특집

# 셀프테스트 컴포넌트 기술 연구

최 은 만[1] 송 호 진[2]

목 차

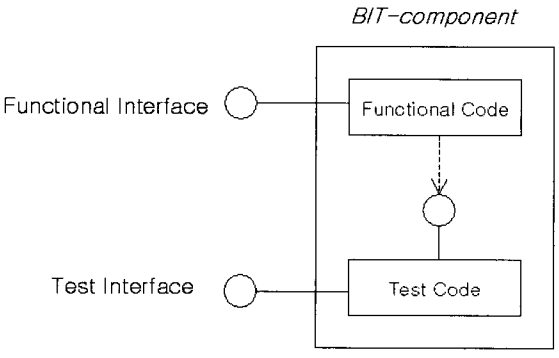


## 1. 서 론

최근 소프트웨어 개발을 위한 각 분야에서 컴포넌트 기반 개발(Component Based Development)기술에 초점을 맞추고 많은 연구와 개발이 이루어지고 있다. 소프트웨어는 컴포넌트의 조립을 통해 완성되며, 이는 개발비용과 시간의 절감, 검증된 컴포넌트의 사용으로 인한 소프트웨어 신뢰성의 증가, 컴포넌트 개발을 통한 자산으로써의 가치 등을 고려해 보았을 때 컴포넌트 기반 개발은 매우 중요한 의미를 지니고 있다.

컴포넌트를 사용하여 소프트웨어를 개발하기 위해서는 신뢰성 있는 컴포넌트를 사용해야 한다. 이에 따라 컴포넌트의 기능 및 성능, 신뢰성을 검증하기 위한 과정은 매우 중요하다고 할 수 있다. 컴포넌트는 보통 실행코드의 형태로 배포되는 것이 일반적이며, 이로 인해 이를 검증하기 위한 테스트 방법도 블랙박스 형태로 제한된다.

이러한 제약을 컴포넌트 셀프테스트 방법을 사용함으로써 극복할 수 있다. 컴포넌트 셀프테스트는 BIT(Built-in Test)를 이용하여 컴포넌트의 내부에 테스트를 위한 기능들을 내장하여 컴포넌트 자체에서 테스트를 수행할 수 있는 방법을 말한다. 또한 컴포넌트에 내장된 BIT는 소프트웨어 개발 라이프사이클(life cycle)전반에 걸쳐 재사용될 수 있으며, 이로 인해 소프트웨어의 유지보수성을 증가시킬 수 있다[1].




(그림 1) 셀프테스트를 위한 BIT 컴포넌트의 개념

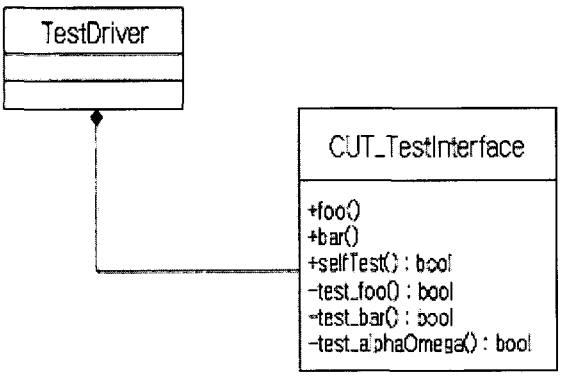
셀프테스트 컴포넌트는 이러한 BIT를 내장한 채로 배포되어 수행되며, 컴포넌트 개발자와 이를 사용하는 사용자는 BIT 인터페이스의 접근을 통해

1) 동국대학교 컴퓨터공학과 부교수
2) 동국대학교 컴퓨터공학과 석사 재학중

컴포넌트의 통합 테스트(Integration test)를 비롯한 지속적인 검증 작업을 수행할 수 있다. 셀프 테스트 컴포넌트를 위한 BIT는 컴포넌트의 본래 기능을 수행하는 기능 인터페이스(Functional Interface)와 하나 이상의 테스트 인터페이스(Test interface)가 합쳐진 형태로 되어 있으며 이는 (그림 1)에 나타나 있다. 테스트 인터페이스는 평상시에는 사용되지 않으며, 테스트가 필요할 때만 활성화되어 테스트를 수행하게 된다[3].

