객체지향 프레임웍의 Hot Spot에 Built-in Tests를 내장하는 방법

(Embedding Built-in Tests in Hot Spots of an Object-Oriented Framework)

신 동 익 † 전 태 웅 †† 이 승 룡 †††
(Dongik Shin) (Taewoong Jeon) (Syungyoung Lee)

요 약 객체지향 프레임웍은 다수의 응용 소프트웨어의 개발에 반복적으로 재사용되므로 철저한 시험이 요구될 뿐만 아니라 재사용 시 확장된 프레임웍에 대해서도 추가적인 시험이 필요하다. 그런데 프레임웍은 개조, 합성된 확장 부위의 시험에 대한 제어와 관찰을 어렵게 하는 성질을 갖고 있다. 본 논문에서는 프레임웍을 개조, 확장하여 응용 프로그램을 구현할 때 발생할 수 있는 오류들이 시험을 통하여 효율적으로 발견될 수 있도록 프레임웍의 가변 부위에 테스터 컴포넌트들을 BIT(Built-in Test)로 내장하는 방법을 기술한다. 프레임웍에 이와 같이 내장된 테스터 컴포넌트들은 프레임웍의 시험 시 제어와 관찰을 용이하게 하여 프레임웍의 시험성을 높여준다. 여기서 제안된 방법으로 설계된 테스터 컴포넌트들은 시험 대상 프레임웍의 확장 부위에 프레임웍 코드의 변경이나 간섭 효과가 없게 부착할 수 있고 필요에 따라 동적으로 탈착할 수 있다.

키워드 : 객체지향 프레임웍, 프레임웍의 가변부위, 프레임웍의 시험성, Built-in 테스트

Abstract Object-oriented frameworks need to be systematically tested because they are reused in developing many applications software. They also need additional testing whenever they are extended for reuse. Frameworks, however, have properties that make it difficult to control and observe the testing of the parts that were modified and extended. In this paper, we describe the method of embedding test components as BIT(Built-In Test) into the framework's hot spots in order to efficiently detect the faults through testing that occurred while implementing application programs by modifying and extending the framework. The test components embedded into a framework make it easy to control and observe testing the framework, and thereby improve the testability of frameworks. Test components designed by the method proposed in this paper can be dynamically attached and detached to/from hot spots of a framework without changes or intervention to the framework code.

Key words : Object-oriented framework, Hotsports of a framework, Testability of a framework, Built-in Tests

1. 서 론

최근의 객체지향 기술은 소프트웨어를 여러 응용 분야에 재사용 가능한 프레임웍(framework)의 형태로 개발하는 기술로 발전하고 있다[1, 2, 3]. 이에 따라 프레임웍을 기반으로 한 소프트웨어의 개발이 빠르게 확산되고 있다. 프레임웍은 다수의 응용 소프트웨어의 개발에 반복적으로 재사용할 목적으로 개발되므로 철저한 시험이 필요하다. 뿐만 아니라 시험이 완료된 프레임웍이 재사용을 위하여 개조, 확장될 때마다 추가적인 재시험이 요구된다. 따라서 프레임웍에 대한 조직적인 시험은 프레임웍에 기반한 소프트웨어 개발의 효율성과 그렇게 개발된 소프트웨어의 신뢰성에 중요한 요인으로 작용한다.

객체지향 프레임웍은 상호 협동하는 다수의 추상 클래스들(abstract classes)과 구체 클래스들(concrete

· 본 연구는 정보통신부의 2000년도 대학기초사업(과제번호 : 2000-112-01)의 지원으로 수행되었음
† 학생회원 : 고려대학교 전산학과
eastwing@tiger.korea.ac.kr
†† 종신회원 : 고려대학교 전산학과 교수
jeon@tiger.korea.ac.kr
††† 종신회원 : 경희대학교 전자계산공학과 교수
sylee@nms.kyunghee.ac.kr
논문접수 : 2000년 6월 26일
심사완료 : 2001년 10월 29일

classes)로 구성된 아키텍쳐를 제공한다[4]. 프레임웍의 클래스들 중 일부는 프레임웍의 아키텍쳐가 허용하는 범위 내에서 응용에 따라 변경 가능한 가변부위로 설계되어 있다. 프레임웍은 이러한 가변부위들을 클래스 상속과 객체 합성 메커니즘에 따라 개조, 확장함으로써 응용 시스템의 구현에 재사용된다. 그런데 프레임웍이 (재)사용되는 상황(context)은 상당히 복잡할 수 있다[5]. 그리고 프레임웍과 같이 시스템 수준의 아키텍쳐가 재사용될 때 발생할 수 있는 아키텍쳐와 컴포넌트들 사이의 불일치(mismatch) 유형들은 매우 다양하고 예측하기 어렵다[6]. 따라서 프레임웍이 재사용될 때 개조, 합성되는 클래스들이 프레임웍이 가정하는 제약조건들을 항상 만족하도록 강제하거나 이들의 위반 여부를 사전에 빠짐없이 확인하기가 쉽지 않다. 따라서 프레임웍이 개조, 확장되면, 변경된 프레임웍에 발생 가능한 점진 결함들(progressive faults)과 회귀 결함(regression faults)들에 대한 (재)시험이 요구된다. 개조, 확장된 프레임웍에 대한 효율적인 시험을 위해서는 결함의 발견에 효과적인 테스트 케이스들을 효율적으로 생성하는 방법뿐만 아니라 프레임웍이 변경될 때마다 이루어지는 변경 부위에 대한 반복적인 테스트를 효율적으로 수행할 수 있도록 제어와 관찰을 용이하게 하는 테스트 실행 환경의 구축이 필요하다.

현재, 객체지향 소프트웨어에 대한 시험 방법들이 많이 소개되어 있다[7, 8]. 이러한 시험 방법들은 대부분 프레임웍의 시험에도 적용이 가능하다. 그러나 객체지향 프레임웍은 기존의 이러한 시험 방법들만으로써는 프레임웍의 변경부위에 대한 효과적인 시험을 어렵게 하는 여러 가지 고유한 문제들을 지니고 있다. 프레임웍에 합성된 확장부위의 실행은 흔히 프레임웍에 의해 제어된다. 이는 시험 초기 조건의 설정과 테스트 데이타의 입력과 같은 시험에 필요한 확장부위의 제어를 어렵게 한다. 프레임웍의 확장부위에 대한 시험 실행 시 실행 시점의 예측이나 제어가 어렵고 시험 결과의 관찰이 어려워서 오동작이 발생 시점과 발생 지점에서 즉각적으로 발견하기 어렵다. 프레임웍 시험의 제어도와 관찰도를 높이기 위해서는 일반적으로 여러 가지 시험 지원 코드를 시험 대상 프레임웍에 추가하는 것이 필요하게 된다. 이때 시험 대상 프레임웍의 원래 코드에 변경이 가해지거나 프레임웍의 실행이 (시험지원코드에 의해) 간섭받음으로 인하여 시험 결과의 신뢰성이 저하되는 문제가 발생하게 된다. 본 연구의 목적은 프레임웍 본래의 코드와 기능에 영향을 주지 않으면서 프레임웍의 시험성을 강화할 수 있도록, 시험 지원 코드를 시험 지원 기능 별로 컴포넌트화하여 프레임웍에 내장하는 방법을 찾아내는 것이다.

본 논문에서는 프레임웍이 개조, 확장되었을 때 발생한 결함들이 시험을 통하여 효과적으로 발견될 수 있도록 프레임웍의 가변 부위에 시험성을 높이는 테스터 컴포넌트들을 BIT(Built-in Test)로 내장하는 방법을 기술한다. 본 논문의 핵심은 프레임웍의 시험성을 안전하게 강화하는 방법이다. 즉, 여기서 제안된 설계 방법에 따라 BIT로 패키징되어 프레임웍에 내장된 테스터 컴포넌트들은 개조, 확장된 프레임웍의 시험 시, 프레임웍 코드의 변경이나 실행에 간섭 없이 시험에 필요한 제어와 관찰을 용이하게 한다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 관련 연구들을 살펴본다. 3장에서는 프레임웍의 시험성 요소들을 기술한다. 4장에서는 시험성을 높이는 테스터 컴포넌트들의 유형과 이들을 프레임웍에 부착 또는 내장한 아키텍쳐를 제안한다. 5장에서는 4장에서 설명한 아키텍쳐에 따라 프레임웍의 가변부위에 테스터 컴포넌트들을 BIT로 설계, 내장하는 방법을 기술한다. 6장에서는 5장에서 제안된 BIT 내장 방법을 응용 프레임웍의 가변부위에 대한 시험 환경의 구축에 실제로 적용한 예를 설명한다. 7장에서는 본 연구의 결과를 분석하고 향후 연구 방향을 제시한다. 그리고 8장에서 결론을 맺는다.

2. 기존의 관련 연구

현재, 객체지향 프레임웍의 개발 방법과 응용 사례들은 풍부하게 소개되어 있다[1, 2, 3]. Gamma[4]나 Pree[21]가 소개한 설계 패턴들은 모두 프레임웍의 설계와 구현에 유용하다. 그러나 프레임웍의 시험 방법들은 거의 다루어지지 않고 있다. 객체지향 소프트웨어의 시험 방법들도 많이 소개되어 있다[7, 8]. 예를 들면, ASTOOT [9]는 대수 기반의 클래스 시험 방법과 지원도구를, Class Bench[10]는 상태 기반의 클래스 시험 방법과 지원환경을 제공한다. 그리고 클래스 계층 구조(class hierarchy)에 대한 점증적인 시험 방법[11]이나 객체지향 소프트웨어의 통합 시험 방법들[12]도 소개되어 있다.

