테스트 주도 개발을 위한 유연한 단위 테스트 도구로 변경

(A Flexible Unit Testing Tool for Test Driven Development)

전 석 환 † 김 정 동 ††
(Seokhwan Jeon) (Jeongdong Kim)

백 두 권 †††
(Doo-Kwon Baik)

요 약 테스트 주도 개발의 장점을 얻기 위해서는 효율적인 테스트 도구의 사용은 필수적이다. 기존의 통합 자동화 테스트 도구는 표준화 되지 않은 스크립트(script) 언어를 사용하거나 단위 테스트에 적합하지 않은 경우가 많다. 기존의 단위 테스트 도구는 대부분 프로그램의 원시코드에 테스트를 위한 코드가 추가된다. 이것은 원시 코드의 복잡도를 높이고 테스트 케이스 변경 시 원시코드의 여러 부분을 수정해야 하는 문제점이 있다. 본 논문에서는 테스트 주도 개발 시 개발자가 표준 자바 스크립트(Java script)를 이용하여 테스트 케이스 수정을 용이하게 할 수 있도록 유연한 테스트 도구의 설계를 제안하고 제안된 설계의 가용성을 검증하고자 테스트 도구를 구현하였다.

키워드 : 단위 테스트, 테스트 도구, 테스트 주도 개발, 익스트림 프로그래밍

Abstract The efficient test tool is indispensable to the test driven development. The test tool is very helpful to get the advantage of the test driven development. Many traditional automation test tool does not support standard script language and are not suitable to unit test. In traditional unit test tool, a code for the test is added at the source code. Such method makes the complexity of a source code and has a problem which must modify many part of the source code when the test case is changed. In this paper, we present a design technique of a flexible unit testing tool which makes a developer can modify easily the test case by using the standard java script in test driven development. We implement the test tool by this design technique to verify the availability of this technique.

Key words : Unit test, Test tool, Test driven development, Extream Programming

1. 서 론

테스트 주도 개발에서는 코드의 작성에 앞서 단위 테스트 케이스를 점진적으로 작성하고 이 테스트 케이스를 만족하도록 실제 프로그램을 작성해가는 개발 방법이다. 테스트 주도 개발에서의 테스트는 소프트웨어에서 중복되는 로직을 제거하고 프로그램의 이해도를 향상시키며, 새로운 코드의 추가로 인한 오류를 직관적으로 알 수 있도록 하는 수단이 된다[1]. 따라서 소프트웨어의 효율적인 설계를 가능하게 하고 테스트에 들어가는 시간적 오버헤드(Overhead)를 줄이고 최종적으로 소프트웨어 개발의 생산성을 향상시킨다[2-4].

테스트 주도 개발에서의 테스트는 주로 단위 테스트에 기반하는데 일반적인 단위 테스트의 프레임워크는 XUnit 프레임워크를 기반으로 하는 JUnit, CUnit, Cpp UTest, CppUnitLite, CPP Unit for C++ 등이 있다[5]. 이러한 단위 테스트 도구는 테스트 대상이 되는 소스코드에 직접 테스트 매크로와 테스트를 위한 헤더파일을 포함하고 이것을 함께 컴파일(Complie)하여 테스트를 수행한다.

테스트와 리팩터링(Refactoring)을 계속 반복하면서 소프트웨어를 개발하는 방법을 사용할 때 기존의 단위 테스트 도구는 원시코드가 변경됨에 따라 테스트 클래스의 복잡도가 높아지고 코드 추가 작업이나 컴파일 과정에서 시간적 낭비를 초래할 수 있다. 또한, 요구사항 변경 시 테스트 케이스를 수정하기가 용이하지 않은 문제점이 있다[6].

따라서, 본 논문에서는 테스트 도구 설계 시 표준 자바 스크립트를 지원하는 스크립트 엔진(Script Engine)과 XML 파서(Parser)를 포함시키고 계층 구조을 이용하여 테스트 도구가 유연한 구조로 설계될 수 있도록

· This paper is supported by the second Brain Korea (BK) 21 Project
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '테스트 주도 개발을 위한 유연한 단위 테스트 도구 설계 및 구현'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 고려대학교 컴퓨터공학과
click92@gmail.com
†† 학생회원 : 고려대학교 컴퓨터학과
kjd4u@korea.ac.kr
††† 종신회원 : 고려대학교 컴퓨터학과 교수
baikdk@korea.ac.kr
논문접수 : 2008년 8월 28일
심사완료 : 2008년 11월 16일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제2호(2009.2)