컴포넌트의 경우 재사용을 위한 환경은 개발환경과 달라질 수 있다. 따라서 테스트가 완전히 끝났다 하더라도 사용 환경에 맞는지 다시 테스트 할 필요가 있다. 이를 위하여 컴포넌트 개발 시 사용된 모든 테스트 오라클을 컴포넌트 사용 환경에서의 테스트에 따로 이용하는 것은 비효율적이다. 그러므로 셀프테스트를 위한 BIT 컴포넌트 내에 이미 포함되어 준비된 테스트 오라클을 사용해 테스트를 수행하는 방법이 더 효율적인 방법이라 할 수 있다.

## 2. 테스트 기능 내장의 개념




(그림 2) Built-in Test Driver

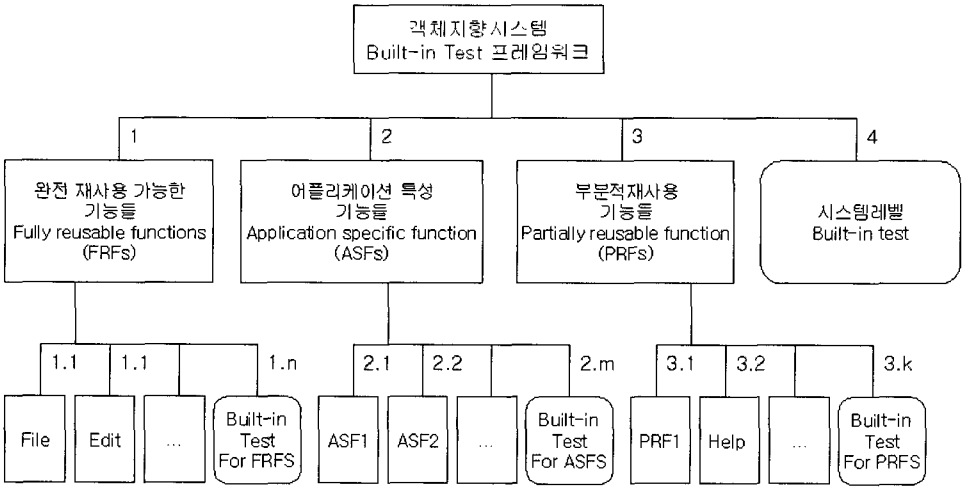
BIT는 컴포넌트 테스트에만 국한된 방법은 아니다. Java나 C++과 같은 객체지향 언어에서 테스트하고자 할 때 클래스 내에 Private으로 선언된 메소드나 인스턴스 변수를 외부에서 접근 할 수 있는 방법이 없기 때문에, 클래스 내부에 BIT를 포함시키고 BIT테스트 인터페이스를 통해 테스트를 수행할 수 있다. 이러한 프로그램 언어 상의 접근 제약에 대한 테스트 한계를 극복할 수 있는 유일한 방법으로서 BIT를 사용한다[3].

Java에서는 BIT를 클래스의 main부에 내장하는 경우와, BIT를 내부 클래스로 작성하여 외부 클래스(outer class)의 메소드와 변수를 직접적으로 접근하여 테스트할 수 있는 방법이 있다. BIT는 테스트하고자 하는 클래스의 내부에 포함되어 작성되기 때문에 클래스 내부의 멤버함수나 변수에 제한 없이 접근하여 테스트할 수 있다. 또한 JUnit과 같은 테스트 프레임워크를 상속받아 테스트에 이용할 수 있다.