개조된 프레임웍의 시험의 어려움은 잘 알려져 있으며 이에 대한 해결 방안들이 문헌에 소개되어 있다[5, 8, 13, 14]. Fayad, et al.[14]은 프레임웍에 테스트 케이스를 생성하는 코드를 BITs(Built-in Tests) 클래스들로 내장하여 프레임웍이 개조, 확장될 때 함께 상속, 개조되어 프레임웍의 시험에 재사용하는 방식을 제시하였다. Binder[8, 15]는 객체지향 소프트웨어의 시험성을 높이는 테스트 지원 환경의 구축 방법과 테스트 설계 패턴들을 상당히 포괄적으로 제시하였다. Binder는 또한

프레임웍의 초기 시험과 재사용된 프레임웍의 시험 방법을 제시하였다[8]. 이들은 모두 프레임웍의 시험에 유용하지만 본 논문에서 다룬 프레임웍의 변경부위에 대한 시험 방법은 구체적으로 제시하고 있지 않다.

3. 프레임웍의 시험성 요소

응용 소프트웨어의 개발을 위하여 개조, 확장된 프레임웍에 발생한 오류들이 시험을 통하여 효과적으로 발견되기 위해서는,

① 개조, 확장된 프레임웍의 시험에 필요한 테스트 케이스들을 효율적으로 생성하거나 재사용할 수 있어야 한다.
② 테스트 케이스의 시험 초기 조건의 설정과 테스트 데이타의 입력이 가능하여야 한다.
③ 시험 실행 시, 결함으로 인한 오동작(malfunction)의 흔적이 남겨져야 한다.
④ 오동작을 포함한 시험 결과가 외부로부터 관찰 가능하여야 한다.
⑤ 관찰된 시험 결과(observed actual result)와 예상 결과(expected result)로부터 오동작의 발생을 감지하여 시험의 성공 여부(pass or fail)를 판정할 수 있어야 한다.

위의 조건들은 모두 시험성과 관련이 있다. 소프트웨어 시험성(testability)은 시험을 통한 소프트웨어 결함의 발견의 용이성을 의미한다[15, 16, 17]. 소프트웨어의 시험성은 다양한 요인들에 의하여 영향을 받을 수 있다[15]. 본 논문에서는 프레임웍의 시험성을 위에서와 같이 프레임웍의 효과적인 시험에 직접적인 영향을 미치는 가용도(availability), 제어가능도(controllability), 민감도(sensitivity), 관찰가능도(observability), 그리고 오러클 정밀도(oracle precision)의 5가지 요소들로 구분한다. 위에서 언급한 5가지 시험성 요소들의 의미는 표 1과 같다.

효율적인 시험을 위해서는 프레임웍을 개발하거나 개조, 확장 시 프레임웍이 높은 시험성을 유지하도록 하여야 한다. 그러나 앞에서 기술한 바와 같이 프레임웍은 시험을 어렵게 하는 성질을 갖고 있다. 그리고 프레임웍은 시험성 외에도 신뢰성(reliability), 견고성(robustness), 유연성(flexibility), 모듈성(modularity), 성능(performance) 등과 같은 다른 중요한 품질 요소(quality factors)들도 높은 수준을 유지하여야 한다. 시험성은 일반적으로 다른 품질 요소들과 상호 보완적이지만 이와 같은 다른 품질 요구 조건들과 상충하는 성격들도 갖고 있다. 예를 들면, 프레임웍의 민감도가 높아지면 결함으로 인한 오동작이 빈번하게 발생하여 신뢰성과 견고성이 오히려 낮아질 수 있다. 또한, 프레임웍의 제어가능도와 관찰가능도가 높아지면 은닉된 정보가 노출되고 프레임웍의 컴포넌트들이 상호 간섭하거나 방해할 가능성이 높아져서 신뢰성, 유연성, 모듈성, 성능 등의 저하를 야기할 수 있다.

표 1 시험성 요소

시험성 요소	의 미
가용도 (availability)	요구된 테스트 충분 기준(test adequacy criteria)을 만족하는 테스트 케이스들을 효율적으로 생성하여 시험에 (재)사용할 수 있는 능력이다.
제어가능도 (controllability)	시험에 필요한 테스트 데이타의 입력, 사건의 발생, 초기 조건의 설정과 변경에 대한 제어 능력이다.
민감도 (sensitivity)	테스트의 실행 시 소프트웨어에 내재한 결함들이 감응하여 오동작의 결과를 남기는 능력이다. 즉, 소프트웨어의 결함이 테스트에 감응하는 민감도를 의미한다.
관찰가능도 (observability)	소프트웨어에 대한 테스트의 실행 결과를 외부에서 관찰할 수 있는 능력이다.
오러클정밀도 (oracle precision)	관찰된 시험 결과로부터 시험의 성공 여부를 판정할 수 있는 능력, 즉 오동작이나 고장의 발생을 감지하여 결함의 존재 여부를 판정할 수 있는 능력을 의미한다.

본 연구팀은 프레임웍의 효과적인 시험을 위한 시험성 요소를 위와 같이 5가지(가용도, 제어가능도, 민감도, 관찰가능도, 오러클 정밀도)로 구분하고 이들 각 시험성 요소를 지원하는 각 테스터 컴포넌트들(시험패턴 생성기, 시험제어기, 센서타이저, 시험감시기, 오러클)을 설계, 통합하여 프레임웍을 효과적으로 시험할 수 있는 프레임웍 시험 지원 환경을 구축하기 위한 연구를 진행하고 있다.

본 논문에서는 앞에서 설명한 바와 같은 프레임웍 시험의 어려움을 해소하기 위하여 프레임웍에 대한 시험의 제어와 관찰을 용이하게 하는 테스트 지원 코드들을 시험 대상 프레임웍과 분리된 일단의 테스터 컴포넌트들로 설계하여 프레임웍에 내장함으로써 프레임웍의 시험성을 높였다. 프레임웍 테스트 지원 코드는 시험 대상 프레임웍과 상호 간섭이 가능한 한 작도록 BIT 컴포넌트로 설계하여 시험성의 강화가 프레임웍의 다른 품질 수준들을 저하시키지 않도록 하였다. 프레임웍에 부착되는 테스터 컴포넌트들은 또한 프레임웍의 시험성을 시험 중에는 높이고 운영 중에는 낮출 수 있도록 필요에 따라 제어와 탈착이 가능하게 설계하였다.

4. 프레임웍의 시험성 강화를 위한 테스터 컴포넌트

프레임웍의 시험성을 강화시키기 위하여 본 논문에서 제안하는 테스트 지원 컴포넌트들의 유형은 표 2와 같다. 시험패턴 생성기는 프레임웍의 시험에 필요한 테스트 패턴 또는 테스트 케이스들을 생성하는 장치로서, FUT(Framework Under Test)의 테스트 가용도를 높여주는 역할을 한다. 시험 제어기는 FUT에 테스트 케이스에 대한 테스트 데이타의 입력과 시험 초기 조건의 설정을 제어하는 장치이다. 센서타이저는 FUT의 결함 부위가 시험에 의해 쉽게 감염(infection)되어 오동작의 흔적을 남길 수 있도록 FUT의 민감도를 높여주는 장치이다. 시험 감시기는 테스트 실행 결과를 감시하고 이를 예상 결과와 비교하여 FUT의 결함으로 인하여 발생한 오동작을 감지하는 장치이다. 오러클은 테스트 케이스에 기대된 시험의 사전, 사후 및 불변 조건(expected pre-, post-condition, invariant)들을 알려주어 시험의 성공 여부를 판정할 수 있게 하는 장치이다.

표 2 프레임웍의 테스트 지원 장치

유 형	역 할	시험성 강화요소
시험패턴 생성기 (test pattern generator)	프레임웍의 확장 부위의 시험에 필요한 테스트 패턴들을 생성한다.	가용도
시험 제어기 (test controller)	FUT(Framework under Test)에 테스트 케이스의 시험 초기 조건을 설정하고 테스트 데이타를 입력한다.	제 어 가능도
센서타이저 (test sensitizer)	테스트의 실행 과정 중에 감응하여 실행 흔적을 남긴다.	민감도
시험 감시기 (test monitor)	FUT에 대한 테스트 케이스의 시험 실행 결과를 관찰하고 오러클의 도움을 받아 관찰된 시험 결과의 성공 여부를 판정한다.	관 찰 가능도
시험 기록기 (test logger)	시험 실행 결과를 수집, 저장한다.	관 찰 가능도
오러클 (test oracle)	테스트 케이스에 기대된 시험의 사전, 사후 및 불변 조건(pre-, post-condition, invariant)을 판정한다.	오러클 정밀도

위와 같은 테스터 컴포넌트들은 FUT의 시험 대상 컴포넌트(CUT: Component under Test)에 BIT(Built-In Test)로 내장되거나 [그림 1], 별도의 테스터 컴포넌트들로 만들어져 FUT의 컴포넌트들과 연결된다 [그림 2]. BIT가 내장된 프레임웍 컴포넌트는 컴포넌트의 본래 기능을 제공하는 기능적 인터페이스 (functional interface) 외에 시험에 필요한 테스트 데이타와 제어의 입출력을 위한 테스트 인터페이스(test interface)를 추가로 제공하게 된다.

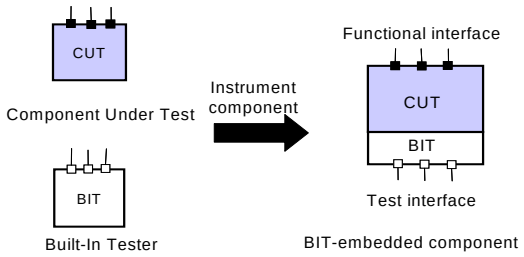

그림 1 Embedding BIT in CUT

BIT를 내장한 CUT의 기능적 인터페이스의 전체 또는 일부는 BIT의 테스트 인터페이스와 내부적으로 연결되어 있다. 이에 따라, BIT가 내장된 CUT는 외부와 상호 작용 시, 기능적 인터페이스의 입출력이 테스트 인터페이스를 경유하거나 컴포넌트에 내장된 BIT의 통제를 받을 수 있다. 프레임웍의 컴포넌트들과 별도로 구성되는 테스터 컴포넌트는 프레임웍 컴포넌트들과 연결되어 클러스터 수준의 프레임웍 시험 환경을 제공한다.