하는 방법을 제안한다. 테스트 도구의 입력 데이타(data)인 XML 기반 문서에는 테스트 대상 모듈(module)에 대한 메타(Meta) 정보와 자바 스크립트로 기술한 테스트 케이스가 포함 되고, 자바 스크립트 객체를 관리하는 모듈을 통해 입력된 정보를 기반으로 테스트 케이스를 검증할 수 있는 테스트를 수행한다. 제안된 설계에 따라 구현된 테스트 도구는 개발자가 자바 스크립트로 테스트 케이스를 쉽게 생성하고 변경할 수 있음을 보인다.

본 논문의 구성은 제2장에서 관련 연구로서 테스트주도 개발과 단위 테스트 도구를 소개하고 제3장에서는 유연한 테스트 도구 설계에 대해 제안한다. 제4장에서는 제안한 테스트 도구와 기존 도구의 결합도를 비교·평가하고 마지막으로 5장에서는 결론 및 향후 연구에 대해 기술한다.

2. 관련 연구

2.1 테스트 주도 개발

최근 주목을 받고 있는 기민한 개발 방법론(Agile Methodology)은 그 점유율을 높여 가고 있고 다수의 기업 및 개발자는 기민한 개발 방법론의 도입에 대해 고려하거나 도입을 준비하고 있다. 기민한 개발 방법론에는 익스트림 프로그래밍(Extreme Programming), 스크럼(Scrum) 등이 속하고 익스트림 프로그래밍의 핵심 과정으로 테스트 주도 개발이 포함된다[7].

테스트 주도 개발의 가장 큰 장점은 프로그램의 개발 과정이 유연하고 실용적이며 높은 품질의 프로그램을 개발할 수 있다는 점이다. 이러한 장점을 유지하기 위해서는 요구 사항의 분석 주기를 짧게 하여 소프트웨어를 개발하는 동안 요구 사항 변경을 즉각적으로 적용한다. 프로그램의 품질을 유지하면서도 요구 사항 변경을 만족하려면 변경되는 프로그램의 모든 코드에 대해서 반복적이고 다양한 테스트가 가능해야만 한다.

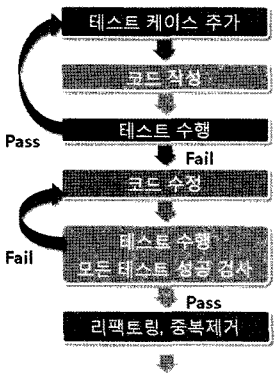

그림 1 테스트 주도 개발의 주기

테스트 주도 개발은 주로 단위 테스트에 기반하는데 기존의 단위 테스트와의 가장 큰 차이점은 그림 1과 같이 테스트 케이스의 개발을 실제 코드의 개발 이전에 한다는 것이다[8]. 따라서 100%의 테스트 커버리지를 목표로 하게 되고 코드의 품질을 향상시킬 수 있게 된다.

테스트 주도 개발은 리팩토링을 통해 테스트에 실패하는 부분을 지속적으로 개선해나가므로 요구 사항 변경이 없다 하더라도 리팩토링의 결과를 확인하기 위한 테스트가 반복된다. 이러한 테스트에 대한 부담은 일반적인 소프트웨어 개발 방법에서 테스트 주도 개발로의 전환을 어렵게 하는 원인이 된다. 따라서 테스트 주도 개발에 적합하며 개발자의 테스트에 대한 부담을 최소화 할 수 있고, 테스트 케이스의 변경에 유연하게 대처할 수 있는 단위 테스트 도구가 필요하다.

2.2 단위 테스트

저 수준(low-level)에서 일어나는 테스트를 단위 테스트 또는 모듈 테스트라고 한다. 단위 프로그램을 각각 테스트하여 저 수준의 오류들을 발견하여 수정하고 나면, 이들을 한데 모아 모듈 그룹 단위로 통합 테스트를 시행한다[8,9]. 테스트 주도 개발에서는 프로그램 원시코드 보다 테스트 프로그램을 먼저 작성해야 하기 때문에 모듈의 기능 명세에 더 중점을 두는 블랙 박스 테스트가 선행되어야 한다[10].

테스트 주도 개발에서 사용하는 기존의 단위 테스트 도구들은 XUnit 프레임워크를 기반으로 만들어진 것이 많은데 표 1은 그 중 하나인 CPPUnit의 사용 예이다[6,11].

