# 객체지향 소프트웨어를 위한 주요 블랙박스 테스트 기법들의 비교

## (Comparison of Major Black-Box Testing Methods in Object-Oriented Software)

서 광 익 † 최 은 만 ††
(Kwang Ik Seo) (Eun Man Choi)

**요 약** 소프트웨어의 규모가 커지면서 시스템 기능 테스트 단계에 일일이 원시코드를 참조하면서 테스트하는 것은 현실적으로 불가능하다. 따라서 시스템 테스트 단계에서 다양한 요구명세 정보들을 이용하는 블랙박스 테스트 방법들이 많이 연구되고 있다. 테스트 기법에 활용한 요구명세는 시스템을 표현하는 기술 방법과 의미의 차이로 인해 테스트 케이스와 커버리지가 크게 달라서 테스트 계획 단계에서 테스트 기법과 그 기법의 적용 결과에 대해 깊이 고려해야 한다. 이러한 필요성으로 인해 본 연구에서는 다양한 블랙박스 테스트 기법들에 대한 실험과 그 결과에 대해 논하였다. 기법의 특성이 미치는 테스트 결과를 분석하고 평가하기 위해 객체지향 시스템의 명세와 모델링 기법으로 가장 잘 알려진 UML을 이용한 테스트 기법을 포함하여 다섯 가지의 블랙박스 테스트 기법을 실행하고 비교하였다. 그리고 다섯 가지 기법들을 서로 다른 두 응용 시스템에 적용하였고 그 결과로 추출된 테스트 케이스의 커버리지를 분석하였다.

**키워드** : 요구명세기반 테스트, 블랙박스 테스트, UML, 사용사례, 협업도, OCL, Object-Z

***Abstract*** As the size of software is getting huge, it has been impossible for testers to test all of source code. Therefore, the method which uses information of requirements for testing has been researched and accepted. Although there are a lot of test methods based on requirement specification, each method has a different approach and coverage. So we should consider their various approach and coverage in the stage of test planning. This paper presents the empirical comparison and the result by applying various black-box testing methods to software system. For this experiment we studied five black-box testing methods including UML based specification technique. We applied the five testing methods to application system and evaluated the coverage of extracted test case.

**Key words** : Specification-based Test, Black-box Test, UML, Use-Case, Collaboration Diagram, OCL, Object-Z

## 1. 서 론

테스트 기법에는 크게 두 가지 접근 방법이 있다. 하나는 테스트 작업이 원시코드로부터 시작되는 화이트 박스 테스트와 프로그램의 내부 구조는 고려하지 않고 프로그램에 대한 입출력으로 얻어지는 결과를 검사하는 블랙박스 테스트 방법이다. 화이트 박스 테스트 방법은 이론적이며 분석적이다. 그리고 프로그램의 소스 코드를 이해하면서 동시에 프로세스의 흐름을 파악해야 하므로 단위 테스트 이상의 규모에서 적용하기에는 한계가 있다. 따라서 실제 소프트웨어 제품을 테스트하는 대부분의 실무에서는 블랙박스 방법을 사용하고 있다.

하지만 동일한 접근 방법인 블랙박스 기법이라 하더라도 요구명세의 표현 기법과 테스트 데이타를 추출하는 방법이 다르면 테스트의 결과는 달라진다. 예를 들어 잘 알려진 UML의 사용사례를 이용한 테스트 방법[1]은 시스템 경계를 기점으로 전체를 블랙박스로 테스트한다. 반면 OCL(Object Constraint Language)을 테스트에 적용하는 방법[2]은 클래스나 멤버함수의 여러 가지 제약사항이나 불변성(invariant)을 근거로 테스트한다.

이와 같이 테스트를 수행하기 위해서는 타겟 시스템의 특성과 목적은 물론이고 테스트의 접근 방법까지 함께 고려해야 한다. 만약 테스트 목적에 맞도록 적절한

† 학생회원 : 동국대학교 컴퓨터공학과 박사과정
bradseo@dongguk.edu
†† 종신회원 : 동국대학교 컴퓨터 멀티미디어공학과 정교수
emchoi@dgu.ac.kr
논문접수 : 2004년 12월 8일
심사완료 : 2005년 11월 7일

테스트 기법을 적용하지 않는다면 테스트 결과를 신뢰할 수 없을 뿐더러 수많은 시간과 인력, 자원의 낭비로 인해 막대한 손실을 초래하게 될 것이다. 따라서 현장에서 테스트 기법들이 테스트 업무에 적절하게 적용되도록 다양한 테스트 기법들의 비교 연구가 있어야 한다.

