# 유닛 테스트 자동화 도구를 위한 프레임워크 설계

김영상[O] 백창현 박승규
아주대학교 멀티미디어 및 컴퓨터 구조 연구실
{guri92[O], labmedia, sparky}@ajou.ac.kr

# A Framework of Automatic Unit Test Tool

Younsang Kim[0], Changhyun Baek, Sungkyu Park
Multimedia & Computer Architecture Lab, Ajou University

**요 약**

테스트 주도 개발은 익스트림 개발 방법론의 바탕이 되는 방법론이다. 테스트 주도 개발은 소스 코드를 구현을 하기 전에 테스트 케이스를 먼저 만드는 작성한다. .Net 개발 환경에서 테스트 주도 개발을 하기 위해 NUnit이라는 유닛 테스트 프레임워크를 많이 사용하고 있다.[1] 본 논문은 NUnit 유닛 테스트 프레임워크를 기반으로 리플랙션을 사용하여 유닛 테스팅 자동화 도구의 제작을 위한 프레임워크를 제안한다. 그리고 이 프레임워크를 이용하여 유닛 테스트 뼈대를 자동으로 생성하는 프로그램을 구현하였다.

## 1. 서 론

신뢰성과 질을 향상시키기 위해서는 소프트웨어는 테스트 되어야 한다.[2]. 유닛이란 서브 프로그램이나 서브 프로그램들의 그룹을 말한다. 소프트웨어가 복잡한 구조로 되어 있다면 모든 코드를 분석하는 것은 많은 노력이 필요하다. 전체가 아닌 단위 유닛을 테스트 하는 유닛 테스트 기법은 이러한 환경을 해결하기 위한 좋은 방법이 될 수 있다.[3] 본 논문에서는 유닛 테스트 기법 중에 xUnit 프레임워크를 기반으로 한 .Net용 프레임워크인 NUnit를 사용하고 있다.

최근 몇 년 동안 새로운 소프트웨어 개발 기법인 익스트림 프로그래밍 기법이 주목을 받아왔다. 익스트림 프로그래밍 기법에서는 테스트 주도 개발을 기본으로 짝 프로그래밍, 리팩토링등의 기법을 통해 생산성 향상시키며 기민한 개발이 가능하도록 하고 있다. [4]

본 논문에서는 우선 리플랙션과 애드인 그리고 xUnit 테스트 프레임워크에 관하여 설명하고 이들을 사용하여 자동화 테스트 도구를 만들기 위한 프레임워크를 설계 한다. 그리고 프레임워크를 사용하여 유닛 테스트 뼈대를 자동 생성하는 툴을 제작하였다. 마지막으로 결론과 향후 진행사항으로 끝을 맺는다.

## 2. 관련 연구

### 2.1 Reflection

리플랙션이란 언어 메커니즘으로 런타임 라이브러리를 로드하여 형식 정보를 받아올 수 있게 한다. 이러한 리플랙션은 여러 OOP 언어에서 적용되어있다.[5] .Net 환경에서 리플랙션은 System.Reflection 네임스페이스에 정의되어있다. 언어에 관계없이 .Net안의 모든 종류의 오브젝트는 각자의 Type 정보를 가지게 되며 GetType 함수를 사용하여 현재 오브젝트의 정보를 받아올 수 있다. 이러한 오브젝트의 정보를 표현해 주는 것이 메타데이터이다. 메타 데이터는 계층적 구조를 가지고 있으며 최상위 개체가 바로 어셈블리이다. 그림1은 이러한 메타 데이터의 계층도를 보여주고 있다.