표 1 CPPUnit의 사용 예제코드

```
class CWebLogin :  public CppUnit::TestCase
{
public:
        CWebLogin();
        CPPUNIT_TEST_SUITE(CWebLogin);
        CPPUNIT_TEST(Test_SetUserInfo);
        CPPUNIT_TEST_SUITE_END();

        void Test_SetUserInfo()
        {
            TryLogin ("TestID", "TestPWD");
        }
```

표 1에서 테스트 대상 클래스는 CWebLogin이고 테스트 프레임워크에 포함된 클래스인 CppUnit::TestCase 클래스를 상속 받는다. XUnit 기반의 단위 테스트 도구은 JUnit 역시 CppUnit과 동일한 구조로 원시코드 내에 테스트를 위한 클래스를 작성하고 이 클래스에서 테스트 대상 함수를 호출한다.

표 1에서와 같이 XUnit프레임 워크 기반의 테스트 도구는 테스트 케이스에 대한 코드가 원시코드에 정의되고 테스트 클래스가 테스트 기반 클래스를 상속받는데 이것은 테스트 대상 모듈의 결합도를 높인다. 결합도가 높은 소프트웨어는 유지보수성이 떨어지는 문제점이 발생하고 테스트 케이스 수정을 어렵게 만든다.

본 논문에서는 이러한 문제점을 해결하고자 테스트 케이스를 분리하고 테스트 클래스는 테스트 대상 인터페이스(Interface)를 호출만 할 수 있는 구조로 테스트 도구를 설계하였다.

3. 유연한 테스트 도구 설계 및 구현

3.1 테스트 도구의 설계

본 논문에서 제안하는 테스트 도구에서는 테스트 대상 모듈과 자바 스크립트의 객체를 사상 시키는 방법을 사용한다. 자바 스크립트 엔진은 자바 스크립트 런타임(Runtime)을 메모리 내에 생성하고 하나 이상의 컨텍스트(Context)를 생성한다. 컨텍스트는 자바 스크립트를 실행하기 위한 변수나 객체를 위한 저장소를 제공하고 동적으로 생성된 자바 스크립트 객체를 포함한다. XML 문서에는 다수의 테스트 대상 모듈에 대한 메타 정보를 기록할 수 있고 자바 스크립트에서는 테스트 대상 모듈을 자바 스크립트 객체로 간주하고 메소드를 호출 할 수 있다.

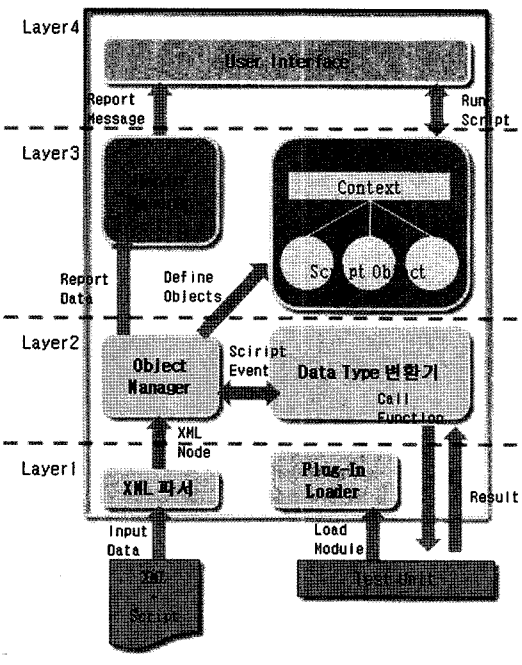

그림 2 테스트 도구 아키텍처

그림 2는 본 논문에서 제안하는 테스트 도구의 아키텍처이다. 그림 2의 Layer 1은 테스트 케이스와 테스트 대상모듈에 대한 메타 정보의 입력부분과 테스트 대상 모듈을 호출하기 위한 모듈 인터페이스 계층이다. Layer 2는 자바 스크립트 런타임에 생성되는 객체와 사상 되는 클래스를 관리하는 계층이다. 네이티브(native) 코드를 사용한 테스트 대상 모듈은 자바 스크립트에서 사용하는 데이타와 그 타입(type)이 다르기 때문에 Layer 2에서는 함수 호출 시 매개변수의 형 변환을 위한 데이타 타입 변환과정이 필요하다. Layer 3은 자바 스크립트 엔진을 내장하고 스크립트를 실행하는 계층이다. Layer 4는 테스트 수행 정보를 사용자가 볼 수 있도록 사용자 인터페이스를 가지는 계층이다. 각각의 계층을 구성하는 요소들은 필요에 따라 다른 모듈로 대체 할 수 있으므로 전체 구조가 유연성을 가진다.