현재 테스트 기법의 비교에 관한 연구들은 성능 및 효율성을 비교하기 위해 고려해야 하는 요소를 정의하고 테스트 케이스를 구하는 연구 논문들이 있다. 이러한 논문들은 비교 실험을 수행 할 때 비용이나 효율성, 편의성, 결함 등을 기준으로 비교할 수 있는 비교 방법을 이론적으로 제안했다[3-5]. 또한 지금까지 수행된 테스트 기법의 비교 논문에서 실제 적용된 시스템들은 대부분 구조적인 방법이거나 화이트박스 테스트 기법을 중심으로 한 테스트 기법들의 비교와 문장 커버리지나 분기문 커버리지 그리고 데이타 흐름에 대한 테스트 기법 비교들이 대부분이다[6-8]. 프로그래밍 개발 관점의 변화 이후, 객체지향 개발 방법과 UML의 모델링 기법이 널리 사용되면서 중요시 되고 있다. 그럼에도 불구하고 객체지향 기반의 시스템을 분석, 설계하는 UML 명세서를 통해 객체지향 언어로 구현한 시스템을 테스트하는 기법들에 대한 비교 연구는 미약한 실정이다. 또한 UML 기반으로 수행되는 테스트 기법들 중 본 논문에서 비교 대상으로 선정한 5가지의 테스트 기법들은 각각의 테스트 방법만 소개하면서 해당 방법에 대한 시스템의 특성을 고려하여 효율적인 테스트를 할 수 있는 방법만 소개했다.

이러한 이유로 인해 본 논문에서는 객체지향 언어로 구현된 동일한 시스템에 대해 서로 다른 요구명세로부터 테스트 케이스를 추출하는 기법들을 적용하고 그 결과의 차이점을 분석하였다. 차이점을 비교하는 기준으로는 위에서 언급한 문장 커버리지나 분기문 커버리지 그리고 데이타 흐름에 대한 커버리지가 아니라 객체가 포함하고 있는 변수와 메소드를 시험하기 위한 접근 여부로 커버리지의 비교 기준을 적용하였다. 왜냐하면 문장 커버리지나 분기분 커버리지 등을 시험하기 위해서는 테스트케이스가 문장 중심이거나 분기문 중심으로 작성되어야 하지만 본 논문에서는 비교하고자 하는 테스트 기법들의 테스트케이스 기술 형식이나 목적이 각기 다름으로 인해 비교 기준의 일치성 확보가 어렵기 때문이다. 따라서 비교 기준을 동일하게 적용하기 위해 메소드와 변수들의 시험 여부를 비교하는 방법을 채택하였다. 테스트 기법 적용 실험으로는 UML의 사용사례와 협업도 그리고 Object-Z를 이용한 방법과 OCL, 마지막으로 사용사례를 확장하고 응용한 방법을 이용하여 각각을 ATM 시스템과 수강신청 시스템 테스트에 적용하였다. 또한 다섯 가지 테스트 기법의 커버리지를 파악하고 서로 상충되는 시스템 테스트 영역에 대해 가장 효과적으로 접근할 수 있는 테스트 기법의 조합을 제안하였다.

본 논문의 구성은 2장에서는 관련 연구로써 실험에 적용하려는 요구명세 기반의 테스트 기법과 절차를 간단히 소개하였고 3장에서는 실제로 각 기법을 ATM 시스템과 수강신청 시스템을 테스트하면서 테스트 케이스를 추출하는 실험 과정과 결과를 정리하였다. 4장에서는 테스트 케이스의 추출 과정에 대한 비교와 분석, 그리고 5장에서는 테스트 케이스가 미치는 시스템의 영역 즉, 커버리지를 분석하고 6장에서는 커버리지의 최대화를 위한 효과적인 테스트 기법의 조합을 제안했다. 마지막으로 7장에서는 결론에 대해 서술하였다.