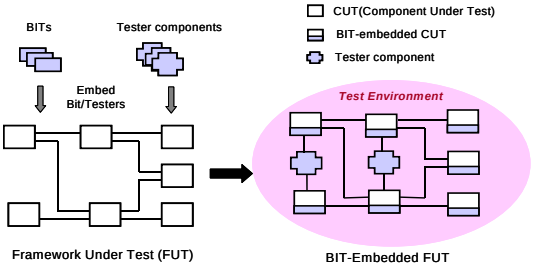

그림 2 Embedding BITs in a FUT

그림 3은 BIT가 내장된 테스트 대상 프레임웍 컴포넌트(CUT)와 CUT에 내장 또는 부착된 테스터 컴포넌트들 사이의 개념적인 자료 흐름을 보여준다. 먼저 시험 제어기는 CUT에 내장 또는 부착된 시험패턴 생성기, 센서타이저, 시험감시기, 오러클 및 시험기록기를 시험 준비 상태로 초기화한다. 그런 후 시험제어기는 CUT의 시험에 필요한 테스트 패턴들을 시험패턴 생성기로부터 test suite로 받아와서 이들을 CUT에 내장된 시험감시기에 시험 데이타로 차례로 입력, 실행시킨다.

시험 패턴 생성기는 CUT로부터 직접 테스트 패턴을 추출하는 것이 아니라 테스트 패턴 추출에 유용한 CUT 테스트 모델(예를 들면, 메소드 호출 순서 패턴 추출에 유용한 CUT 상태 머신 모델(state machine model))을 테스트 패턴 생성기 내부에 갖고 있어서 이로부터 테스트 패턴을 생성한다. 테스트 제어기는 태스트 패턴 생성기로부터 얻어낸 시험패턴(테스트 데이타 생성 정보)에 따라 테스트 데이타를 생성하여 CUT에 전달한다. 예를 들면 테스트 패턴이 CUT의 메소드 호출 순서 패턴일 때 이로부터 생성된 테스트 데이타는 메소드 호출 순서 패턴을 따르는 특정한 메소드 호출 시퀀스이다.

CUT의 실행 도중, 시험 결과의 분석에 중요한 단서가 되는 데이타(clue data)들은 CUT의 실행 전후에 센서타이저에 의해 감응 데이타(sensitized data)로 수집된다. 시험 예상 결과는 시험감시기로부터 시험 데이타를 넘겨받은 오러클에 의해 얻어진다. CUT에 내장된 오러클은 CUT의 시험 기대 결과(expected test result)인 실행 전후 조건(pre- and post-condition)과 불변조건(invariant)을 계산하여 그 결과를 시험감시기에게 알린다. 그러면 시험감시기는 감응 데이타를 포함한 시험 실행 결과(actual result)와 오러클로부터 얻어진 시험 기대 결과(expected result)를 비교하여 시험의 성공 여부(pass or fail)를 판정한다. 시험 성공 여부가 판정된 시험 결과는 시험감시기에 의해 CUT에 부착된 시험기록기에게 넘겨진다. 시험기록기는 시험감시기로부터 넘겨받은 시험 결과를 현재의 시험 정황(test context)에 맞게 기록, 저장한다.

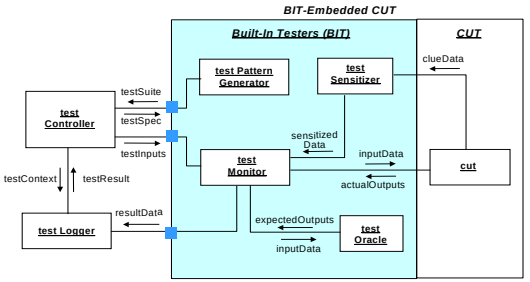

그림 3 BIT-Embedded CUT

위에서 설명한 시험감시기와 오러클에서는 시험 기대 결과의 계산은 오러클이, 시험 성공 여부의 판정은 시험감시기가 각각 분담하여 수행한다. 그러나 경우에 따라서는 오러클이 시험 성공 여부까지 판정하고 시험감시기는 CUT로부터는 시험 실행의 실제 결과를, 오러클로부터는 판정된 시험 성공 여부 결과를 수집, 감시하도록 설계할 수도 있다. 또한 CUT가 상태 의존적인 행위(state-dependent behavior)를 갖는 경우, 오러클은 CUT의 실행 결과로서 기대된 CUT의 상태 변화를 추적, 기억하여 시험 기대 결과의 계산에 사용한다.

CUT에 대한 사전, 사후 및 불변 조건들은 CUT의 계약 명세(contract specification)를 테스터 컴포넌트들에 의해 검사가 가능한 일단의 assertion들로 캡슐화되어, assertion의 추상화 수준, 강도, 성격에 따라 센서타이저, 시험관측기 및 오러클에 분산 배치된다. 이와 같이 CUT에 내장 또는 부착된 테스터 컴포넌트들은 개조, 확장된 프레임웍의 컴포넌트들이 시험 실행 시 약속된 계약을 서로 준수하는지 여부를 assertion들을 통하여 감시할 수 있게 함으로써 프레임웍의 시험을 지원한다. 이러한 assertion들의 유형, 표기 방법, 삽입 위치 등은 기존의 연구 결과들이 나와 있다[18, 19, 20].

개조, 확장된 프레임웍에 대한 클러스터 수준의 시험을 위해서는 시험에 관여된 각각의 프레임웍 컴포넌트들에 시험감시기, 센서타이저, 오러클을 BIT로 내장한다. 그리고 이렇게 BIT가 내장된 CUT들을, 이들에 대한 클러스터 수준의 시험을 지원하는 시험 제어기와 시험 기록기를 그림 4와 같이 부착, 연결함으로써 프레임웍 시험 환경을 구축한다. 그림 4에서, 시험제어기, 시험기록기, CUT에 내장된 BIT, 그리고 CUT들 사이의 바인딩은 동적으로 이루어지며, 이에 따라 FUT에 내장 또는 부착된 테스터 컴포넌트들은 필요에 따라 개별적인 제어와 탈착이 가능하다.

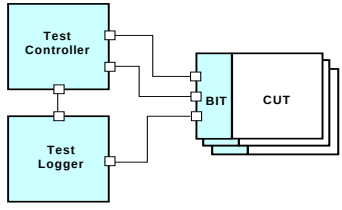

그림 4 BIT-Embedded FUT

5. 프레임웍의 가변 부위에 BIT 내장하는 방법

5.1 프레임웍의 가변 부위

프레임웍은 특정 응용 도메인에 변경없이 공통적으로 사용되도록 설계된 고정 부위(frozen spot)들과 응용 문제에 따라 개조와 확장이 가능하도록 설계된 가변 부위(hot spot)들로 구성되어 있다[21, 22]. 프레임웍은 또한

가변 부위를 개조 또는 확장할 때 준수해야 할 시스템 구성 요소들 사이의 결합 및 상호작용 규칙들을 포함하고 있다. 프레임웍의 구성 요소들 사이의 결합 및 상호작용 규칙들은 계약에 의한 설계(design by contract) 원리[18, 23, 24]를 사용하여 엄밀하게 정의할 수 있다.

객체지향 프레임웍에서는 고정 부위들과 가변 부위들이 클래스 또는 메소드 단위로 캡슐화되어 설계된다. 프레임웍의 고정 부위에 해당하는 클래스(메소드)를 템플릿 클래스(메소드)라고 부르고 고정 부위와 접속된 가변 부위에 해당하는 클래스(메소드)를 후크 클래스(메소드)라고 부른다 [21, 22].

템플릿 클래스와 후크 클래스는 서로 다른 클래스일 수도 있고 동일한 클래스일 수도 있다. 전자의 경우에는, 후크 클래스의 객체를 참조하는 인스턴스 변수(instance variable)를 통하여 템플릿 클래스에 후크 클래스의 서브 클래스 객체를 합성함으로써 가변 부위인 후크 클래스가 개조, 확장된다. 후자의 경우에는, 하나의 클래스 내에 템플릿 메소드와 후크 메소드가 함께 정의되어 있어서, 가변 부위인 후크 메소드가 서브클래스에 의해 개조, 확장된다.

Pree는 객체지향 프레임웍의 컴포넌트들 사이의 합성 패턴들을 1) 템플릿 클래스 객체에 합성되는 후크 클래스 객체의 개수(복수 허용 여부)와, 2) 상호 합성된 템플릿 클래스와 후크 클래스 사이의 상속 관계 여부의 기준에 따라 7개의 합성 패턴들로 추상화하여 분류하였다[3]. 객체지향 프레임웍은 이러한 합성 패턴에 따라 합성된 템플릿 클래스 객체들과 후크 클래스 객체들 사이에 허용 가능한 상호 작용들을 통하여 프레임웍의 기능들을 제공한다.

따라서 프레임웍의 기능성에 대한 시험은 템플릿 클래스들에 구현된 고정 기능들, 후크 클래스들에 허용 가능한 가변 기능들, 그리고 템플릿 클래스들과 후크 클래스들 사이의 상호작용을 통하여 구현된 기능들을 모두 포함하여 시험하여야 한다. 특히, 프레임웍의 가변 부위가 개조, 확장되면 1) 프레임웍의 가변 부위인 후크 클래스를 개조, 확장함으로써 구현된 가변 기능들과 2) 이에 영향을 받는 프레임웍의 고정 부위와 확장 부위 사이의 상호 작용들을 (재)시험하여야 한다.