(그림 3)은 이러한 BIT를 포함하는 자바 클래스를 작성한 것으로서 클래스 내에 BIT를 포함하는 내부 클래스를 작성하여 내부 클래스 내에 main부를 테스트 인터페이스로 사용하도록 하는 내용을 보여주고 있다.

```
public class ClassUnderTest {
   private int x;
   public void setx(int newx) {
      x = newx;
   }
   public static class BuiltInTest {
     public static void main(String argv[]){
       test001();
       return;
     }
     public void test001() {
       ClassUnderTest out = new ClassUnderTest();
       out.setx(0);
       if(x == 0) {
           System.out.println("passed");
       } else {
           System.out.println("failed");
       }
        return;
     } // End built-in Test
  } // End ClassUnderTest
```

(그림 3) Built-in Test 테스트 메소드의 예



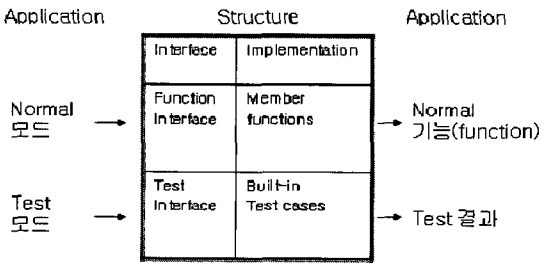

(그림 4) 객체지향 시스템 Built-in Test 프레임워크

BIT는 단순한 하나의 객체 모듈에 대한 테스트부터, 여러 개의 객체로 구성된 시스템 프레임워크에 대한 테스트에 이용될 수 있다. 다음 (그림 4)는 다수의 객체로 구성된 소프트웨어 시스템의 구조와, 이를 테스트하기 위한 BIT 클래스와 서브시스템을 나타낸 것이다[4].

시스템은 완전 재사용 가능한 기능, 애플리케이션 특성 기능, 부분적 재사용 기능으로 분류하여 각각의 목적에 따른 BIT를 이용한 개별적인 테스트를 수행하게 된다. 기능별 테스트가 완료되면, 시스템 프레임워크 전체의 시스템 수준의 BIT를 이용한 시스템 레벨 테스트를 진행한다.

이러한 BIT 오브젝트(그림 5)들은 Normal과 Test의 두 가지 모드로 나뉘어진다. BIT는 평상시 Normal모드 상태로 동작하게 되며, 테스트 필요 시 Test모드로 활성화되어 내장된 BIT를 이용한 테스트를 수행하게 된다.

최종 사용자는 Test 모드 상의 모든 BIT 멤버함수들을 재사용할 수 있다. 이러한 BIT 방법을 사용함으로써 블랙박스(기능) 또는 화이트박스(구조)에 의해 생성된 테스트 케이스를 모두 시험해 볼 수 있다.




(그림 5) BIT Normal 모드와 Test 모드

## 3. 셀프테스트를 위한 BIT 컴포넌트의 설계

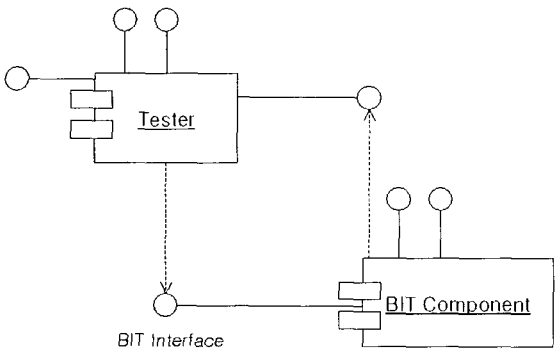
소프트웨어의 조각으로 개발된 컴포넌트의 테스트는 기존의 테스트방법과는 다른 성향을 나타낸다. 보편적으로 컴포넌트의 사용자는 컴포넌트의 내부에 대해서 잘 알지 못한다. 컴포넌트 사용자는 오직 외부에서 보여지는 컴포넌트의 인터페이스와 행위만을 관찰할 수 있을 뿐이다. 이에 따라 BIT 메커니즘을 이용하여 외부로부터 소프트웨어 컴포넌트의 내부에 접근하는 방법은 반드시 필요하다. 셀프테스트를 위한 BIT는 추후 프로그래밍에 드는 수고와 시스템 오버헤드를 최소화 할

수 있도록 설계되어져야 한다.