메타 데이터는 다음과 같은 사항을 제공하고 있다.[6]

- 어셈블리를 통한 클래스 타입 제공
- 클래스안의 메소드와 변수 정보
- 버전 정보
- 다른 어셈블리와 모듈간의 종속성
- 보안 Attribute

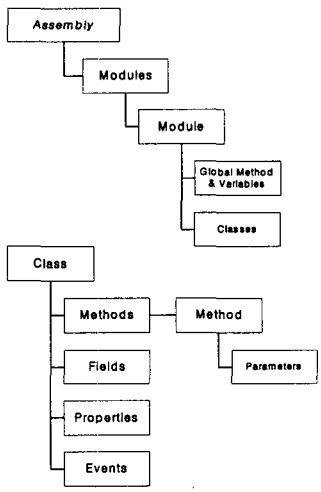


그림 1 메타데이터 계층도

### 2.2 애드인

.Net 개발 환경에서 Visual Studio.Net IDE의 기능을 확장하는 방법에는 메크로와 애드인이 있다.

이중 애드인은 EnvDTE 네임스페이스에 있는 객체를 사용하여 Visual Studio.Net의 IDE를 직접 작성할 수 있다. 이중 본 논문은 프로젝트 솔루션 정보, 코드 편집기 객체, 도구 상자 객체를 구현

에 이용하였다.

2.3 xUnit 프레임 워크

xUnit을 바탕으로 한 유닛 테스트 프레임워크는 Kent Beck에 의하여 Smalltalk 버전으로 최초로 개발 되었다. 이 유닛 테스트 프레임워크를 바탕으로 JUnit, NUnit, cppUnit등의 다양한 언어의 버전으로 변형되어 사용되고 있다.[7] 최근에 많이 사용하고 있는 자바 개발용 IDE인 Eclipse 또한 JUnit을 탑재 하고 있다. 이중 본 논문에서는 .Net용 유닛 프레임워크인 NUnit 을 바탕으로 구현하였다.

NUnit의 기본구조는 크게 다음 3가지 키워드와 단정 함수로 구현 되어있다.

첫번째, "[TestFixture]"라는 키워드를 통해 현재의 클래스가 테스트용 클래스라는 것을 보여준다.

두번째 "[Test]"라는 키워드는 테스트용 클래스 중에서 실제로 테스트에 사용하는 함수라는 것을 말해준다.

마지막 "[SetUp]" 키워드는 현재 클래스의 초기 설정을 해주는 함수라는 것을 말해준다.

그리고 AssertEquals 와 같은 단정 함수를 사용하여 테스트 케이스를 만들어 유닛의 테스트를 구현한다. 이렇게 작성된 테스트용 클래스를 테스트 러너를 사용하여 테스트 결과를 확인한다.

## 3. 유닛 테스트 자동화 툴을 위한 프레임워크

3.1 기본 구조

애드인 프로그램은 보통 Connect 시점에서 등록이 된다. 이런 경우는 ConnectMode 를 ext_cm_UISetup 일 경우 메뉴를 등록한다. 하지만 본 프레임워크의 경우는 OnStartupComplete 이벤트가 들어올 경우에 기능을 등록한다. 이때에 등록할 기능들은 Abstract Factory 패턴을 사용하여 동일 인터페이스(추상 클래스)로 등록이 된다.[8] 자동화 유닛 테스트 기능도 이때 등록이 된다.

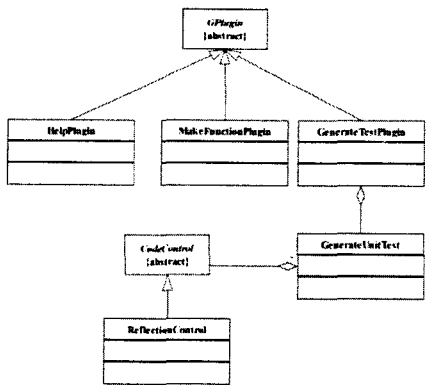

그림 2 클래스 다이어그램

그림2는 제안 프레임워크의 기본 구조를 보여주고 있다. ReflectionControl 클래스는 리플랙션 기능을 담당하고 있다. 현재 프로젝트의 Output파일을 어셈블리로 로드하여 어셈블리 안의 클래스 정보, 클래스 안의 메소드, 프로퍼티등의 정보를 제공한다. GenerateUnitTest는 ReflectionControl의 정보를 바탕으로 유닛 테스트 자동화를 담당 하게 된다.