5.2 프레임웍의 시험 방법

객체지향 프레임웍의 기능성은 프레임웍의 허용 가능한 합성 패턴에 따라 합성된 고정부위들과 가변부위들 간의 상호작용을 통하여 구현된다.

프레임웍의 가변부위는 개조, 확장되어 확장부위로 구체적으로 구현된다. 그리고 실행 시(run-time)시, 가변부위는 동적 바인딩(dynamic binding)에 의해 확장부위로 대체(substitution)된다. 따라서 프레임웍의 기능성은 실행 시 고정부위에 플러그인(plug-in)된 확장부위와의 상호작용을 통해 구현된다. 그리고 실행 시, 프레임웍의 기능성이 올바르게 구현되려면 계약으로 명시한 프레임웍의 고정부위와 가변부위 간의 상호작용 규칙을 가변부위를 개조, 확장한 확장부위가 준수해야만 한다.

따라서 본 논문에서 제안한 프레임웍 시험 방법은 프레임웍의 고정부위(템플릿 클래스) 객체들과 가변부위(후크 클래스)를 개조, 확장하여 구체적으로 구현한 확장부위 객체들이 합성되어 구성된 객체 구조(object structure)에 일련의 테스트케이스들을 입력하여 확장부위 객체들이 가변부위에 명시된 계약 명세를 준수하는지 여부를 시험하는 것이다.

5.3 BIT가 내장된 프레임웍 가변 부위

본 장에서는 객체지향 프레임웍의 가변 부위가 개조, 확장되었을 때 프레임웍의 고정 부위와 확장 부위 사이의 상호 작용이 가변 부위에 허용된 계약 명세를 준수하는지 여부를 시험하기 위하여 4장에서 설명한 테스터 컴포넌트들을 프레임웍의 확장 부위에 BIT로 설계, 내장하는 방법을 기술한다.

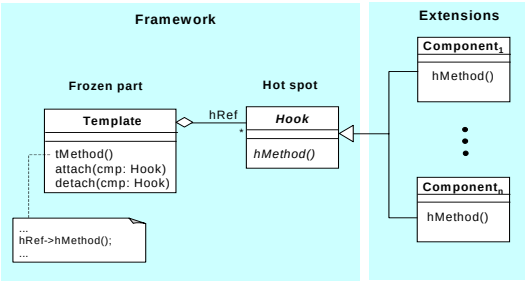

그림 5 Hot Spots of a framework

그림 5는 템플릿 클래스 객체에 후크 클래스 객체가 1:n으로 연결되는 합성 패턴을 갖는 프레임웍 가변 부위를 보여준다. 그림 5의 합성 패턴을 갖는 템플릿 클래스의 객체는 자신의 인스턴스 변수를 통하여 후크 클래스의 서브 클래스에 속한 객체와 합성된다[21]. 그림 5에서 고정, 가변 및 확장 부위는 다음과 같다.

합성 패턴: 1:n connection
고정 부위: Template (템플릿 클래스)
가변 부위: Hook (후크 클래스)
연결 부위: hRef (인스턴스 변수)
확장 부위: Component$_i$ (후크 서브클래스)

프레임웍의 가변 부위가 그림 5와 같은 합성 패턴에 따라 확장되면, 템플릿 클래스의 고정 부위와 이와 연결된 후크 클래스가 서브 클래스로 개조, 확장된 가변 부위 사이의 상호작용을 (재)시험하여야 한다. 그림 6은 그림 5에서의 가변 부위인 Hook 클래스가 서브 클래스로 확장된 컴포넌트를 시험 대상 컴포넌트(CUT)로 하여 테스터 컴포넌트들을 프레임웍에 부착한 합성 구조를 보여준다. 그림 6에서 음영 부분에 놓인 클래스들이 프레임웍 가변 부위에 내장 또는 부착되는 테스터 컴포넌트들을 나타낸다.

그림 6에서 프레임웍 가변부위에 내장되는 테스터 컴포넌트들은 CutMonitor, TestController, CutOracle, CutSensitizer, TestLogger이다. CutMonitor는 시험의 관찰도(observability)를 높이기 위하여, 가변부위의 시험대상 클래스 객체(Cut)의 시험 실제 결과(actual result)와 기대 결과(expected result)를 각각 Cut(또는 CutSensitizer)와 CutOracle로부터 얻어내어 이를 성공 여부에 대한 판정 결과와 함께 TestLogger에 넘긴다. TestController는 시험의 제어가능도(controllability)를 높이기 위하여, 가변부위를 시험 초기 조건에 맞게 설정하고 템플릿 메소드를 시험 순서대로 호출하고 수집된 시험 결과를 TestLogger로부터 얻어낸다. CutOracle은 시험의 오러클 정밀도(oracle precision)를 높이기 위하여, Cut의 기대 결과(사전, 사후 조건 등)를 계산하여 CutMonitor에게 알려준다. CutSensitizer는 시험의 민감도와 관찰도를 높이기 위하여, Cut의 실행 초기 상태, 실행 중간 상태 및 외부에서 접근이 금지된 private 상태 값 등을 포착하여 CutMonitor에게 알려준다. TestLogger는 CutMonitor로부터 넘겨받은 시험 결과를 수집, 기록, 저장하여 시험 결과의 확인, 특히 실패(fail)로 판정 난 시험 결과에 대한 검색을 지원한다.

그림 6에서 TestController와 TestLogger는 Cut의 외부로부터 Cut에 부착되는 테스터 컴포넌트들이다. 나머지 테스터 컴포넌트들은 Cut에 내장되는 BIT들이다. Tester 클래스는 Cut에 내장된 BIT 컴포넌트들에 대한 추상 클래스로서, Hook 클래스를 통하여 Template 클래스 객체에게 Cut와 동일한 인터페이스를 제공한다. Cut의 BIT 테스터 컴포넌트들인 센서타이저(CutSensitizer)와 시험감시기(TestMonitor)는 Hook 클래스의 서브 클래스로 정의된 Tester 클래스의 서브 클래스들로 설계되어 있다. BIT들과 Cut는 Tester 클래스의 인스턴스 변수인 tCut를 통하여 연결된 일련의 체인 구조를 형성한다. Template 객체는 자신의 hCut 변수가 참조하는 Hook 클래스 인터페이스를 통하여, 이러한 체

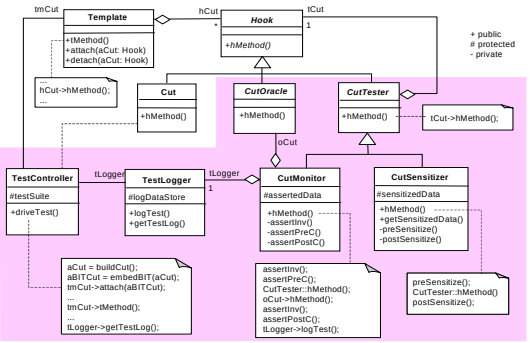

그림 6 Class Structure of BIT-Embedded Framework Hot Spot

인 구조로 BIT가 내장된 Cut에 부착된다. 이렇게 부착된 Template 객체는 일련의 BIT 컴포넌트들을 경유하여 체인의 끝에 놓인 Cut와 연결된다.

Cut에 체인 구조로 내장된 각각의 테스터 컴포넌트들은 자신의 hMethod가 호출되면 이를 자신의 tCut 변수를 통하여 연결된 후속(successor) 컴포넌트에게 위임하기(forwarding) 위한 hMethod의 호출 전후에 자신에게 부과된 시험 기능을 수행한다. 하지만 이러한 과정에서 Template 객체와 Cut 객체 사이의 상호 작용은 Cut에 부착된 테스터 컴포넌트들의 간섭을 받지 않는다. 이러한 테스터 컴포넌트들의 설계 패턴은 후크 클래스의 서브 클래스인 Cut에 대한 Decorator 설계 패턴[4]과 유사하다. 이렇게 내장된 테스터 컴포넌트들은 Cut에 대한 시험 기능을 분담하여 수행한다. 이 중 가장 바깥 쪽에 부착된 시험감시기는 템플릿 객체의 Cut에 대한 접근을 감시하는 Proxy[4]의 역할을 담당한다.

그림 6에서 센서타이저는 tCut의 hMethod의 호출 전후에 Cut의 시험에 단서가 되는 데이타를 수집한다. 예를 들면, Cut가 호출되기 직전의 상태를 보관하여 호출 후 사후 조건과 불변 조건의 검사가 가능하게 한다. 시험감시기(CutMonitor)는 Cut의 후크 메소드 호출 전후의 조건(pre-, post-condition) 및 불변 조건들을 검사하여 계약의 위반 여부를 감시한다. 포착된 시험 결과들은 testLogger에 의해 수집, 기록된다.

시험 감시기에 의해 참조되는 cutOracle은 Cut의 후크 메소드의 실행 후의 기대 결과(expected result)를 계산하는 오러클의 역할을 수행한다. 그림 6에서 시험감시기는 cutOracle로부터 얻어진 기대 결과를 실제 결과(actual result)와 함께 사용하여 Cut에 대한 사전, 사후, 불변 조건의 위반 여부를 계산함으로써 시험의 성공

여부를 판정한다. 그러나 경우에 따라서는 오러클이 Cut의 사전, 사후 및 불변 조건의 만족 여부를 계산하도록 오러클의 역할을 강화할 수 있다. 예를 들면, Cut가 상태 의존적인(state-dependent) 복잡한 행위를 가질 경우 Cut에 기대된 상태 변화(expected state changes)를 오러클이 계산, 추적하게 함으로써 상태 의존적인 계약 조건의 준수 여부의 판정을 오러클에 일임할 수 있다.

