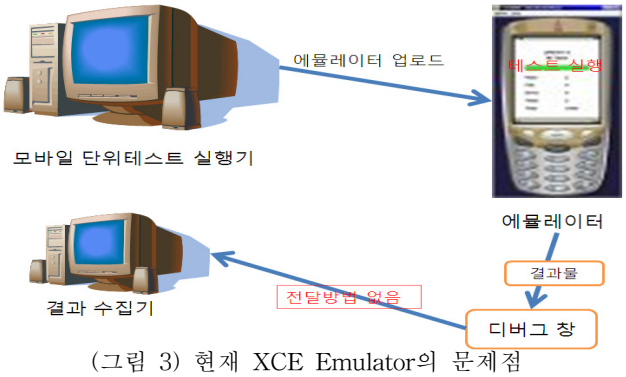
현재의 JMUnit에서는 리플랙션을 해결하기 위해서 다음과 같이 생성자에 super(테스트 개수, “테스트클래스이름”)와 같은 방식을 사용한다. 본 연구에서도 1차적으로 이와 같은 방식을 사용하여 모바일에서 리플랙션 기능을 구현한다.



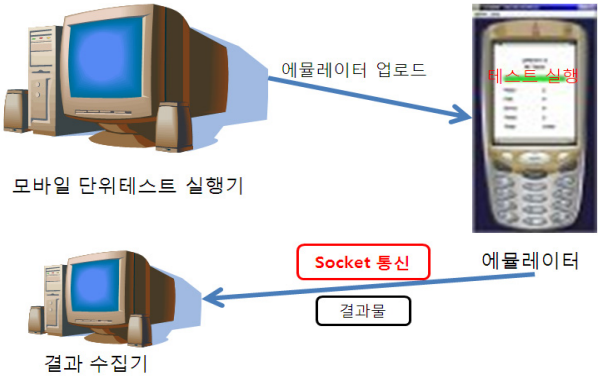
(그림 1) 테스트 선택기의 리플랙션 기능 구현

먼저 테스트 클래스는 생성자를 통해서 테스트 클래스에 포함되어 있는 테스트 케이스의 개수와 클래스 이름과 같은 정보를 알려준다. 그렇게 하면 단위 테스트 실행기에서는 테스트 선택기와 같은 test(int testNumber) 메소드를 자동으로 호출하여 테스트들을 수행하게 된다. 해당 결과는 단위 테스트 실행기가 수집하여 테스트 결과를 보고하게 된다. 테스트 선택기는 test를 switch 문을 사용하여 테스트의 번호마다 해당 테스트를 호출하도록 한다.

이와 같은 테스트를 모바일 응용 SW 환경에서 수행할 경우 위 (그림 2)과 같이 일반적으로 사용되는 XCE 에뮬레이터에서는 폐쇄된 환경으로 인해 테스트 결과를 수집하기 어려운 측면이 있다. 이를 위해 아래 (그림 3)와 같이 소켓(Socket) 통신을 사용하여 테스트 결과를 에뮬레이터 외부의 개발환경에 전달하는 방식으로 구성한다.



(그림 3) 현재 XCE Emulator의 문제점



(그림 2) 소켓 통신을 이용한 결과 출력

4. 단위 테스트 개발 절차

4.1 단위 테스트 생성 절차

1) 테스트 메소드 개발

단위 테스트의 메소드 작성 요령은 아래 <표 2>의 예와 같이 작성한다.

<표 2> 테스트 메소드 예시

```
public void testIsEmpty() throws AssertionFailedException
{
        Foo foo = new Foo();
        assertTrue(foo.isEmpty());
        foo.add(new Object());
        assertFalse(foo.isEmpty());
}
```

- Foo: 테스트 대상 클래스
- isEmpty(): 테스트 대상 메소드

테스트 메소드의 이름은 JUnit과 달리 임의의 이름을 넣어도 상관없지만 기존의 관습대로 앞에 test를 붙여주는 것이 일반적이다. 테스트 코드 내부는 테스트를 수행하기 위한 테스트 준비 코드와 실제 테스트를 체크하기 위한 판별 부분으로 되어 있다. 판별은 assertTrue, assertFalse, assertEqual등의 assert 류의 메소드를 사용한다. 이러한 판별 결과는 테스트에 기록되어 테스트 프레임워크에서 테스트 결과를 보고하는데 사용된다.

2) 생성자 메소드 개발

생성자 메소드의 작성 요령은 아래 <표 3>과 같다.