BIT 컴포넌트는 다음과 같은 두 가지 주요한 측면으로 분류 될 수 있으며, 이렇게 분류된 두 가지 측면의 테스트를 수행할 수 있도록 BIT가 설계되어야 할 것이다[5].

- 계약 테스팅(Contract Testing) : 컴포넌트 간의 상호 동작에 대한 관점에서, 컴포넌트가 명시된 계약사항에 맞게 동작하는지를 검증하는 것.
- 서비스 품질 테스팅(Quality of Service Testing) : 서비스 품질 테스팅의 주요 초점은 운용 환경에서 소프트웨어 컴포넌트가 계속적으로 올바른 서비스를 제공하는지를 검증하는 것이다. 서비스 품질 테스팅은 컴포넌트와 컴포넌트가 전개되는 시스템 사이의 상호 작용에 대한 테스트를 말한다.

BIT 컴포넌트는 BIT 인터페이스를 통해 연결된 테스터(Tester)를 이용해 테스트를 수행하게 되며, 테스터와 BIT 컴포넌트와의 관계는 (그림 6)과 같은 클라이언트/서버 관계의 형태로 나타낼 수 있다[1].

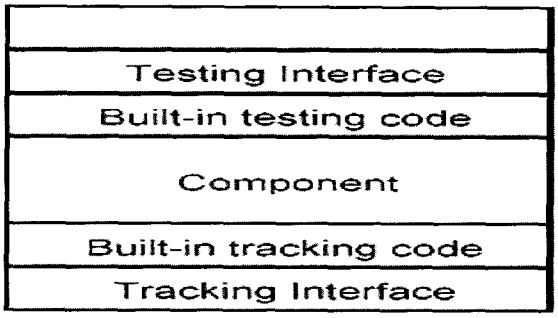



(그림 6) BIT 컴포넌트와 Tester와의 관계

테스터는 BIT 컴포넌트의 BIT 인터페이스를 통해 테스트 오퍼레이션을 호출하여 테스트를 수행하고 수행 결과를 테스터의 인터페이스를 통하여 전달받게 된다. 테스터는 소프트웨어 컴포넌트 라이프사이클 전반에 걸쳐 재사용되며, 실행환경 및 실행시간에 영향을 받지 않고 테스트를 수행할 수 있는 장점을 지닌다.

## 4. EJB 셀프테스트 컴포넌트 아키텍처

EJB(Enterprise Java Beans)는 현재 컴포넌트 개발을 위한 컴포넌트 표준으로 자리잡고 있다. 현재 엔터프라이즈급 컴포넌트 기반 소프트웨어 개발을 위한 표준으로서 가장 많이 사용되고 있다. EJB를 통한 BIT 컴포넌트 개발을 통해 컴포넌트의 테스트 능력(testablity)을 높이고 플러그 인 테스트(plug-in-and-test)환경을 제공하며 컴포넌트 테스트 자동화를 이루기 위한 방법으로써 유용하게 사용될 수 있다. 테스트 가능한 BIT 빈(Bean)의 구조는 (그림 7)에 나타나 있다[6].




(그림 7) BIT 빈의 구조

BIT 빈은 테스팅 인터페이스를 통해 컴포넌트 내부의 BIT의 테스팅 코드를 수행하여 Built-in Tracking 코드를 통해 테스트 결과나 테스트 수행내용을 기록(tracking)하게 된다. 기록 결과는 외부의 파일이나, 데이터베이스에 저장하게 된다. 다음 (그림 8)은 테스트 수행 기록을 위한 테스트 메시지와 메시지 기록 큐의 구조를 자바 코드의 형태로 나타낸 것이다[8].

```
public class TraceMessage {
    public int traceID;
    public long timeStamp;
    public int traceType;
    public Byte[] traceBuffer;
    public TracMessage(int sizeofBuffer){
        traceBuffer = new byte[sizeofBuffer];
    }
}
import TraceMesssage;
public class TraceMessageQueue {
    public void send(TraceMessage msg){
    // send the message to the message queue
    .....
    }
    public void receive(TraceMessage msg){
    ..... // receive the message from the queue
    }
}
```

(그림 8) 테스트 기록을 위한 기록 메시지와 큐

테스트 인터페이스는 다음과 같은 구조를 통해 3가지 기본기능을 제공하게 된다.