프레임웍의 가변 부위의 시험에 필요한 초기 조건의 설정과 테스트 데이타의 입력은 testController에 의해 제어된다. testController는 시험 시작 전, Cut의 객체 구조(object structure) 인스턴스와 BIT 컴포넌트들을 생성하고 생성된 BIT들을 Cut 인스턴스에 내장한다. 그런 후 각각의 테스트 케이스 또는 test suite의 시험에 요구된 초기 조건으로 BIT가 내장된 Cut와 testLogger를 초기화한다.

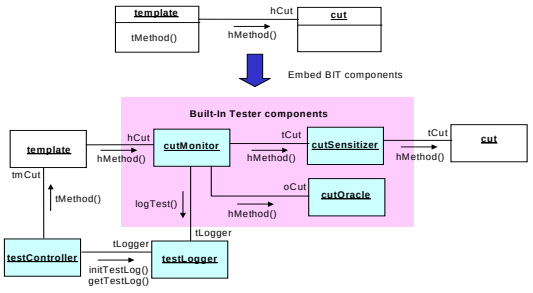

그림 7 An Object Structure of BIT-Embedded Framework Hot spot

그림 7은 그림 6의 설계 패턴에 따라 템플릿 객체와 CUT 객체 사이에 시험 감시기, 센서타이저 및 오러클을 내장한 객체 구조(object structure)를 보여준다. 그림 8은 그림 7의 객체 구조에서 템플릿 객체의 tMethod가 시험 제어기에 의해 호출되었을 때 템플릿, CUT 및 테스터 객체들 사이의 일련의 상호작용을 보여준다.

그림 8에서, 호출된 템플릿 객체의 tMethod는 hCut 변수를 참조하여 후크 클래스 타입 객체의 hMethod를 호출한다. hMethod의 호출은 시험감시기와 센서타이저를 경유하여 cut의 hMethod에 도달하여 실행되고 실행 결과는 다시 센서타이저와 시험감시기를 경유하여 템플릿 객체의 tMethod로 리턴된다. 시험감시기는 센서타이저의 hMethod 호출 전후에 오러클의 도움을 얻어 cut의 사전, 사후 및 불변조건의 위반 여부를 검사하고 그 결과를 testLogger에게 넘긴다. 센서타이저는 cut의 hMethod 호출 전후에 cut의 사전, 사후 상태를 포착하여 저장한다. 이러한 과정 중에 시험감시기, 센서타이저 및 오러클은 template 객체와 cut 사이의 상호 작용에 필요한 시험 기능을 그러한 상호 작용에 간섭하지 않고 수행하게 된다.

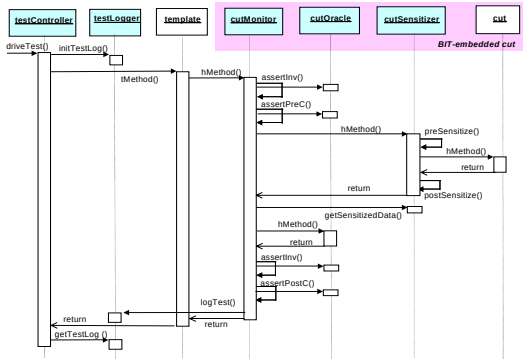

그림 8 A Sequence Diagram of BIT-Embedded Framework Hot Spot

그림 7과 그림 8에서 예로 든 시험 대상 가변부위(hot spot)의 객체 구조는 템플릿 클래스에 하나의 후크 클래스 객체가 합성된 경우이다. 그러나 본 논문의 BIT 내장 방식은 위와 같이 템플릿, 후크 객체가 1:1로 합성된 가변부위(hot spot)뿐만 아니라 여러 개의 템플릿, 후크 객체들로 합성된 가변부위에 대한 클러스터 수준의 시험도 지원한다. 예를 들면, 그림 6에서와 같은 1:n connection 합성 패턴에 따라 템플릿 객체에 여러 개의 후크 객체들이 합성된 객체 구조에서는 각각의 후크 객체에 BIT 컴포넌트들이 위에서 설명한 방식으로 내장된다. 그리고 시험 대상 후크 클래스가 템플릿 클래스의 서브 클래스로 정의되어 있어서 템플릿 객체와 후크 객체가 순환적으로 합성된(recursive object composition) 경우에도 각각의 후크 객체에 BIT 컴포넌트들이 위와 유사하게 내장된다.

또한 본 논문에서는 한 개의 후크 메소드에 대한 시험을 예로 들었지만 여러 개의 후크 메소드들로 구성된 후크 클래스와 템플릿 클래스의 상호작용에 대한 시험으로도 쉽게 확장이 가능하다. 이 경우, Cut에 내장된 각 BIT 컴포넌트에서 각각의 후크 메소드에 대한 처리를 그림 6의 hMethod의 처리와 유사하게 반복하여 정의한다.

본 장에서 설명한 프레임웍 가변부위에 내장되는 BIT 컴포넌트들은 시험감시기, 시험제어기, 오러클, 센서타이저, 그리고 시험기록기이다. 이들이 프레임웍의 시험성을 높이는 이유를 요약하면 다음과 같다. 시험감시기는 가변부위의 시험대상 클래스 객체(CUT)의 시험

실제 결과(actual result)와 기대 결과(expected result)를 시험기록기에 넘겨 기록함으로써 시험의 관찰도(observability)를 높여준다. 시험제어기는 가변부위를 시험 초기 조건에 맞게 설정하고 템플릿 메소드를 시험 순서대로 호출함으로써 시험의 제어가능도(controllability)를 높여준다. 오러클은 CUT의 기대 결과를 계산하여 시험감시기에게 알려줌으로써 시험의 오러클 정밀도(oracle precision)을 높여준다. 오러클 정밀도의 구체적인 향상 정도는 오러클에 구현된 기대 결과 계산의 정밀도에 따라 달라지지만 오러클이 내장되지 않은 경우보다는 높아진다. 센서타이저는 CUT의 실행 초기 상태, 실행 중간 상태 및 외부에서 접근이 금지된 private 상태 값 등을 포착하여 시험감시기에게 알려줌으로써 시험의 민감도와 관찰도를 높여준다. 제어도, 관찰도, 민감도는 시험성의 주요 요인으로 작용하므로 이들의 강화는 시험성의 강화를 가져온다.

이러한 테스터 컴포넌트들이 프레임웍에 안전하게 내장되는 이유를 요약하면 다음과 같다. 본 장에서 제안한 BITs(Built-in Tests) 컴포넌트들 중 시험감시기, 오러클, 센서타이저는 이들이 내장된 가변부위의 템플릿 객체가 후크 클래스 객체(CUT)의 메소드 호출 시, CUT의 본래 기능은 CUT가 실행할 수 있게 CUT에게 전달하여 실행하게 하고, 후크 클래스 CUT의 실행 전후에 시험 전용 기능만을 수행한다(그림 8 참조). 따라서 이들 BITs(Built-in Tests) 컴포넌트들은 이들을 내장한 프레임웍의 본래 기능에 영향을 주지 않으면서 시험 지원 기능을 수행하게 된다. 시험제어기와 시험기록기는 프레임웍 내부에 내장되는 것이 아니라 프레임웍 외부 시험 환경의 일부로 구성되어 시험 대상 프레임웍과 연결되어 작동하므로 역시 프레임웍 자체의 기능 수행에 영향을 주지 않는다. 시험패턴 생성기는 프레임웍 내부에 내장될 수 있으나 시험 대상 프레임웍과 직접 상호작용은 하지 않고 시험제어기에 의해서만 구동하므로 역시 프레임웍의 실행에 영향을 주지 않는다.

6. 적용 예

본 장에서는 지금까지 설명한 프레임웍의 가변부위에 테스터 컴포넌트들을 부착, 내장하는 방법의 타당성과 효용성을 시험하기 위하여 5장에서 기술한 설계 방법에 따라 실제 프레임웍의 가변부위에 테스터 컴포넌트들을 구현, 내장하여 시험 지원 환경을 구축한 예를 설명한다.

6.1 경보 감시 시스템(Alarm Monitoring System)

여기서 예로 든 시험 대상 프레임웍은 제어 공정의 상태를 감시하는 시스템 응용 분야에 사용하기 위하여 본 연구팀에 의해 개발된 경보 감시 프레임웍(alarm monitoring framework)이다. 본 프레임웍은 외부 환경의 상태 변화를 감시하여 비정상적인 상태로의 변화를 경보로 알려주는 경보 감시 시스템(alarm monitoring system)[그림 9]으로 확장 가능한 응용 프레임웍이다.

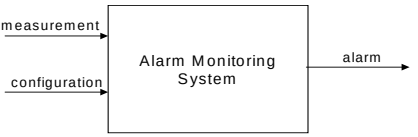

그림 9 Alarm Monitoring System

그림 10은 본 경보감시 프레임웍의 클래스들의 합성 구조의 일부를 나타낸 클래스 다이어그램이다.

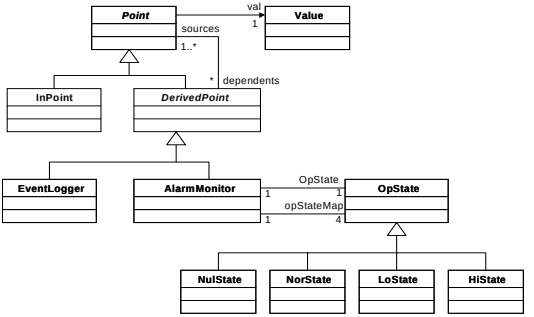

그림 10 Partial Class Structure of the Alarm Monioring Framework

그림 10에서 InPoint는 외부로부터 계측값을 직접 입력 받는 클래스이다. DerivedPoint는 시스템 내부의 다른 Point 클래스 객체들로부터 얻어진 계측값들을 사용하여 새로운 계측값 또는 상태값을 계산, 처리하는 클래스이다. AlarmMonitor는 DerivedPoint의 서브 클래스로서 계측값의 변동에 따른 정상 또는 비정상 상태로의 변화를 감지하여 비정상 상태로의 변화 시 이를 경보로 알려주는 클래스이다. AlarmMonitor는 Point 클래스에 객체 합성되는 후크 클래스로서 응용에 따라 다양한 서브클래스로 확장된다. 그림 10에서는 4개의 State 객체들을 구성 요소로 갖는 State 패턴[4]으로 설계된 AlarmMonitor를 보여준다. 본 논문에서는 그림 10의 프레임웍에서 Point 클래스 객체에 후크 클래스 객체로 합성된 AlarmMonitor 클래스 객체를 Cut로 하여 테스터 컴포넌트들을 부착한 예를 설명한다.