GPlugin은 추상 클래스로 애드인의 여러가지 기능들(예를 들어 유닛 테스트 자동화, 리팩토링 도구, 헬프)을 관리 하기 위한 최상위 클래스이다. 이것을 상속받아 여러 기능들이 해당하는 각자의 역할을 하게 되는데 이중에 GenerateTestPlugin은 유닛 테스트 자동화 기능을 담당하게 된다. 애드인과 관련하여 메뉴가 보여야 하는 시점 관리, 애드인의 등록 그리고 GenerateUnitTest 클래스를 통한 유닛 테스트 자동화 실행을 담당하게 된다.

3.2 구현 메카니즘

자동화 유닛테스트 생성 구현은 다음과 같은 순서로 진행된다.

(1) 현재 사용하고 있는 바이너리, 즉 exe나 DLL파일을 프로젝트의 설정 정보를 통해 위치를 받아와 로드 한다. 어셈블리로 로딩을 하여 현재 유닛 테스트로 만들려는 메타 정보를 받아온다.

(2) 기본적인 클래스 구조를 생성하여 버퍼에 넣는다..

- 기본 네임스페이스를 설정하여 넣는다.
- System, NUnit.Framework 네임스페이스 추가
- [TestFixture] 키워드와선택된 네임스페이스의 이름에 Test 를 추가하여 생성한다.

(3) 클래스 안의 함수를 버퍼에 넣는다.

- 생성자를 넣는다.
- [SetUp] 키워드와 테스트 설정 함수를 넣는다.
- [Test] 키워드와 Public 으로 선언된 함수를 테스트할 함수를 넣는다.
- 생성한 함수에 테스트 케이스를 생성한다.

(4) 버퍼의 내용을 파일에 써 넣는다.

(5) 생성된 파일을 프로젝트에 추가한다.

## 4. 유닛 테스트 뼈대 자동 생성 구현

본 논문에서 제안한 프레임워크로 테스트 케이스의 생성을 제외한 뼈대를 자동으로 생성하는 프로그램을 구현하였다.

4.1 기본 동작

특정 클래스에 대해 유닛 테스트가 필요할 경우 또는 테스트 주도 개발 방식으로 해당 클래스의 추상 형태의 함수로 구현되어있는 클래스의 기본 유닛 테스트를 만들고자 할 경우에는 원하는 클래스를 선택 한 후 마우스 오른쪽 버튼으로 "Code Window" 메뉴를 호출 하여 유닛테스트 뼈대를 생성한다. 생성된 테스트 뼈대는 현재 선택한 프로젝트에 등록된다. 자동으로 만들어진 뼈대 안에 자신이 필요한 케이스를 수동으로 작성한다.

4.2 구현 결과

그림 3은 SampleClassForTest라는 클래스로 생성자와 두개의 퍼블릭 함수로 되어있다. 이 클래스로 유닛 테스트를 생성하면 그림 4와 같은 결과를 얻을 수 있었다.

SampleClassForTest 클래스를 테스트하기 위한 SampleClassForTestTest 클래스가 생성되었고 생성자와 설정 함수 그리고 퍼블릭 함수 2가지를 테스트하기 위한 testMyPlus와 testMyMinus함수가 생성되었다.

```
namespace SampleClass
{
    public class SampleClassForTest
    {
        public SampleClassForTest()
        {
        }

        public int MyPlus(int nParam1, int nParam2)
        {
            return nParam1 + nParam2;
        }

        public double MyMinus(int nParam1, int nParam2)
        {
            return (double)(nParam1 - nParam2);
        }
    }
}
```

그림 3 테스트 하려는 클래스

```
namespace SampleClass
{
    using System;
    using NUnit.Framework;
    using System;

    [TestFixture]
    public class SampleClassForTestTest
    {

        //! Auto generated Code : Function SampleClassForTestTest
        public SampleClassForTestTest()
        {

        }

        [SetUp]
        //! Auto generated Code : Function Setup
        public void Setup()
        {

        }

        [Test]
        //! Auto generated Code : Function MyPlus
        public void testMyPlus()
        {
            //! Add User Unit Test code
        }

        [Test]
        //! Auto generated Code : Function MyMinus
        public void testMyMinus()
        {
            //! Add User Unit Test code
        }

    }
}
```

그림 4 자동 생성된 유닛 테스트용 클래스

실제로 개발자는 이렇게 생성된 유닛 테스트 뼈대를 사용하여 그안에 수동으로 자신이 필요한 유닛테스트를 작성하게 된다.

