JUnit과 JTestCase 프레임워크에 기반한 테스트 데이터 및 코드 생성 도구

이유정[0] 최승훈
덕성여자대학교 전산학과
(yjlee[0], csh)@duksung.ac.kr

Test Data and Code Generation Tool based on JUnit and JTestCase Framework

Yu-Jeong Lee[0] Seung-Hoon Choi
Dept. of Computer Science, Duksung Wemon's University

요 약

신뢰성있는 소프트웨어의 개발을 위해 테스트의 중요성은 매우 크다. 특히, 최근에 점진적이고 반복적인 소프트웨어 개발 방법론이 각광을 받으면서 소프트웨어의 잦은 변경에 따른 회귀 테스트의 중요성이 점점 커지고 있다. 이에 따라 단위 테스트의 자동화에 대한 연구가 활발히 진행되고 있다. JUnit은 자바 클래스의 단위 레벨 테스팅을 도와 주는 테스트 지원 프레임워크이다. 또한, JTestCase는 테스트 데이터와 테스트 코드를 분리함으로써, 데이터 중심 테스팅(data-driven testing)을 지원하기 위해 개발된 JUnit 확장 프레임워크이다. 본 논문에서는, 이 두 개의 테스트 프레임워크와 자바 리플렉션 API를 이용하여, 하나의 클래스 파일을 읽어 들여 XML 형태의 테스트 데이터 파일과 테스트 드라이버 코드를 자동 생성하는 도구를 제안한다. 그리고, 구체적인 예를 통해 본 논문에서 제안하는 도구의 유용성을 보여준다. 본 논문의 테스트 도구는 회귀 단위 테스트에 필요한 노력을 줄여주고, 자바 클래스 단위 테스트를 지원하는 도구 개발의 기반 기술을 제공하며, 궁극적으로 소프트웨어 개발의 생산성을 향상시켜 준다.

1. 서론

신뢰성있는 소프트웨어 개발을 위해서 가장 중요한 단계 중의 하나가 소프트웨어 테스팅이다. 특히, 최근에 점진적이고 반복적인 소프트웨어 개발 방법론이 각광을 받으면서 " 분석 조금, 설계 조금, 구현 조금, 테스트 조금" 정책이 강조되고 있다[1]. 이에 따라 소프트웨어의 잦은 변경에 따른 회귀 테스트의 중요성이 점점 커지고 있다.

최근 각광 받는 새로운 소프트웨어 개발 패러다임인 XP(eXtreme programming)은 무엇보다 테스팅의 중요성을 강조한다[2]. 개발자는 코드 개발과 동시에 그 코드를 테스트하기 위한 테스트 코드를 작성함으로써, 개발 초기부터 테스트에 관심을 두자는 것이다.

테스트의 가장 기본적인 활동은 단위 테스팅(unit testing)이다. 객체 지향 시스템에서의 단위 테스팅이란, 일반적으로 클래스의 인터페이스를 이루는 메소드들의 시험을 의미한다[2]. 단위 테스트를 지원하기 위한 많은 방법들이 제안되었으며, 자바 프로그램을 위한 여러 가지 테스트 프레임워크가 개발되었다. 그 중에서 JUnit[3]은 자바 클래스의 단위 레벨 테스팅을 도와 주는 테스트 지원 프레임워크이다. 또한, JTestCase[4]는 테스트 데이터와 테스트 코드를 분리함으로써, 데이터 중심 테스팅(data-driven testing)을 지원하기 위해 개발된 JUnit 확장 프레임워크이다.

본 논문에서는, 이 두 개의 테스트 프레임워크와 자바 리플렉션 API를 이용하여, 하나의 클래스 파일을 읽어 들여, XML 형태의 테스트 데이터 파일과 테스트 드라이버 코드를 자동 생성하는 도구를 제안한다.

본 논문의 전체적인 구성은 다음과 같다. 2장에서 본 논문과 관련된 연구에 대해 언급하고, 3장에서 Money 클래스 예제를 이용하여 본 논문에서 제안한 테스트 데이터 및 코드 자동 생성 과정을 기술한다. 마지막으로, 4장에서 결론 및 향후 연구에 대해서 설명한다.

본 연구는 한국과학재단 목적기초연구(R06-2002-003-01006-0) 지원으로 수행되었음.

2. 관련 연구

JUnit은 자바 프로그램의 단위 테스트 자동화를 위하여 테스트 케이스 실행(test execution) 및 비교(comparison)를 위한 프레임워크를 제공한다. Assertion 방식[5]을 이용하여 각 메소드의 실제 결과값과 예상 결과값을 비교하는 방식을 취한다. JUnit은 테스트 자동화를 위한 간단하고 재사용하기 쉬운 클래스들을 제공하지만, 테스트 코드 안에 테스트 케이스가 포함되어 있기 때문에 테스트 데이터 변경 및 추가 시에 테스트 코드를 다시 컴파일해야 하는 단점을 가지고 있다.

