객체지향 프레임웍 후크 클래스의 시험성 강화를 위한 테스트 센서타이저 설계

정문호[0] 전태웅
LG전자기술원 MI그룹[0], 고려대학교 전산학과
mhjung@LG-Elite.com[0],jeon@selab.korea.ac.kr

The design of test sensitizer for high testability of hook classes in an object-oriented framework

Munho-Jung[0] Taewoong Jeon
MI gr., LG Electronics Institute of Technology[0], Dept. of Computer Science, Korea University

요 약

프레임웍의 결함들을 효과적으로 발견하기 위해서는 테스트 실행 과정 중에서 결함들이 민감하게 감응하여 결함으로 인한 오동작의 흔적이 남겨질 수 있어야 한다 그런데 프레임웍은 개조, 합성된 확장 부위에 결합되는 후크 클래스(hook class)들의 시험에 대한 제어와 관찰이 어려운 성질을 가지고 있다. 이를 해결하기 위해 프레임웍의 정상동작 여부를 판단하는데 단서가 되는 자료(clue data)를 포착하여 외부로 드러내는 기능을 수행하는 테스트 센서타이저를 설계하였다.

1. 서 론

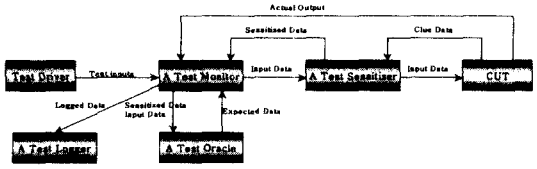
최근의 객체지향 기술은 여러 응용 분야에 재사용 가능한 프레임웍(framework)의 형태로 발전하고 있다[2,3,4]. 소프트웨어에서 신뢰성은 핵심적인 품질 요소이다. 객체지향 프레임웍은 다수의 응용 프로그램의 개발에 반복적으로 재사용되므로 신뢰성은 더욱 중요하다. 신뢰성 높은 객체지향 프레임웍 기반의 소프트웨어 개발을 효과적으로 지원하기 위해서는 객체지향 프레임웍을 구축, 유지보수 및 재사용하는 단계에서 효율적으로 (재)시험할 수 있어야 한다.

현재, 객체지향 소프트웨어에 대한 시험 방법들이 많이 소개되어 있다[7,8]. 그러나 객체지향 프레임웍은 기존의 이러한 시험 방법들만으로는 프레임웍의 변경부위에 대한 효과적인 시험을 어렵게 하는 여러 가지 고유한 문제들을 지니고 있다. 프레임웍의 확장부위에 대한 시험 실행 시 실행 시점의 예측이나 제어가 어렵고 시험 결과의 관찰이 어려워서 오동작이 발생 시점과 발생 지점에서 즉각적으로 발견되기 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 시험 지원 코드에 의해 간섭 받음으로 인하여 시험 결과의 신뢰성이 저하되는 문제가 발생하게 된다.

본 논문은 프레임웍 본래의 코드와 기능에 영향을 주지 않으면서 프레임웍의 가변부위인 후크 클래스의 시험에 대한 민감도와 관찰도를 높임으로써 프레임웍의 시험성을 강화하는 방법과 이를 지원하는 테스트 센서타이저(Test Sensitizer)의 설계를 기술한다. 본 논문에서 제안한 테스트 센서타이저는 시험 대상 후크 클래스 객체의 실행 사전, 사후 상태를 포착하여 시험의 성공 여부 판단에 단서가 되는 자료로 제공하며, 제어와 탈착이 가능한 BIT(Built-In Test) 컴포넌트의 형태로 설계되었다.

2. 전체 시험환경

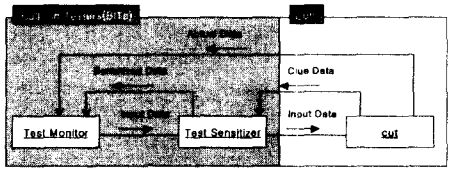
그림 1은 테스트 환경에서 CUT(Component under Test)와 테스트 센서타이저를 포함한 시험지원 컴포넌트들의 협동과정(collaboration)을 표현하고 있다.