## 5. 결론 및 향후 연구

현재의 소프트웨어는 광범위해지고 사용자 요구 사항의 변화가 많아지면서 과거의 폭포수 모델을 개선하려는 노력이 이루어지고 있다. 그중에 하나가 기민한 개발 방법론이다.[9] 이러한 기민한 개방 방법론의 대표적인 방법이 익스트림 프로그래밍 개발 방법론이다. 익스트림 프로그램 개발 방법론은 테스트 주도 개발을 바탕으로 만들어졌다. 이러한 테스트 주도 개발에서는 유닛을 테스트를 하는 것은 중요하다. 하지만 xUnit 프레임워크로 유닛 테스트를 하기 위해서는 개발자의 일정한 노력이 필요하다. 이런 노력은 비효율 적이며 자동화가 되면 유닛 테스트의 효율성은 증가한다.

본 논문에서는 우선 리플랙션과 유닛 테스트 프레임워크와 에드인에 대해 살펴보았다. 이를 이용하여 자동화 유닛 테스트 툴을 위한 프레임워크에 대해 제안하였다. 그리고 이를 사용하여 유닛 테스트 뼈대를 자동으로 생성하는 프로그램을 구현해 보았다

현재는 수동으로 작성하기 번거로운 유닛 테스트 뼈대를 작성하는 수준이다. 리플랙션을 사용하여 테스트가 가능한 메소드의 리턴값, 파라메터 정보들을 얻어올 수 있다. 이 정보를 바탕으로 블랙박스 테스트 기법을 사용하여 케이스를 자동으로 생성하는 기능의 제작이 필요하다.

또한 테스트 케이스의 자동 생성을 위하여 Residual branch coverage approach[10]를 사용한 툴들이 개발되어 있다. [11][12] 이들은 자바를 이용하고 있으므로 이 프레임워크를 이용하여 C#용으로 변환한다면 케이스 자동화를 구현 할 수 있을 것이다.

그리고 본 프레임워크를 이용하여 Mock 객체를 자동 생성하는 것 또한 가능하다. Mock 객체란 테스트하기 힘든 객체를 테스트하기 위하여 실체 코드가 아닌 비슷한 행동을 하는 더미 객체를 말한다. [13] 그밖에 리팩토링 도구, 정적 분석 도구 또한 이 프레임워크를 이용하여 구현이 가능하다.

## 6. 참조


[1] NUnit , http://www.nunit.org

[2] A Semi-Automatic Generator for Unit Testing Code Files Based on JUnit , Cheng-hui Huang, Huo Yan Chen, ICSMC.2005

[3] Unit testing for software assurance, Hamlet.R, CMPASS.1989

[4] Extreme Programming Explanied, Kent Beck,

[5] Aspect-oriented programming with c# and .Net, Schult.W, Polze.A, ISORC2002

[6] Introduction To .Net Metadata , Matt Pietrek , http://www.develop.com/conferences/conferencedotnet/materials/N4.pdf

[7] Pragmatic Unit Testing with JUnit, Andy Hunt, Dave Thomas

[8] Design Patterns of GoF , E. Gramma, R. Helm, R.Johnson, J.Vlissidies, Addison Wesley

[9] Agile software development methods, P. Abrahamsson, O.Salo, J. Ronkainen, J. Warsta

[10] Residual Test Coverage Monitoring, Christina Pavlopoulou, Michal Young, ICSE.1999

[11] Hansel 1.0 . http://hansel.sourceforge.net

[12] Gretel, http://www.cs.uoregon.edu/research/perpetual/dasada/Software/Gretel/

[13] Endo-Testing : Unit Testing with Mock Objects, Tim Mackinnon, Steve Freeman, Phlip Craig, XP2000