본 논문에서 제안하는 테스트 도구의 설계에 대한 자세한 설명은 다음과 같다.

• XML 파서

제안한 설계에서 XML 기반 문서에는 테스트 대상에 대한 메타 정보와 테스트케이스를 기술한 자바 스크립트가 테스트를 위한 입력 데이타가 된다. XML 파서는 입력된 테스트 데이타를 파싱(parsing)하여 XML 노드와 스크립트를 분리시키고 구조화하여 메모리에 저장한다.

• Object Manager

XML 노드는 테스트 대상에 대한 메타 정보로써 테스트 대상의 경로, 함수명, 매개변수 타입과 매개변수 수 등의 정보를 가진다. Object Manager는 이 정보를 이용하여 테스트 대상을 자바 스크립트의 런타임 내에 생성시킨 스크립트 객체에 사상시키고 테스트 실행 시 호출해야 하는 함수들을 스크립트 객체의 메소드로 정의한다. 정의된 메소드가 스크립트에 의해 호출되면 컬백(callback)함수가 호출되고 이를 통해 테스트 대상 함수가 호출된다. 함수 호출 시 각각의 매개변수는 Data Type변환기에 의해서 자바 스크립트에 정의된 매개변수를 네이티브 타입의 매개변수로 변환시키고 매개변수를 한번에 전달하기 위해 직렬화 한다. 또한, 기본적인 매개변수 이외에 반환 값 및 오류정보를 전달받기 위한 변수가 추가로 전달된다.

• Script Engine

스크립트 엔진은 테스트 실행 명령에 따라 입력된 자바 스크립트를 실행시킨다. 자바 스크립트를 이용하여 테스트 케이스를 작성할 때는 자바 스크립트 런타임 내에 미리 생성된 테스트 대상과 사상된 다수의 객체와 객체가 가진 메소드를 조합하여 기술할 수 있고 문법은 표준 ECMA-262 스펙에 따른다[12].

• Report Manager

자바 스크립트의 실행과 이에 따른 네이티브 함수의 호출이 이루어지면서 테스트 도구는 그 수행 결과와 정상 실행여부에 대한 결과를 Report Manager로 정리하여 테스트 도구의 사용자 인터페이스에 나타낸다.

• Data Type 변환기

자바 스크립트 엔진의 런타임 내에서 사용되는 스크립트 변수 타입과 테스트 대상모듈과의 매개변수 교환을 위해 매개변수의 직렬화와 데이타 형의 변환을 수행한다.

• Plug-In Loader

테스트 대상에 대한 메타 정보를 이용하여 테스트 대상 모듈을 동적으로 로드한다. 테스트 대상 모듈의 종류나 특성에 따라 Plug-In Loader는 IDL 정의 및 상속등의 방법으로 대체되거나 두 가지 형태 모두를 지원함으로써 테스트 대상 모듈과의 다양한 인터페이스를 지원하는 구조로 확장될 수 있다.

표 2는 XML 기반 문서에 테스트 대상 모듈에 대한 메타 정보와 테스트 케이스를 자바 스크립트로 기술한 예이다. 표 2에서 <module>은 테스트를 위해 플러그인(PlugIn) 함수를 추가한 테스트 대상 소스를 포함한 모듈이다. XML 파서를 통해 생성된 <module>노드는 자바 런타임에서 하나의 객체로 생성된다. <module>의 자식 노드인 <method> 노드는 이 객체의 메소드(method)가 된다. <script> 노드 내에 테스트 케이스를 생성하여 테스트 대상에 대한 테스트 스크립트를 기술할 수 있다.

표 2 테스트 정보의 표현

```
<?xml version="1.0" encoding="euc-kr"?>
<module>
  <name>TestWebLogin</name>
  <path>D:₩bin₩debug₩TestWebLogin.dll</path>
    <type>Win32DLL</type>
    <method paramCount=2>
      <name>webLogin</name>
      <param type="string"/>
      <param type="string"/>
    </method>
</module>
<script>
if (TestWebLogin.webLogin('testid', 'testpassword'))
{
        TestModule.TestFunction('Test Message');
}
```

3.2 테스트 도구의 구현

그림 3은 본 논문에서 제안하는 설계에 따라 테스트 도구를 구현하여 특정 웹 서비스에 로그인 처리를 하는 모듈에 대한 테스트 결과이다.