여기서 예로 든 AlarmMonitor는 그림 11의 state-

chart에 명시된 동적 행위를 갖는다. 그림 11에서와 같이 AlarmMonitor는 자신의 update() 오퍼레이션이 호출될 때 파라미터로 받은 계측값이 설정된 상하한치를 초과하게 되면 이를 high alarm 또는 low alarm으로 경보를 발생시킨다. 그리고 계측값이 상하한 경계 사이에서 동요할 때 발생하는 반복적인 경보를 피하기 위한 deadband를 갖는다.

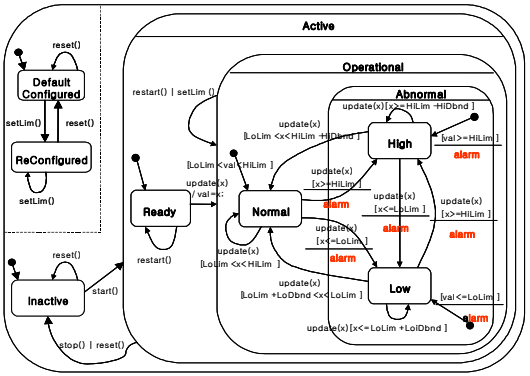

그림 11 Object Behavior of the AlarmMonitor Class

그림 12는 계측값의 변동에 따른 상태 변화와 경보 발생의 시점들을 보여주는 한 예이다.

그림 10의 프레임웍을 사용하여 여러 가지 다양한 경보 감시 시스템들을 구현할 수 있다. 예를 들면, 그림 13은 2개의 서로 다른 입력 소스(input source)로부터 입력된 계측값들에 대한 개별적인 경보 감시를 수행하는 시스템을 보여준다. 그림 14는 하나의 입력 소스(input source)로부터 일련의 계측값들이 순차적으로 입력될 때 이에 관여된 경보 감시 시스템 객체들 사이의 상호작용을 보여준다.

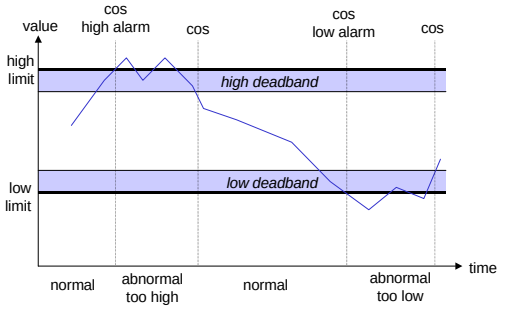

그림 12 State Changes as the measured value changes

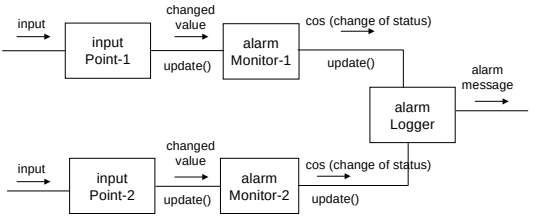

그림 13 An Object Structure of BIT-Embedded Alarm Monitoring System

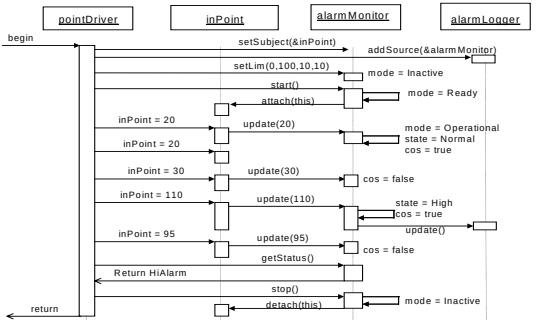

그림 14 A Sequence Diagram of Alarm Monitoring System

6.2 BIT의 설계와 내장 예

지금까지 설명한 경보 감시 프레임웍(alarm monitoring framework)의 InPoint 클래스에 DerivedPoint의 인터페이스를 통하여 후크 클래스로 합성되는 Alarm Monitor 클래스를 Cut로 하여 테스터 컴포넌트들을 구현하였다. 그림 15는 이와 같이 AlarmMonitor에 테스터 컴포넌트들을 BIT로 설계하여 Cut에 내장한 설계 구조를 보여준다. 그림 15에서 BIT로 구현된 테스터 컴포넌트들은 CutMonitor, CutSensitizer, 그리고 CutOracle이다. CutMonitor와 CutSensitizer는 이들의 상위 클래스인 CutTester의 인스턴스 변수 tCut를 통하여 서로 연결되어 AlarmMonitor에 내장된다. CutSensitizer는 tCut의 update() 함수 호출 전후에 AlarmMonitor의 내부 상태를 검사하여 보관한다. CutOracle은 그림 11에 명시된 상태 행위(state behavior)를 시뮬레이션(simulation)하도록 구현하였다. CutMonitor는 Cut Sensitizer로부터 얻어지는 Cut의 실제 행위(actual behavior)와 CutOracle로부터 얻어지는 기대 행위(expected behavior)를 사용하여 cut의 사전, 사후, 불변조건의 위반 여부를 계산하도록 구현하였다. 그리고 시험 대상 프레임웍에 시험 초기 조건을 설정하고 테스트

데이타를 입력하는 테스트 드라이버(test driver)와 시험 결과를 수집, 기록하는 시험기록기(test logger)가 BIT가 내장된 시험 대상 프레임웍에 부착될 수 있는 형태로 구현되었다.

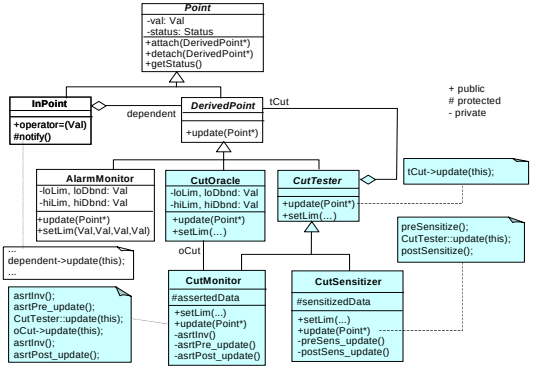

그림 15 Class Structure of BIT-Embedded Alarm Monitor

이 외에도 AlamrmMonitor의 테스트 케이스들을 오프라인(off-line)으로 생성하는 시험패턴 생성기가 구현되었다. 시험패턴 생성기는 계측값을 파라미터로 갖는 update()함수 호출의 시퀀스를 테스트 케이스들로 생성한다. 이 때 테스트 패턴은 그림 11의 statecharts의 Operational 모드(mode) 내에서 이전과 동일한 상태로 회귀될 때까지 허용 가능한 상태변화 시퀀스를 경로들로 갖는 상태 전이 신장 트리(state transition spanning tree)의 각 경로들이다. 시험패턴 생성기는 앞에서 언급한 바와 같이 Cut에 BIT로 내장되어야 하지만 여기서 오프라인(off-line) 테스트 도구로 구현하였다.

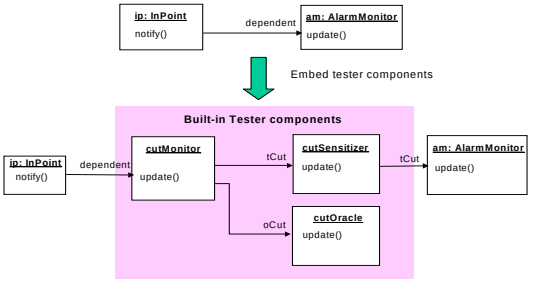

그림 16 Embedding BIT in an AlarmMonitor

구현된 테스터 컴포넌트들의 객체 생성(instantiation)과 이들을 Cut인 AlarmMonitor 객체에 부착, 내장하는 작업은 테스트 드라이버(test driver)에 의해 시험 실행 초기에 이루어진다. Cut에 내장된 BIT 컴포넌트들은 동적으로 탈착(detach/attach)될 수 있다. InPoint 객체에 BIT가 내장된 AlarmMonitor가 부착될 때 Alarm Monitor에 내장된 BIT들도 함께 부착된다. 이와 같은 테스터 컴포넌트들의 구현과 시험 대상 프레임웍으로의 통합은 시험 대상 프레임웍의 코드를 전혀 변경하지 않고 이루어졌다.

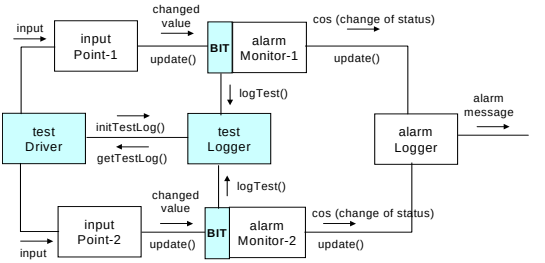

그림 17 A Possible Object Structure of BIT-Embedded Alarm Monitoring System

그림 15와 같이 설계된 테스터 컴포넌트 객체들은 InPoint 객체에 후크 클래스 객체로 합성되는 Alarm Monitor 객체에 그림 16과 같이 내장된다. 그림 17은 그림 13과 같이 구성된 경보 감시 시스템(alarm monitoring system)의 각 AlarmMonitor 객체에 BIT 컴포넌트들이 위와 같은 방법으로 내장된 시스템을 보여준다. 이와 같이 테스터 컴포넌트들이 내장 또는 부착된 경보 감시 시스템은 시험 시 테스터 컴포넌트들에 의해 시험 조건의 설정과 시험 결과의 감시가 경보감시의 본래의 기능에 간섭 없이 투명하게(transparently) 이루어진다.