[그림 1] 전체 시험환경의 Collaboration

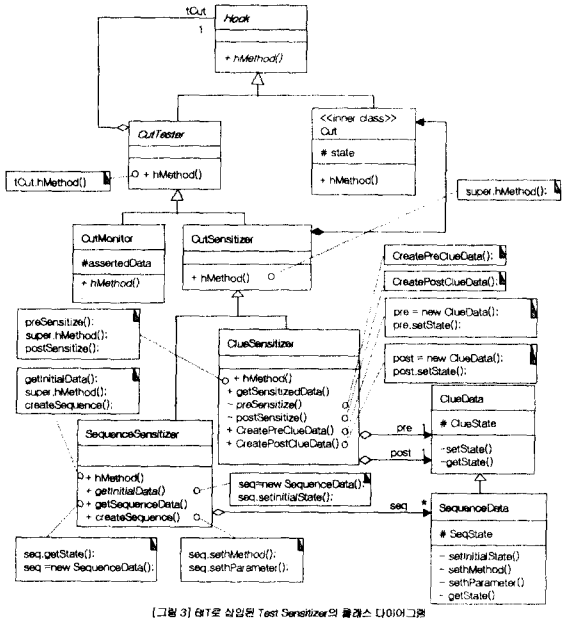
그림 1에서 CUT는 실행 도중 시험의 단서가 되는 실행 내용을 단서자료(ClueData)로 센서타이저에게 보낸다. 센서타이저는 이 과정에서 시험 실행 전후에 CUT를 참조하여 단서자료를 가져오고 이를 상태와 행위를 포함하는 객체로 만들어 머멘토 객체(Memento Object)[1]에 저장한다.

3. Test Sensitizer 설계



[그림 2] BIT Cut Sensitizer의 Object 다이어그램

본 장에서는 2장에서 설명한 시험 환경을 지원하는 테스트 센서타이저의 설계 구조와 동작 원리를 기술한다. 테스트 센서타이저는 그림 1의 시험 환경 하에서 실행되는 시험 대상 후크 클래스 (CUT) 객체의 실행 전후 시점에서 CUT의 상태를 객체화하여 단서자료를 만든다. 그리고 CUT로부터 얻어낸 단서자료를 감응자료(sensitized data)로 변환하여 테스트 모니터(test monitor)에게 전달한다. 테스트 모니터는 이를 다시 테스트 오라클(test oracle)에 전달하여 시험 예상 결과를 생성하는 데 영향을 준다 테스트 모니터는 테스트 오라클로부터 얻어낸 예상결과와 테스트 센서타이저의 도움을 받아 CUT로부터 얻어낸 실제결과를 비교하여 시험의 성공 여부를 판정한다. 테스트 모니터와 테스트 센서타이저는 그림 2와 같이 CUT에 BIT 컴포넌트로 내장된다.

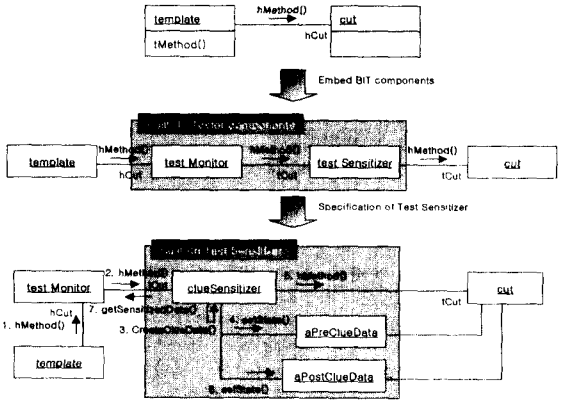


[그림 3] BIT로 삽입된 Test Sensitizer의 클래스 다이어그램

3-1. BIT Test Sensitizer 클래스 다이어그램

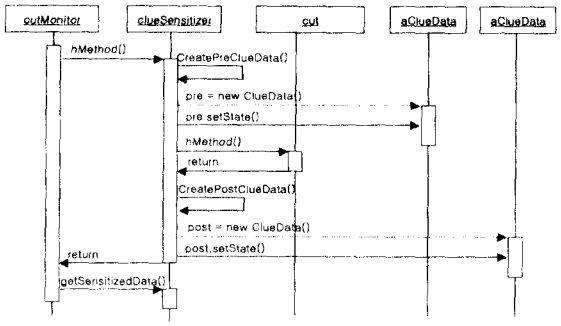
그림 3에서 CutSensitizer는 Cut를 내부 클래스로 포함하고 있어서 Cut의 내부에 private과 protected로 선언된 멤버변수와 함수들에 직접적인 접근이 가능하다. CutSensitizer는 ClueSensitizer와 SequenceSensitizer의 두개의 서브클래스를 갖는다. ClueSensitizer는 CutMonitor로부터 전달된 hMethod() 호출 전후의 상태를 ClueData에 저장한다. SequenceSensitizer는 hMethod()가 호출될 때마다 createSequence()를 호출하여 hMethod의 종류와 파라미터 등을 SequenceData에 저장한다. SequenceData에는 또한 시스템이 실행되는 실행 순서가 저장된다. 이 과정에서 ClueData에 저장된 사전상태 (prestate), 사후상태 (poststate)는 이후 테스팅 과정에서 필요할 때에 시스템 상태의 복구를 가능하게 한다.