그림 3의 ①은 로드한 모듈의 종류와 각각의 모듈에서 호출 가능한 함수를 계층적으로 보여준다. 테스트 대상 모듈은 다수가 로드 될 수 있고 각각은 자바 스크립트 런타임내의 객체에 사상된다. 그림 3의 ②는 각 모듈이나 함수에 대한 간단한 정보를 나타낸다. 툴바의 실행 버튼을 클릭하면 자바 스크립트가 실행되면서 자바 스크립트 런타임내의 객체의 메소드가 호출되고 사상된 테스트 대상 모듈의 함수들이 호출된다. 그림 3의 ③은 함수의 호출이 성공적인 경우이고 호출 결과로 리턴값인 웹 쿠키를 출력한 것이다. 그림 3의 ④는 호출된 함수내부에 오류가 존재하여 함수의 호출이 실패함을 보인다.

그림 3의 사용자 인터페이스는 테스트 케이스의 로드 및 실행과는 별도로 아키텍처의 최상위 계층에서 다양하게 변경될 수 있다.

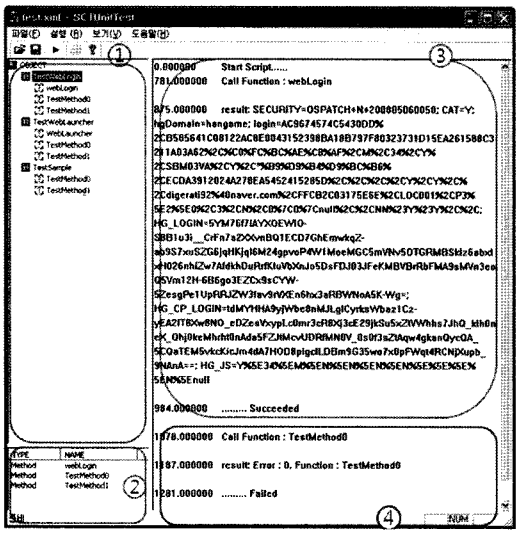

그림 3 테스트 도구 구현 및 실행화면

4. 비교 평가

본 논문에서는 동일한 테스트 대상 코드에 대해 기존의 테스트 도구와 제안한 도구에 대해 각각 모듈 간의 결합도를 비교하였다. 결합도는 소프트웨어의 구성요소들이 얼마나 밀접하게 연관되어 있는가를 측정하는 척도이다[13,14].

본 논문에서는 CBO(Coupling Between Objects)와 WMC(Weighted Methods per Class) 등의 척도를 사용하였다. CBO는 하나의 클래스가 다른 클래스나 속성을 사용하여 결합된 수를 의미한다[15].

$$CBO(C_i) = \sum C_u(i) + \sum C_c(i) + \sum C_p(i)$$

$\sum C_u(i)$:= 클래스 C_i가 사용하고 있는 클래스의 수

$\sum C_c(i)$:= 클래스 C_i가 포함하고 있는 클래스의 수

$\Sigma C_p(i)$:= 클래스 C_i를 포함하고 있는 클래스의 수

WMC는 클래스에 정의된 메소드의 복잡도의 합으로 WMC의 값이 크면 해당 클래스는 두 개 이상의 클래스로 분할될 필요가 있다.

$$WMC = \sum_{i=1}^{n} c_i$$

여기서 메소드의 복잡도는 순환수(cyclomatic number)로 구한다[16].

표 3은 기존의 XUnit 기반 테스트 도구와 본 연구에서 제안한 방법에 대해 각각 CBO와 WMC 등을 측정한 결과이다.

표 3에서 나타난 바와 같이 본 연구 결과는 기존의 테스트 도구보다 WMC나 CBO의 수치가 작다. 기본의 테스트 프레임워크는 테스트 대상 클래스가 여러 클래스와 연관되게 하여 결합도가 높고 테스트 케이스 변경이나 반복되는 테스트에서 효율성이 떨어진다.