6.3 BIT가 내장된 프레임웍의 시험 예

이와 같이 테스터 컴포넌트들이 부착, 내장된 경보 감시 시스템의 시험 시, 테스터 컴포넌트들이 제대로 작동되는지를 시험하기 위하여 즉각적으로 발견되지 않는 잘못된 상태 변화를 유발하는 오류들을 AlarmMonitor 객체에 심은 후 위에서 설명한 테스트 패턴들을 가지고 시험하였다. 시험은 BIT를 내장하지 않은 경우와 내장한 경우로 나누어 시행하였다. 그 결과 BIT가 내장된 시험에서는 알려진 모든 오류들이 발견되지는 않았지만 오류로 인한 잘못된 상태 변화들은 모두 시험기록기(test logger)에 의해 기록되었다. 이와 반면 BIT를 내장하지 아니한 시험에서는 정상 상태를 경보로 잘못 처리한 오류들에 한에서 경보기록기(alarm logger)에 의해 발견이 가능하였다.

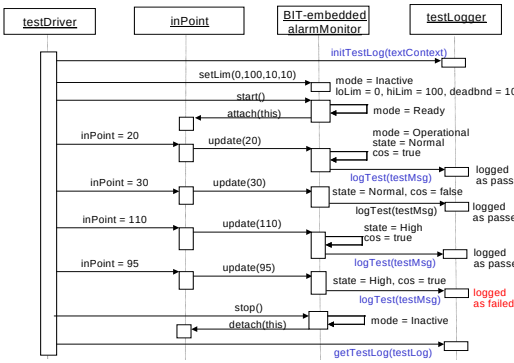

그림 18 A Test Scenario of BIT-Embedded Alarm Monitor

AlarmMonitor 객체의 오류로 인한 오동작이 BIT로 내장된 테스터 컴포넌트들에 의해 발견된 경우를 예를 들어 설명한다. 계측값이 상한값을 초과하여 현재 상태를 비정상 상태로 감지하고 있는 AlarmMonitor에 상한치보다는 작지만 deadband 내에 속한 계측값이 입력된 경우를 살펴보자. 이 경우 AlarmMonitor는 현재 상태를 계속 비정상 상태로 처리하여야 한다. 그런데 AlarmMonitor에 오류가 있어서 deadband를 고려하지 못하고 이를 정상 상태로의 복귀로 간주하였다고 가정하자. 그림 18은 이렇게 가정된 상황에서 testDriver, InPoint 객체, BIT가 내장된 AlarmMonitor, 그리고 testLogger 사이의 상호 작용을 보여준다. (그림 18에서 update() 함수의 호출 시 실제로는 이를 호출한 InPoint 객체의 참조(reference)가 파라미터로 넘겨지지만 여기서는 편의상 계측값을 파라미터로 나타내었다.) 그림 18에서 deadband 보다 작은 범위 내에 속한 정상 계측값인 95가 inPoint로부터 AlarmMonitor로 넘겨졌을 때 이를 정상 상태로의 복귀로 잘못 처리한 결과가 AlarmMonitor에 내장된 BIT에 의해 감지되어 testLogger를 통해 시험 실패(failed test)로 기록된다. AlarmMonitor는 비정상 상태로의 변화 시에만 경보를 발생하므로 이러한 성격의 오동작은 위와 같은 테스터 컴포넌트들의 지원 없이는 즉각적으로 발견되지 않는다.

7. 연구 결과의 분석 및 향후 연구 방향

본 논문은 기존의 객체지향 소프트웨어 시험 방법 연구들이 직접적으로 다루지 않은 프레임웍의 가변 부위에 대한 시험 방법을 연구하였다. 본 논문의 연구 성과는 프레임웍의 가변부위에 대한 시험 시 프레임웍 코드의 변경이나 실행에 간섭없이 시험의 제어와 관찰을 용이하게 하는 BIT 내장 설계 패턴의 도출이다. 본 논문의 연구 결과와 가장 밀접한 기존의 관련 연구는 Hoffman의 ClassBench[10]와 Fayad의 Built-in Tests(BITs)[14]이다. ClassBench는 시험 대상 클래스에 대한 테스트 케이스의 자동 생성과 오러클의 내장 방법을 제공한다. ClassBench가 클래스 단위 시험(class unit testing)을 지원하는데 비하여 본 연구는 프레임웍의 가변부위를 구성하는 템플릿-후크 클래스들 사이의 상호작용에 대한 시험을 지원한다. Fayad의 BITs는 클래스에 대한 테스트 케이스들을 BIT 클래스들로 구성, 내장하여 클래스들이 서브클래스들로 확장, 개조될 때 BIT 클래스들도 함께 상속되게 함으로써 테스트 케이스들의 효율적인 재사용을 지원한다. 이와 반면 본 논문의 BIT들은 테스트 케이스 이외에 시험의 감시, 제어, 오러클 등과 같은 테스트 지원 코드를 포함한다. Fayad의 BITs는 프레임웍의 후크 클래스들이 서브클래스들로 확장될 때에도 적용이 가능하므로 본 논문의 BITs(Built-in Tests) 컴포넌트들 중 하나인 시험 패턴 생성기를 이를 내장한 후크 클래스의 확장 시 함께 상속되어 재사용 가능하도록 보완하는데 유용할 것으로 판단한다.

6장에서의 실험 결과는 본 논문에서 제안한 BIT 내장 설계 패턴에 따라 만들어진 테스트 컴포넌트들이 개조, 확장된 실제 프레임웍의 시험성을 높이는데 효과적으로 사용될 수 있음을 보여준다. 하지만 본 논문에서 제안된 BIT 내장 방식이 위에서 예로 든 구현 사례보다 훨씬 복잡하고 방대한 프레임웍의 시험을 지원하는데 확대 적용(scale up)될 수 있기 위해서는 더 많은 적용 사례들과 아래에 설명한 고려 사항들을 반영한 개선 노력이 필요하다.

본 논문에서 제안한 BITs(Built-in Tests) 컴포넌트들은 이들을 내장한 프레임웍 기능에는 영향을 주지 않지만 프레임웍의 성능, 실시간 제약, 동기화 등에는 영향을 줄 수 있다. 따라서 이러한 기능 외적인 프레임웍 성질에도 영향을 주지 않는 방법에 대한 연구가 필요하다.

본 논문에서 제시한 설계 패턴의 테스터 컴포넌트들은 시험 대상 프레임웍에 따라 이에 맞는 구체적인 형태로 설계, 구현된다. 그런데 특정 프레임웍을 개조, 확장하여 시험할 때마다 이에 필요한 테스터 컴포넌트들을 개별적으로 구현하여 시험 대상 프레임웍에 내장하는 것은 시험 환경의 구축과 유지보수에 따르는 시험 비용이 너무 높아지는 문제를 갖고 있다. 따라서 프레임웍의 테스터 컴포넌트들과 이들로 구성되는 프레임웍 시험 지원 환경을 보다 효율적으로 구축할 수 있는 방

법이 필요하다. 이를 위해서는 본 논문에서 제시된 테스터 컴포넌트 설계 패턴을 테스트 지원 프레임웍으로 확장, 구현하여 프레임웍의 시험 지원 환경의 효율적인 구축에 재사용 가능하게 하는 연구가 필요하다.

본 논문에서는 시험 기능 자체의 알고리즘보다는 부과된 시험 기능을 분담, 상호 협동하여 수행하는 BIT 컴포넌트들로 테스트 지원 코드를 설계하여 CUT에 내장하는 방법에 초점을 두고 있다. 즉, 프레임웍에 내장되는 각 테스터 컴포넌트들의 내부 설계보다는 1) 테스터 컴포넌트들의 인터페이스, 상호 연결 및 상호 작용 관계 2) 그리고 테스터 컴포넌트들과 이들을 내장한 프레임웍 컴포넌트들과의 상호 연결 및 상호 작용의 설계에 중점을 두고 있다. 예를 들면 테스트 케이스 생성 방법이나 테스트 오러클(test oracle)이 기대 행위(expected behavior)를 계산하는 방법 자체보다는 이러한 시험 기능들을 프레임웍의 가변부위에 BIT로 내장하는 방법을 설명하였다. 프레임웍에 내장된 BIT들이 결함의 발견에 실제적인 효과를 주기 위해서는 BIT에 부과된 테스트 케이스 생성과 오러클과 같은 시험 지원 기능들이 물론 BIT 내부에 구체적으로 고안, 구현되어야 한다. 테스트 오러클(test oracle)과 테스트케이스 생성 방법들은 기존의 문헌에 많이 소개되어 있다. 본 논문에서는 제안된 내장 방법의 타당성 실험을 위하여 이러한 알려진 방법들을 사용하여 테스터 컴포넌트들의 개별적인 시험 기능들을 간단하게 구현하였다. 현재, 객체지향 프레임웍의 가변부위에 대한 시험에 효과적인 테스트 오러클(test oracle)과 테스트케이스 생성 방법에 대한 연구가 본 논문의 연구 주제와 관련하여 진행 중에 있다. 진행 중인 테스트 오러클과 테스트케이스 생성 방법에 관한 연구 내용은 다음과 같다.

1) 테스트 오러클에 대한 연구

실행 시, 프레임웍의 기능성은 프레임웍의 고정부위와 가변부위를 확장한 확장 부위와의 상호작용으로 구현되므로 프레임웍의 시험성 강화를 위하여 프레임웍의 가변 부위를 개조, 확장한 확장부위가 가변부위에 명시된 계약 명세를 준수하는지 여부를 판단할 수 있는 메커니즘, 즉 테스트 오러클에 대한 연구, 개발이 요구된다.