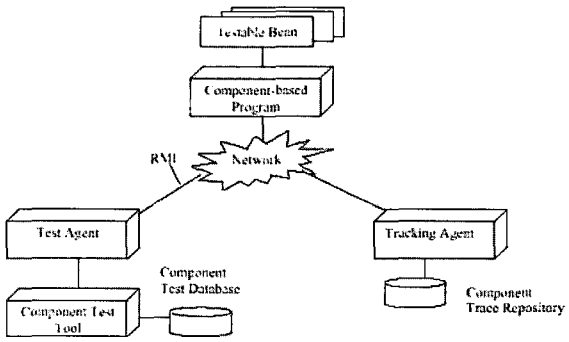
- 테스트 셋업(테스트 케이스 또는 데이터) : setParameters를 통해 오퍼레이션의 파라메터를 셋팅한다.
- 명세된 기능에 대한 테스트 수행 : runMethod를 통해 주어진 오퍼레이션을 수행
- 테스트 결과 보고와 확인 : validateTest Result를 통해 테스트 결과를 확인한다.

이와 같은 테스트를 위한 표준 인터페이스를 정의하여 테스트에 이용하게 된다.

J2EE환경에서의 컴포넌트들은 여러 시스템에서 분산되어 전개될 확률이 매우 높다고 할 수 있다. 이로 인해 단순한 객체나 객체지향 시스템 프레임워크와는 다른 형태의 테스트가 필요하다. 분산환경의 컴포넌트 셀프 테스트를 위한 기본환경은 다음과 같은 형태로 갖춰져야 한다.

- BIT 빈(셀프테스트가 가능한 BIT 컴포넌트)
- 테스트 에이전트(Test agent)
- 컴포넌트 테스트 저장소(Component test repository)
- 기록 에이전트(Tracking agent)
- 컴포넌트 기록 저장소(Component tracking repository)

J2EE에서의 분산 환경은 네트워크를 통해 각 분산 컴포넌트 객체들이 RMI프로토콜을 이용하여 데이터를 주고받게 된다. 다음 (그림 9)는 분산환경에서의 셀프테스트를 위한 BIT 컴포넌트 테스트 환경을 나타내고 있다.




(그림 9) 분산 환경에서의 BIT 빈 테스트[6]

각각의 분산된 BIT를 내장한 빈들은 컴포넌트 테스트 툴에 관련된 테스트 에이전트를 통해 네트워크를 통한 RMI프로토콜을 이용해 각각의 테스트 오퍼레이션을 호출하여 수행하게 된다. 또한 기록 에이전트를 통해 테스트가 수행된 내용과 결과를 컴포넌트 기록 저장소에 저장하게 되는 구조를 가지고 있다.

이렇게 저장된 정보는 컴포넌트 유지보수 측면이나 컴포넌트 재사용 측면에서 매우 중요한 정보로 이용 될 것이다. 이러한 구조는 .NET과 같은 표준 컴포넌트 프레임워크에서 같이 적용될 수 있다. 이러한 테스트를 수행함으로써 단순한 컴포넌트 결함을 찾아내는 것만이 아니라, 컴포넌트간의 행위 및 컴포넌트의 수행 능력을 분석, 측정할 수 있다[7].

## 5. 결론 및 향후 연구

객체지향 소프트웨어 컴포넌트는 캡슐화의 특성으로 인해 외부에서 내부의 행위나 구조를 알 수 없어, 테스트에 어려움이 따르게 된다. 이러한 문제점을 컴포넌트 셀프 테스트를 위한 BIT를 이용하여 해결할 수 있다. 컴포넌트 셀프테스트를 위한 BIT는 재사용의 측면에서 소프트웨어 테스트에 소요되는 비용과 수고를 줄일 수 있으며, 소프트웨어 품질을 높일 수 있다. 또한 안정적인 테스트 모델을 제공하고, 서드파티(third-party) 컴포넌트의 행위를 모니터링 할 수 있으며, 유연성 있는 테스트 제어 능력을 제공하게 된다.