<표 3> 생성자 메소드 예시

```
public FooTest()  {
        super(3, "FooTest");
}
```

super(테스트 개수, “테스트클래스이름”)를 넣어준다. 예제에서 3의 경우는 수행하여야할 테스트 케이스가 3개가 있다는 의미이고 실제로 테스트 수행은 testIsEmpty(), testAdd(3, 5, 8), testAdd(2, 1, 3)의 3개의 테스트를 수행하여야 한다.

3) 테스트 선택기 개발

테스트 선택기는 프레임워크에서는 test라는 메소드 이름으로 되어 있다.

<표 4> test 메소드 예시

```
public void test(int testNumber) throws Throwable{
        switch(testNumber)
        {
                case 0: ....
        }
}
```

위 <표 4>의 구조는 동일한 패턴이므로 이 패턴을 그대로 사용도록 하며 switch 문 내에 추가되는 테스트 메소드를 추가한다. 이때 테스트의 종류에 따라서는 하나의 테스트 메소드를 파라미터로 전달하여 여러 개의 테스트 케이스를 만드는 것이 가능하다.

최종적인 개발 예시는 아래 <표 5>와 같다.

<표 5> 최종 테스트 케이스 예시

```
public void test(int testNumber) throws Throwable
{
        switch(testNumber)
        {
                case 0:
                        testIsEmpty();
                        break;
                case 1:
                        testAdd(3, 5, 8);
                        break;
                case 2:
                        testAdd(2, 1, 3);
                        break;
                default:
                        break;
        }
}
```

본 코드에서 testAdd의 경우는 testAdd(2, 1, 3)과 같이 테스트 메소드를 재사용한 것을 알 수 있다. 여기서의 주의 사항은 testNumber의 시작은 0번부터 시작된다는 점이다. 테스트 케이스가 3개라면 생성자에서는 super(3, "테스트클래스이름")으로 작성되어야 하고 test 메소드에서는 0,1,2의 case문으로 작성해야 한다.

4.2 테스트 코드 수정 절차

테스트 클래스의 수정은 일반적으로 테스트 케이스를 추가하거나 삭제하거나 단지 메소드 내용을 수정하는 경우로 볼 수 있다. 이럴 경우 다음의 절차대로 수행하여야 한다.

1) 생성자 메소드 수정, 삭제 혹은 추가
2) 삭체 혹은 추가인 경우 테스트 클래스의 생성자 메소드의 수정
3) 테스트 선택기 수정

삭제 혹은 추가할 경우 달라진 테스트 케이스의 개수를 생성자 메소드에 반드시 반영시켜야한다. 테스트 선택기에 달라진 테스트 케이스의 호출 코드를 삽입하여야 한다.

4. 결론

본 논문에서는 모바일 단위 테스트에 대한 절차 및 개발 방법에 대하여 알아보았다. 현재는 테스트 시점이 프로젝트의 완료시점에서 행해지고 있다, 하지만 완료시점에서 테스트를 할 경우 심각한 버그나 오류가 발생하였을 경우 프로젝트의 전체적인 변경이 불가피한 문제점이 발생한다.
그러나 본 논문에서 제시한 방법처럼 테스트를 개발 과정에서 개발과 동시에 한다면 많은 오류와 버그들을 감소시킬 수 있을 뿐만 아니라 시간, 비용 또한 절약 할 수 있는 장점이 있다.
다중 모바일 S/W 개발 환경에서 TDD를 적용하기 위해서는 기존의 테스크탑 어플리케이션에서 사용되는 테스트 기법 및 도구들은 사용되기 힘들며 다중 모바일 환경에서 동작하는 테스트 실행엔진의 개발이 필요하다. 향후 연구에서는 TDD를 적용한 자동화 테스트 도구 및 실행 엔진을 개발할 예정이며 이에 관련된 여러 방법들을 연구중에 있다.

참고문헌

[1] 다중 플랫폼 지원 모바일 응용 SW 개발환경 기술 개발 수행계획서, 2007.3.
[2] 단위 테스트 방법 기술서, 2007. 9
[3] 다중 플랫폼 지원 모바일 응용 SW 개발환경 (MOPAD) 사용자 시나리오, 버전 1.1, 2007.7.
[4] 다중 플랫폼 지원 모바일 응용 SW 요구사항명세서, 버전 1.0, 2007.8.20.
[5] David Weiss and Martin Zduniak, "Automated Integration Tests for Mobile Application in Java 2 Micro Edition", Poznan University of Technology, 2007
[6] Alex Ruiz and Yvonne Wang Price, "Test-Driven GUI Development with TestNG and Abbot", IEEE Software, May/June 2007.
[7] Atif M. Memon Ph.D, "A Comprehensive Framework for Testing Graphical User Interfaces", University of Pittsburgh, 2001.
[8] www.sourceforge.net/project