3-2. Test Sensitizer 객체 구조



[그림 4] Clue Sensitizer의 객체구조

그림 4 ClueSensitizer의 객체구조는 template, testMonitor, clueSensitizer, 그리고 cut로 연결되었고 hMethod()의 실행에는 아무런 영향을 미치지 않음을 보여준다. 다만 테스트 컴포넌트들을 Template에서 cut로 연결되는 실행과정에 삽입하여 테스트를 수행하고 있다. Built-In Test Sensitizer의 내부는 clueSensitizer와 사전상태를 저장하는 aPreClueData와 사후상태를 저장하는 aPostClueData로 구성되어 있다. 시험 실행 시 test Monitor에서 hMethod()가 전달되면 clue Sensitizer가 cut의 전달 사전의 상태를 aPreClueData에 저장하고 hMethod()가 cut에서 실행되면 aPostClueData에 사후상태가 저장된다.



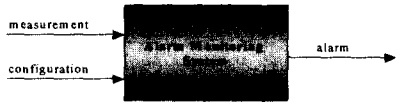
[그림 5] Clue Sensitizer의 시퀀스 다이어그램

3-3. Test Sensitizer 시퀀스 다이어그램

그림 5에서 *cutMonitor* 객체로부터 hMethod()가 호출되기 직전에 clue 센서타이저가 Cut의 이전상태를 포착하여 aClueData에 저장하고 Cut의 hMethod()에 영향을 미치지 않고 실행한다. hMethod()의 리턴 값을 받으면 Cut의 실행 후의 상태를 다시 Clue 센서타이저가 포착하여 aClueData에 저장한다. 이후 cut 모니터가 cut의 정상적인 동작 여부를 판단하기 위해서 ClueSensitizer에게 감응자료(Sensitized Data)를 요청하게 된다.

4. 적용 사례

프레임웍의 가변부위에 테스터 센서타이저를 부착, 내장하는 방법의 타당성과 효용성을 시험하기 위하여 3장에서 기술한 설계 방법에 따라 실제 프레임웍의 가변부위에 테스트 센서타이저를 구현, 내장하여 시험 지원 환경을 구축한 예를 설명한다.



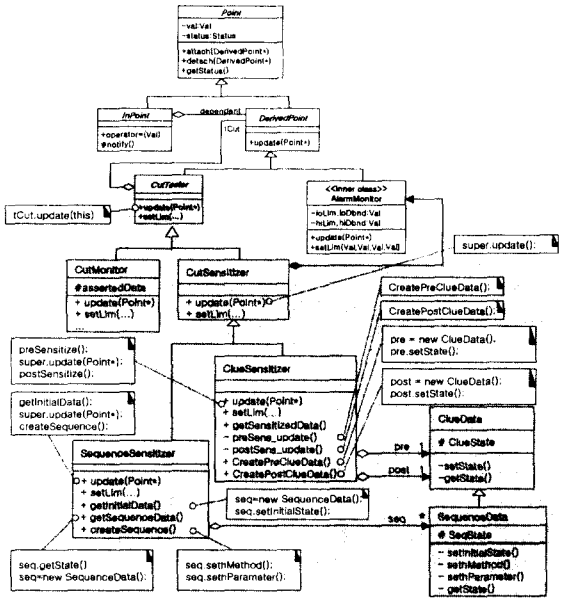
[그림 6] Alarm Monitoring System

Alarm Monitoring System(그림 6)은 제어 공정의 상태를 감시하는 시스템 응용 분야에 사용하기 위하여 본 연구팀에 의해 개발된 것이다. 이 시스템은 외부 환경의 상태 변화를 감시하여 비정상적인 상태로의 변화가 발생 시 알람을 발생한다.

4-1. Alarm Monitoring System 의 클래스 다이어그램

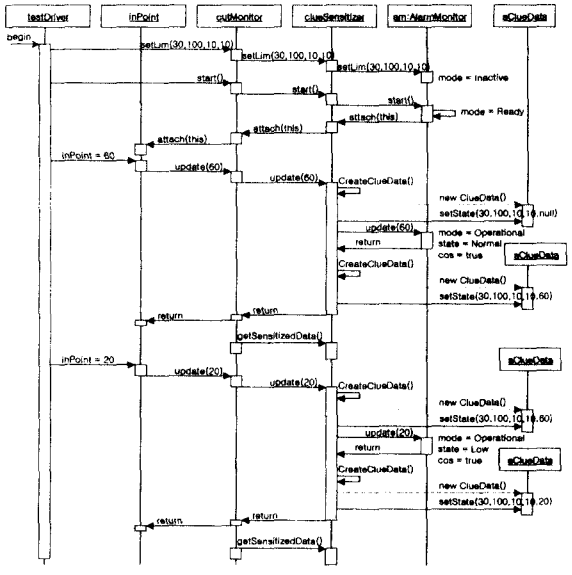
그림 7에서 Cut인 AlarmMonitor가 CutSensitizer의 내부 클래스로 삽입되어 이를 포함하는 CutSensitizer는 Cut인 AlarmMonitor의 내부에 private으로 선언된 loLim(low Limit), loDbnd(low Deadband), hiLim(high Limit), 그리고

hiDbnd(high Deadband)의 값을 직접적으로 접근할 수 있다.