프레임웍의 가변부위는 실행 시 고정부위에 플러그인(plug-in)될 확장부위가 준수해야 할 계약을 명시하므로, 프레임웍의 가변부위는 확장부위가 가변부위에 명시된 계약을 준수하는지 여부를 판단할 수 있는 기준(standard) 즉, 테스트 오러클의 역할을 수행한다. 그리고 프레임웍의 가변부 위는 확장부위로 개조, 확장되어 구현되므로 프레임웍의 가변부위에 대한 행위는 확장부위의 행위보다 상대적으로 추상적이다.

따라서 본 연구팀이 프레임웍의 시험성 강화를 위하여 연구, 개발 중에 있는 테스트 오러클은 주어진 테스트케이스들을 프레임웍의 가변부위와 확장부위에 동시에 입력하였을 때 구체적으로 구현된 확장부위의 행위가 생성한 실제값(actual value)이 Statecharts으로 표현된 추상적인 가변부위의 행위 명세에서 생성한 기대값(expected value)에 대응되는지 여부를 판단할 수 도구이다.

프레임웍에 입력된 일련의 테스트케이스들에 의해 확장부위의 행위가 생성한 실제값은 구체적으로 구현된 확장부위의 인스턴스 값들의 조합으로 표현되고, Statecharts로 표현된 가변부위의 행위 명세가 생성한 기대값은 Statecharts 상의 한 상태(state)로 표현된다.

따라서 본 연구팀이 연구, 개발 중에 있는 프레임웍의 테스트 오러클은 실행 시 입력되는 일련의 테스트 데이타에 대한 확장부위 행위의 실제값과 가변부위 행위 명세의 기대값 간에 매핑(mapping)이 존재하는지 여부에 따라 성공(pass) 또는 실패(fail)로 판단하는 도구이다.

2) 테스트케이스 생성 방법에 대한 연구

신뢰성 높은 프레임웍 기반의 소프트웨어 개발을 효과적으로 지원하기 위해서는 객체 지향 프레임웍의 가변부위에 대한 기능성을 효과적으로 (재)시험할 수 있는 테스트케이스 생성 방법에 대한 연구가 필요하다.

이를 위해서는 우선적으로 프레임웍의 가변부위에 대한 테스트 패턴 추출 방법에 대한 연구가 요구된다.

따라서 본 연구팀은 프레임웍의 가변부위에 대한 테스트 패턴을 구조적 테스트 패턴과 행위적 패턴으로 구분하고 이들을 추출하는 방법을 다음과 같은 방법으로 연구하였다.

① 프레임웍의 가변부위에 허용 가능한 합성 패턴을 분석하여 얻은 구조적 정형 테스트 모델로 부터 가변부위의 객체 구조 도메인을 유한 개의 서브 도메인들로 분할하여 구조적 테스트 패턴을 추출하는 방법을 연구하였다.

② 프레임웍의 가변부위에 허용 가능한 상호작용 패턴을 분석하여 얻은 행위적 정형 테스트 모델로부터 가변부위의 상호작용 도메인을 유한개의 서브 도메인으로 분할하여 행위적 테스트 패턴을 추출하는 방법을 연구하였다.

본 연구팀은 앞으로 프레임웍의 가변부위에 대한 구조적 / 행위적 테스트 패턴 추출 방법의 연구 결과로 추출된 프레임웍의 가변부위에 대한 구조적 / 행위적 테스트 패턴으로부터 테스트케이스들을 효율적으로 생

성하기 위한 방법을 연구할 계획이다.

8. 결 론

객체지향 프레임웍은 개조와 확장이 용이한 클래스들로 분해될 수 있는 유연한 아키텍처를 제공함으로써 효율적인 소프트웨어의 개발을 지원하지만 시험의 제어와 관찰이 어려운 성질을 갖고 있다. 시험성을 높이기 위하여 프레임웍에 시험 지원 코드를 임의로 추가할 경우 프레임웍 본래의 기능에 영향을 주게되어 시험의 신뢰성이 저하된다. 본 논문에서는 개조, 확장된 객체지향 프레임웍의 시험성을 프레임웍 본래의 기능에 영향을 주지 않으면서 높일 수 있도록, 테스트 지원 코드를 일단의 테스터 컴포넌트들로 설계하여 시험 대상 프레임웍에 BIT로 내장하는 방법을 설명하였다. 개조, 확장된 프레임웍의 가변 부위에 이와 같이 부착된 테스터 컴포넌트들은 프레임웍의 기능에 간섭하지 않으면서 시험의 제어와 관찰을 용이하게 하고, 필요에 따라 동적으로 제어(enable/disable)와 탈착(attach/detach)이 가능한 시험 지원 환경을 제공한다.

참 고 문 헌

[1] Fayad, M.E., Schmidt, D.C. and Johnson, R.E. Building Application Frameworks, John Wiley & Sons, Inc., 1999.

[2] Fayad, M.E., Schmidt, D.C. and Johnson, R.E. Implementing Application Frameworks, John Wiley & Sons, Inc., 1999.

[3] Fayad, M.E. and Johnson, R.E. Domain-Specific Application Frameworks, John Wiley & Sons, Inc., 2000.

[4] Gamma, E., et al., Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[5] Codenie, W, et al., From Custom Applications to Domain-Specific Frameworks, Comm. ACM, 40(10), Oct. 1997, pp. 71-77.

[6] Garlan, D., et al., Architectural Mismatch or Why its hard to build systems out of existing parts. Proc. 17th Int'l Conf. Software Engineering, Apr. 1995, pp. 179-185.

[7] Kung, D.C., Hsia, p. and Gao, J. (eds.). Testing Object-Oriented Software. IEEE CS Press, 1998.

[8] Binder, R.V. Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley, 2000.

[9] Doong, R. and Frankl, P. The ASTOOT Approach to Testing Object-Oriented Programs. ACM Transactions on Software Engineering and Methodology, 3(2), Apr. 1994, pp. 101-130.

[10] Hoffman, D. and Strooper, P. ClassBench: a Framework for Automated Class Testing. Software Maintenance: Practice and Experience, 27(5), May 1997, pp. 573-597.

[11] Harrold, M.J., Mcgregor, J.D. and Fitzpatrick, K.J. Incremental Testing of Object-Oriented Class Structures. Proceedings of 14th Int'l Conference on Software Engineering, May 1992, pp. 68-80.

[12] Jorgensen, P.C. and Erickson, C. Object-Oriented Integration Testing, Comm. ACM, 37(9), Sept. 1994, pp. 30-38.

[13] Sparks, S, et al., Managing Object-Oriented Framework Reuse, IEEE Computer, September 1996, 29(9), pp. 52-61.

[14] Fayad, M.E., Wang, Y. and King, G. Built-In Test Reuse. Building Application Frameworks, Fayad, M.E., et al., John Wiley & Sons, Inc., 1999, pp. 488-491.

[15] Binder, R.V. Design for Testability in Object-Oriented Systems. Comm. ACM, 37(9), Sep. 1994, pp. 87-101.

[16] Voas, J.M., Morell, L and Miller, K. Predicting Where Faults Can Hide from Testing. IEEE Software, Mar. 1991, pp. 41-48.

[17] Voas, J.M. and Miller, K.W. Software Testability: The New Verification. IEEE Software, 12(3), May 1995, pp. 17-28.

[18] Meyer, B. Applying Design by Contract. IEEE Computer, Oct. 1992, pp. 40-51.

[19] Rosenblum, D.S. A Practical Approach to Programming with Assertions. IEEE Transactions on Software Engineering, 21(1), Jan. 1995, pp. 19-31.

[20] Voas, J. and Kassab, L. Using Assertions to Make Untestable Software More Testable, Software Quality Professional Journal, 1(4), Sep. 1999

[21] Pree, W. Design Patterns for Object-Oriented Software Development. Addison-Wesley, 1995.

[22] Schmid, H.A. Systematic Framework Design by Generalization. Comm. ACM, 40(10), Oct. 1997, pp. 48-51.

[23] Helm, R, et al. Contracts: Specifying Behavioral Compositions in Object-Oriented Systems, Proc. OOPSLA'90, Ottawa, Canada, 1990.

[24] Steyaert, P, et al. Reuse Contracts: Managing the Evolution of Reusable Assets, Proc. OOPSLA'96, San Jose, CA, USA, Oct. 6-10, 1996.

신 동 익

1996년 2월 고려대학교 전산학과(학사). 1998년 8월 고려대학교 전산학과(석사). 1998년 9월 ~ 현재 고려대학교 전산학과 박사과정. 1998년 9월 ~ 현재 고려대학교 전산학과 시간강사. 관심분야는 소프트웨어 테스팅, 정형명세

전 태 웅

1981년 서울대학교 계산통계학과(학사). 1983년 서울대학교 계산통계학과(석사). 1992년 Illinois Institute of Technology (IIT), Chicago, Illinois(전산학 박사). 1983년 ~ 1987년 금성통신(현 LG전자) 연구소 주임연구원. 1992년 ~ 1995년 LG산전 연구소 책임연구원. 1995년 ~ 현재 고려대학교 전산학과 부교수. 관심분야는 소프트웨어 테스팅, 객체지향 개발 방법, 소프트웨어 아키텍쳐 등.

이 승 룡

1978년 2월 고려대학교 재료공학(학사). 1987년 12월 Illinois Institute of Technology(IIT), Chicago, Illinois 전산학 석사. 1991년 12월 IIT, Chicago, Illinois, 전산학 박사. 1991년 9월 ~1993년 8월, Governors State University, Illinois, Department of Computer Science 조교수. 1993년 9월 ~현재 경희대학교 전자정보학부 교수. 관심분야는 내장형 시스템, 실시간 시스템, 실시간 고장허용시스템, 멀티미디어 시스템