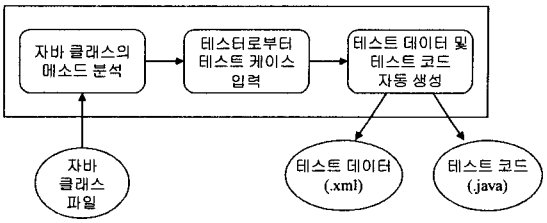
이러한 문제점을 해결하기 위하여 JUnit을 확장한 여러 가지 방법들이 제안되었다. [6]은 코드 작성 시에 XML의 한 형태인 JML(Java Modeling Language)을 이용하여 테스트 입력으로서 사전 조건(precondition)을, 테스트 결과로서 사후 조건(postcondition)을 표현함으로써 테스트 코드를 생성한다.

JXUnit[7]와 JTestCase[4]는 테스트 데이터와 테스트 코드를 분리함으로써 테스트 데이터 변경 시 테스트 코드를 재컴파

일해야 하는 JUnit의 단점을 해결하고자 하였다. JXUnit 은 제공되는 별도의 태그들을 이용하여 Test data를 작성하며, 테스트 코드의 eval( ) 메소드에서 실제 테스트하고자 하는 클래스의 메소드를 호출하는 방식을 취한다. JTestCase는 비교적 쉬운 태그를 이용하여 테스트 데이터를 XML 형태로 분리할 수 있도록 해 주며 JUnit을 이용한 테스트 코드와 그 구조가 유사하다. 본 논문에서는 JUnit과 JTestCase 프레임워크를 이용하여 테스트 데이터와 테스트 코드를 자동 생성하는 도구를 구현하였다.

3. 테스트 자동 생성 도구

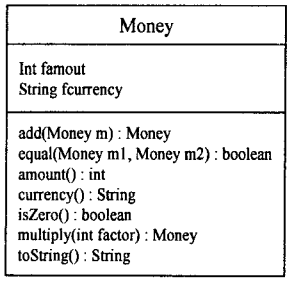
본 논문에서 제안한 테스트 자동 생성 도구의 전체적인 구조는 [그림1]과 같다. 테스트 자동 생성 도구는 먼저 선택된 자바 클래스 파일로부터 생성자와 멤버 메소드에 대한 정보(형식 인자의 타입과 개수, 반환형 등)를 분석한다. 테스터가 테스트 데이터와 예상 결과 값을 입력하면, 분석된 정보와 입력 값들을 바탕으로 XML 형태의 테스트 데이터와 자바 프로그램 형태의 테스트 코드가 자동 생성된다.



[그림1] 테스트 자동 생성 도구

3.1 예제: Money 클래스

본 논문의 테스트 자동 생성 도구를 설명하기 위하여 JUnit 프레임워크에서 제공한 Money 클래스 예제를 사용한다. 본 논문에서는 보다 일반적인 경우를 포함시키기 위해 원래의 Money 클래스를 약간 수정했으며, 수정된 Money 클래스의 UML 다이어그램은 [그림2]와 같다.



[그림2] Money 클래스 다이어그램

3.2 메소드 분석

객체 지향 시스템에서의 단위 테스트는 클래스가 포함하고 있는 각각의 메소드를 테스트하는 것이다. 각각의 메소드를 테스트하기 위해서는 테스트 데이터를 실행시키는 데 필요한 객체를 생성하는 과정(set up)이 필요하며, 테스트 코드는 반드시 이러한 set up 과정을 포함해야 한다. Set up을 위한 코드를 자동 생성하기 위해서는 그 메소드를 테스트하기 위해 생성해야 하는 객체의 개수가 몇 개인지를 분석하는 과정이 필요하다. 본 도구에서는 [표1]에서 보는 바와 같이, 자바의 리플렉션 API를 이용하여 테스트할 각 메소드의 프로토타입을 분석하여 생성해야 하는 객체의 개수를 파악한다.