표 3 기존 테스트 모델과 제안 모델의 결합도 비교

매트릭 (Metric)	기존 모델	제안 모델
LOC (Line of Code)	61	42
WMC	16	8
CBO	6	2
DIT (Depth of Inheritance Tree)	1	0

본 논문에서 제안한 테스트 도구는 설계 시 테스트 케이스의 작성부분을 분리하고 테스트 대상 클래스는 단순히 테스트 인터페이스가 수행하는 기능에 집중할 수 있도록 고안되었다. 또한, 분리된 테스트 케이스는 스크립트로 기술되어 있으므로 반복되는 테스트에서 재활용성이 높다.

5. 결론 및 향후 연구

본 논문에서는 테스트 주도 개발에서 테스트 케이스 변경 시 테스트 대상의 메타 정보와 자바 스크립트를 이용하여 빠르게 변경된 테스트를 수행할 수 있는 단위 테스트 도구의 설계를 제안하고 구현하였다. 또한, 기존의 테스트 도구와의 비교 평가를 통해 제안한 테스트 도구는 기존의 도구에 비해 모듈의 결합도가 낮음을 보였다.

향후 연구로는 다양한 가상 객체나 사용자 인터페이스를 지원하면서 인터페이스 상속 기법등을 적용하여 테스트 대상에 대한 다양한 인터페이스를 지원하는 개선된 테스트 도구를 설계할 것이다.

참 고 문 헌

[1] Boby George and Laurie Williams, "A structured experiment of test-driven development," Information and Software Technology, Vol.46, Issue.5, pp. 337-342, 2004.

[2] David Janzen and Hossein Saiedian, "Test-Driven Development: Concepts, Taxonomy, and Future Direction," The flagship publication of the IEEE Computer Society, Vol.38, No.9, pp. 1-3, 2005.

[3] Hans Wasmus and Hans-Gerhard Gross, "Evaluation of Test-Driven Development," Delft University of Technology Software Engineering Research Group Technical Report Series, 2007.

[4] 정창신, 이계임, 김종희, 정순기, "XML기반 테스트 정보를 공유하는 소프트웨어 테스팅 자동화 프레임워크의 설계", 한국컴퓨터정보학회 논문지 제10권 제3호, pp. 89-99, 2005.

[5] Charles D. Allison, "The simplest unit test tool that could possibly work," Journal of Computing Sciences in Colleges, Vol.23, Issue.1, pp.183-189, 2007.

[6] Panagiotis Louridas, "JUnit: Unit Testing and Coding in Tandem," IEEE SOFTWARE, Vol.22, No.4, pp. 12-15, 2005.

[7] Andrew Begel and Nachiappan Nagappan, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," Empirical Software Engineering and Measurement, pp. 255-264, 2007.

[8] 김희진, 최병주, 윤석진, "테스트 주도 개발 (TDD)에서의 모바일 응용 소프트웨어 성능 테스트 방안", 한국정보과학회 학술발표논문집 제34권 제2호(B), pp. 143-146, 2007.

[9] Michael Ellims and James Bridges and Darrel C. Ince, "The Economics of Unit Testing," Empirical Software Engineering, Vol.11, No.1, pp. 5-31, 2006.

[10] 서광익, 최은만, "객체지향 소프트웨어를 위한 주요 블랙박스 테스트 기법들의 비교", 한국정보과학회, pp. 1-17, 2006.

[11] 김영상, 백창현, 박승규, "유닛 테스트 자동화 도구를 위한 프레임워크 설계", 한국정보과학회 한국컴퓨터종합학술대회 논문집 pp. 184-186, 2006.

[12] Oystein Hallaraker and Giovanni Vigna, "Detecting malicious JavaScript code in Mozilla," Engineering of Complex Computer Systems, pp. 85-94, 2005.

[13] 차석기, 임정은, 백두권, "런타임을 고려한 소프트웨어 컴포넌트 메트릭스", 한국정보과학회 2007 한국컴퓨터종합학술대회 논문집 제 34권 제1호(B), pp. 90-95, 2007.

[14] S.R. Chidamber and C.F. Kemerer, "A Metrics Suit for Object Oriented Design," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol.20. No.6, pp. 476-493, 1994.

[15] 김유경, 고병선, 고현희, 박재년, "객체지향 소프트웨어의 품질 메트릭스에 관한 연구", 한국정보과학회 학술발표논문집 제26권 제1호(A), pp. 104-112, 1999.

[16] 윤희환, 김영집, 구연설, "객체지향 프로그램의 화이트박스와 블랙박스 재사용성 측정 메트릭스", 한국정보과학회 논문지 소프트웨어 및 응용 제28권 제2호, pp. 104-112, 2001.