셀프테스트를 위한 BIT는 EJB와 .NET과 같은 분산 컴포넌트 환경에서의 테스트 에이전트를 이용한 방법으로 통해 테스트를 수행할 수 있으며, 테스트 에이전트의 실제 구현이 필요하다. 분산된 웹 컴포넌트가 BIT로 구성되어 있다면 단순히 같은 컴포넌트 플랫폼 상의 컴포넌트 테스트뿐만이 아니라, 이 기종 플랫폼간의 상호 연동이 가능한 웹 서비스와 같은 기술을 이용하여 서로 다른 소프트웨어 컴포넌트 프레임워크간의 컴포넌트 테스트도 가능할 것이다.

## 참고문헌


[1] Yingxu Wang, "A Method for Built-in Tests in Component-based Software Maintenance", Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on, 1999, pp.186-189.

[2] J Vincent, "Built-in-Test Vade Mecum - Part1, A Common BIT Architecture", Southampton Institute/Bournemouth University, UK. 2002.

[3] Robert B.Binder, Testing Object-Oriented Systems, Models, Patterns, And Tools, ADDISON-WESLEY.

[4] Yingxy Wang, "On Built-in Test Reuse in Object-Oriented Framework Design", ACM Journal on Computing Surveys, Vol.32, No.1, March, pp.7-12

[5] Hakan Edler, " BIT in software Component", EC IST 5th Framework. 1999.

[6] Jerry Gao, "On Building Testable Software Components", San Jose State University.

[7] Jerry Gao, "Component Testablity and Component Testing challenges" , International Workshop on Component-Based Software Engineering, 2002.

[8] Jerry Gao, " Tracking Software Components" , Journal of Object Oriented Programming, 2001.

[9] Hans-Gerhard Gross, "Component+ Methodology - Built-in Contract Testing: Method and Process", IESE-Report 030.02/E, 2002

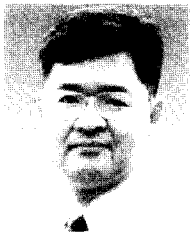
[10] Stephen H.Edwards, "A Framework for Practical, Automated Black-Box Testing of Component-Based Software", Softwere Testing, Verification and Reliable, 2001

[11] Jonas Hornstein, "Test Reuse in CBSE Using Built-in Tests", Workshop on Component-based Software Engineering, 2002, pp. 8-11

[12] Hans-Gerhard Gross, "Built-in testing for Component-based Development",

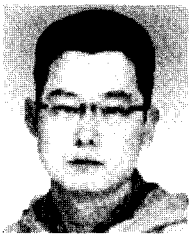
16th European Conference on Object-Oriented Programming, 2002.
[13] J Vincent, "Built-In-Test Vade Mecum - Part3, Quality of Service Testing", Southampton Institute/Bournemouth University, UK, 2002.

## 저자약력



**최 은 만**

1982년 동국대학교 전산학과 (학사)
1985년 한국과학기술원 전산학과 (공학석사)
1993년 일리노이 공대 전산학과 (공학박사)
1985년-1988년 한국표준연구소 연구원
1988년-1989년 데이콤 주임연구원
1993년-현재 동국대학교 컴퓨터공학과 부교수
2002년-2001년 콜로라도 주립대 전산학과 방문교수
관심분야 : 소프트웨어 테스트, 컴포넌트 기반 소프트웨어 개발, 소프트웨어 품질, 메트릭
이 메 일 : emchoi@dgu.ac.kr



**송 호 진**

2002년 호서대학교 전산학과 (학사)
2002년-동국대학교 컴퓨터공학과 (석사 재학 중)
관심분야 : Component Self-Testing, Component Domain Analysis, EJB, Component이해 및 활용 방안
이 메 일 : nemoz@dongguk.edu