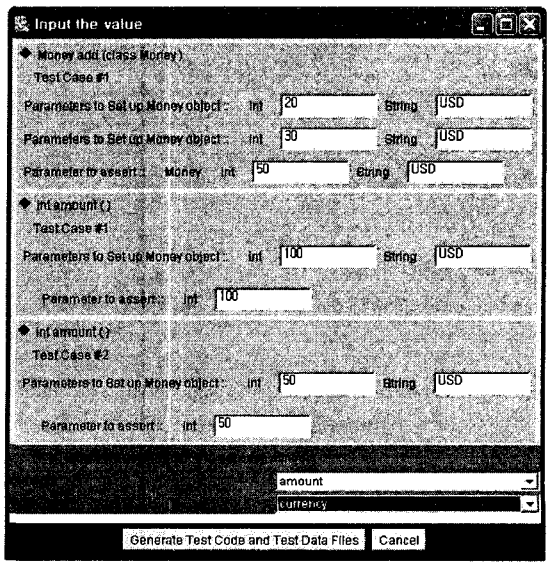
각 메소드를 테스트하기 위해서는 기본적으로 이 메소드를 실행시키기 위한 객체가 하나 필요하다. 여기에 형식 인자와 반환형에 포함되어 있는 참조형 선언의 개수를 더하면 이 메소드를 테스트하기 위해 생성해야 하는 객체의 개수가 구해진다. 예를 들어, Money add (Money m) 메소드의 경우, 형식 인자에 참조형이 1개, 반환 형에 참조형이 1개 포함되어 있으므로, 이 메소드를 테스트하기 위해 set up 과정에서 생성해야 하는 Money 객체의 개수는 모두 3개이다.

[표1] 형식 인자와 반환 값에 따른 Set Up 시 객체의 개수 분류

Parameter / Return type	참조형인자 없음	참조형인자 N개(n≥1)
참조형 아니다.	테스트에 필요한 객체의 수 : 1 Assert에 필요한 객체의 수 : 0 => 총 필요한 객체의 수 : 1 Ex) int amount()	테스트에 필요한 객체의 수 : 1 + n Assert에 필요한 객체의 수 : 0 => 총 필요한 객체의 수 : 1+n Ex) boolean equal(Money m1, Money m2)
참조형이다.	테스트에 필요한 객체의 수 : 1 Assert에 필요한 객체의 수 : 1 => 총 필요한 객체의 수 : 2개 Ex) Money multiply(int factor)	테스트에 필요한 객체의 수 : 1 + n Assert에 필요한 객체의 수 : 1 => 총 필요한 객체의 수 : 2 + n Ex) Money add(Money m)

3.3 테스트 케이스 입력

테스터는 테스트 자동 생성 도구가 제공하는 메소드 목록 중에서 테스트 하고자 하는 메소드의 종류와 테스트 횟수를 선택한 후, 테스트 하고자 하는 메소드마다 실인자값과 예상 결과값을 입력한다. 또한, 각 속성마다 속성의 값을 접근할 때 호출해야 하는 getter 메소드를 선택한다. 본 논문의 도구를 이용한 테스트 케이스 입력 과정은 [그림3]과 같다. 이 그림은 add( ) 메소드를 한번, amount( ) 메소드를 두 번 테스트하기 위한 테스트 케이스 입력 과정을 보여준다.



[그림3] add() 와 amount() 테스트 케이스 입력

3.4 테스트 코드 자동 생성

테스트 생성 도구는 테스터가 선택한 테스트할 메소드의 종류와 실인자값, 예상 결과값을 이용하여 JTestCase를 바탕으로 한 테스트 코드 파일을 자동 생성한다. Money 클래스로부터 자동 생성된 테스트 코드는 [그림4]와 같다.

```
public class MoneyTest extends TestCase {
  .....
  public static void main(String args[]) {                                    ①
      junit.textui.TestRunner.run (suite());  }
  public MoneyTest(String name) {
  ... .....
      String dataFile = "money.tests.test-data.xml";      _jtestcase = new JTestCase(dataFile, "MoneyTest");  }   ②
  public static Test suite() {
      TestSuite retval = new TestSuite();
      retval.addTest(new MoneyTest("testAdd"));      retval.addTest(new MoneyTest("testAmount"));              ③
      return(retval);  }
  .....
  public void testAdd(){
      try{ Vector testCases = _jtestcase.getNameOfTestCases("testAdd");
        for (int i=0; i<testCases.size(); i++) {
                String testCase = (String)testCases.elementAt(i);
                Hashtable params = _jtestcase.getTestCaseParams("testAdd", testCase);
                int parameter_1_1 = ((Integer)params.get("parameter_1_1")).intValue();
                String parameter_1_2 = ((String)params.get("parameter_1_2"));
                param_1 = new Money(parameter_1_1,parameter_1_2);
                .....
                result_1 = (Money)(param_1.add(param_2));
                int intResult_1 = result_1.amount();
                String strResult_1 = Integer.toString(intResult_1);
                boolean resultSucceed_1 = _jtestcase.assertTestCase("expected_1", strResult_1, "testAdd", testCase);
                String strResult_2 = result_1.currency();
                boolean resultSucceed_2 = _jtestcase.assertTestCase("expected_2", strResult_2, "testAdd", testCase);
                if(resultSucceed_1 && resultSucceed_2)   {                                                  ④
                      System.out.println("--MoneyTest::testAdd(). asserting result SUCCEED!");      }
                else { fail("***Fail to test Add when asserting result!"); }}
      } catch (Exception e) {e.printStackTrace();      } }
  .....
}
```

[그림4] 자동 생성된 테스트 코드

자동 생성된 테스트 코드는 크게 다음 네 부분으로 구성된다.