[그림 7] BIT 컴포넌트가 내장된 Alarm Monitor의 Class Diagram

4-2. Alarm Monitoring System의 시퀀스 다이어그램



[그림 8] Clue Sensitizer가 내장된 Alarm Monitor의 Sequence Diagram

① testDriver에서 am:AlarmMonitor의 setLim()를 호출하면 입력된 파라미터 값인 loLim, hiLim, loDbnd, hiDbnd이 각각 30, 100, 10, 10값이 am:AlarmMonitor에 설정되고 am의 모드가 inactive로 설정된다. 이 과정 중 setLim()는 BIT 컴포넌트인 cutMonitor 와 clueSensitizer를 거쳐 전달된다.

② testDriver에서 시험을 시작하기 위해 start()를 호출하면 이는 cutMonitor와 clueSensitizer를 거쳐 am에 전달된다. 이때 am의 모드가 Ready 상태로 변경되고 inPoint에 am의 자신의 attach() 메소드를 이용해 부착된다.

③ testDriver에서 inPoint값으로 60이 전달되면 inPoint 객체는 update() 메소드를 통해 am에 전달되게 된다. 이 과정에서 clueSensitizer가 update()의 파라미터 값을 포착하여 이전 과정에서 저장된 loLim, hiLim, loDbnd, hiDbnd 값들과 함께 aClueData객체에 저장하게 된다.

④ 60을 파라미터 값으로 갖는 update() 메소드가 실행되면서 am의 모드가 Operational로 변경되고 상태가 Normal로 설정된다. cos(Change of State)가 true값으로 설정된다.

⑤ update() 메소드가 실행된 후 다시 ClueSensitizer는 aClueData의 객체를 하나 만들어서 사후 자료인 loLim, hiLim, loDbnd, hiDbnd, inPoint값인 30,100,10,10,60을 aClueData에 저장하게 된다.

⑥ testDriver에서 inPoint값으로 20이 전달되면 ③, ④, ⑤ 과정이 반복되게 된다.

⑦ CutMonitor에서 clueSensitizer의 getSensitizedData() 메소드를 호출하면 ClueSensitizer는 머멘토 객체(Memento Object)로 저장되어 있는 단서자료(ClueData)를 cutMonitor에게 전달하게 된다.

5. 결론

본 논문의 연구성과는 프레임웍의 가변부위에 대한 시험 시 삽입되는 시험지원 도구인 테스트 센서타이저가 프레임웍 코드의 변경이나 실행에 간섭 없이 시험의 민감도(sensitivity와 관찰가능도(Observability)를 효과적으로 높이도록 설계한 것이다. 민감도는 소프트웨어에 내재한 결함들이 테스트 실행에 감응하여 오동작의 흔적을 남기는 능력을 의미한다. 관찰가능도는 테스트 실행 결과를 외부에서 관찰할 수 있는 능력을 의미한다.

테스트 센서타이저의 설계 과정에서는 시험대상(CUT)을 센서타이저의 내부클래스(inner class)로 설정하여 삽입하여 시험대상 내부의 은닉되어 있는 정보-private과 protected로 선언된 멤버함수와 변수를 외면화(externalization)하였다. 또한 세부 설계를 위해 객체지향 설계 패턴인 Mmemento와 Iterator 패턴[1]을 사용하였다. 하지만 시험대상을 테스트 센서타이저의 내부클래스로 정의할 경우 decorator로 전달되는 hMethod()를 제외한 CUT의 외부연결이 단절되는 결과를 초래할 수 있다. 향후 시험 대상을 내부클래스로 정의하지 않고 메타 정보를 이용해 설계할 경우 이와 같은 단점을 극복할 수 있을 것이다.

6. 참고 문헌

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

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

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

[4] Fayad, M.E. and Johnson, R.E. Domain-Specific Application Frameworks, John Wiley & Sons, Inc., 2000.
Kung, D.C., Hsia, p. and Gao, J. (eds.). Testing Object-Oriented Software. IEEE CS Press, 1998.

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