(1) main( ) : JUnit 도구를 실행한다.

(2) 생성자(예: MoneyTest()): 테스트 데이터 파일(xml)로부터 테스트 케이스들을 읽어 들인다.

(3) suite( ): TestSuite 객체를 생성하고 여기에 테스트 하고자 메소드들을 추가한다.

(4) 테스트할 메소드를 실제 실행하는 부분(예:testAdd()):

- 테스트 데이터 파일로부터 읽어 들인 실인자값을 이용하여 테스트에 필요한 객체를 생성함
- 테스트할 메소드를 호출한 후 성공 여부를 확인함

3.5 데이터 파일 자동 생성

테스트 생성 도구는 테스터가 선택한 테스트할 메소드의 종류와 실인자값, 예상 결과값을 이용하여 XML 형태의 테스트 데이터 파일을 자동 생성한다. 자동 생성된 테스트 데이터 파일은 [그림5]와 같다. 테스트 데이터 파일의 구성 요소는 다음과 같다. 테스트하기 위한 메소드는 <method> 엘리먼트로 표현되며, 테스트 시 생성해야 하는 객체의 실인자값들은 <params>의 <param>엘리먼트로 표현된다. 또한, 예상 결과값을 위한 객체의 실인자값들은 <asserts>의 <assert> 엘리먼트로 표현된다.

```
<?xml version="1.0" encoding = "UTF-8" ?>
<tests xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://jtestcase.sourceforge.net/config/jtestcase.xsd">
 <class name="MoneyTest">
  <method name="testAdd" test-case="add1">
   <params>
    <param name="parameter_1_1" type="int">20</param>
    <param name="parameter_1_2" type="String">USD</param>
    <param name="parameter_2_1" type="int">30</param>
    <param name="parameter_2_2" type="String">USD</param>
   </params>
   <asserts>
    <assert name="expected_1" type="String" action="EQUALS">50</assert>
    <assert name="expected_2" type="String" action="EQUALS">USD</assert>
   </asserts>
  </method>
  <method name="testAmount" test-case="amount1">
   <params>
    <param name="parameter_1_1" type="int">100</param>
    <param name="parameter_1_2" type="String">USD</param>
   </params>
   <asserts>
    <assert name="expected" type="String" action="EQUALS">100</assert>
   </asserts>
  </method>
  <method name="testAmount" test-case="amount2">
   ..............
  </method>
 </class>
</tests>
```

[그림5] 자동 생성된 테스트 데이터

4. 결론 및 향후 연구

본 논문에서는, 자바 프로그램의 단위 테스트 프레임워크인 JUnit과 JTestCase, 자바 리플렉션 API를 이용하여, XML 형태의 테스트 데이터 파일과 테스트 드라이버 코드를 자동 생성하는 도구를 제안하였다. 그리고, 구체적인 예를 통해 본 논문에서 제안하는 도구의 유용성을 보여주었다.

본 논문의 테스트 자동 생성 도구는, 테스트 데이터와 테스트 코드의 자동 분리를 가능하게 함으로써, 테스트 데이터 파일 만의 편집이 가능하도록 해준다. 따라서 테스터는 프로그래밍에 대한 지식 없이 테스트 데이터를 추가 또는 변경할 수 있으며, 테스트 코드는 재컴파일이 필요없는 장점을 가진다.

본 논문의 테스트 도구는 회귀 단위 테스트에 필요한 노력을 줄여주며, 자바 클래스 단위 테스트를 지원하는 도구 개발의 기반 기술을 제공함으로써, 궁극적으로 소프트웨어 개발의 생산성을 향상시켜 준다.

향후 연구 과제로서, 두 개 이상의 객체들이 상호 작용하는 프로그램에 대한 테스트 코드 및 데이터 자동 생성 도구, 데이터베이스를 접근하는 클래스를 위한 테스트 자동 도구, 스트링 타입 이외의 assertion을 지원하는 테스트 프레임워크에 대한 연구가 필요하다.

참고 문헌

[1] J. D. McGregor & D. A. Sykes, "A Practical Guide to Testing Object-Oriented Software", 2001, Addison-Wesley.

[2] R. Hightower & N. Lesiecki, "Java Tools for Extreme Programming", 2002, John Wiley & Sons.

[3] http://www.junit.org

[4] http://jtestcase.sourceforge.net/docs/main.html

[5] B. Korel and A. M. Al-Yami, "Assertion-Oriented Automated Test Data Generation", Proceedings of the 18th international conference on Software engineering, May 1996.

[6] Yoonsik Cheon and Gary T.Leavens, "A Simple and Practical Approach to Unit Testing : the JML and JUnit Way", November 2001.

[7] http://jxunit.sourceforge.net/