## 2. 요구명세를 기반으로 하는 테스트 기법

과학의 발전과 함께 소프트웨어의 복잡도가 급격히 높아지고 있다. 따라서 이러한 시스템의 이해와 의사소통을 돕기 위해 다양한 요구명세 기법 및 모델링 기법이 연구되고 있다. 요구명세를 이용한 테스트 기법들로는 UML에서 사용하는 사용사례(Use Case) 이용 기법[1], 협업도 이용 기법[9], Object-Z 이용 기법[10], OCL 이용 기법[2], 사용사례를 확장하고 응용한 기법[11], 상태다이어그램(StateChart Diagram) 이용 기법[12], 페트리 넷(Petri Net) 이용 기법[13] 등이 있다.

전통적 개발 방법에서 객체지향 개발 방법으로 소프트웨어 개발의 패러다임이 변화되었고 이로 인해 현업에서는 대부분 시스템 개발 구현의 언어로 객체지향 언어를 사용한다. 본 연구에서도 객체지향 방법과 밀접한 관계가 있는 테스트 기법들을 선택하여 실험하였다. 또한 실험 대상 시스템도 객체지향 언어인 자바(Java)로 구현된 시스템을 선택하였다. 객체지향 관점에서 시스템을 모델링하는 UML과 관련 있는 테스트 기법으로 사용사례와 협력도, 확장된 사용사례, 상태도가 적용된 Object-Z 테스트 기법과 UML에서의 제약사항을 위해 고안된 OCL을 이용한 기법을 비교하였다. 객체지향 기법을 적용한 시스템 분석 및 설계에서는 UML을 사용하는 것이 보편화 되었다. 따라서 개발자가 시스템을 구현하기 위해 UML로 시스템의 요구사항과 설계서를 명세하고 구현한 후 구현된 시스템을 테스트하기 위해서는 UML로 정의된 시스템 요구 분석서와 설계 명세서를 참고 할 것이다. 따라서 자주 사용되는 UML을 이용한 테스트 케이스 추출 기법은 객체지향 관점에서의 개발에 있어서 중요한 부분이 된다. 본 논문에서 비교 실험하고자하는 테스트 기법은 모두 이러한 UML 명세 언어에서 정의하고 있고, 실제로 개발자가 시스템을 개발할 때 자주 사용하는 명세 기법이다. 따라서 UML에서 정의하고 있는 여러 표현기법 중에서 사용사례,

OCL, 협업도, 상태도를 통해 객체지향 언어 명세를 위해 수정된 Object-Z를 이용한 테스트 케이스 추출 기법은 UML이 자주 사용되므로 인해 주요한 테스트 기법이라 할 수 있다. 반면 현재 실험하려는 두 대상 시스템이 동기화 검증이 필요한 병렬 또는 리얼타임 시스템이 아니라 정해진 작업 순서에 의해 변환되는 시스템이다. 즉, 상태 변화 시뮬레이션과 사용자 인터페이스를 통해 데이타를 조작하기 위한 순차적인 트랙잭션이 주요 기능인 ATM 시스템과 수강신청 시스템이다. 따라서 페트리 넷과 같이 병렬처리와 동기화 명세를 위해 고안된 기법은 생략하기로 한다.

### 2.1 사용사례 기반 테스트 케이스 추출 기법

사용사례는 업무 관련자들 사이의 의사소통을 명확하게 할 뿐만 아니라 시스템이 제공해야 하는 기능들을 발견하고 명세하기 위해 사용된다. 그리고 정의한 기능들을 시스템이 올바르게 제공하고 있는지 검증할 수 있는 시나리오를 작성하는데 기초적인 자료가 된다. 이러한 사용사례를 이용한 테스트 케이스의 추출 방법[1]은 그림 1과 같다.

사용사례를 이용하여 시스템의 기능을 명세하고 이를 바탕으로 사건 흐름을 정리한다. 자연어로 정리된 사건 흐름을 그래프로 재정의하면서 발생 가능한 일련의 시나리오를 작성한다. 마지막으로 테스트하고자 하는 시나리오에 대해 예외사항들을 추가하여 테스트 케이스를 추출한다. 이러한 방법은 시스템 내부의 알고리즘이나 프로그램 모듈간의 상호작용 등은 고려되지 않는다. 다만 사용자가 실제 시스템을 사용하면서 발생할 수 있는 상황들을 미리 파악하고 정의하여 검증하는 과정이라 할 수 있다.

### 2.2 협력도를 이용한 테스트 케이스 추출 기법

UML의 협력도(Collaboration Diagram)는 객체들 간의 상호작용 행위를 나타내기 위해 객체간의 연관 관계와 그들 사이에서 주고받는 메시지들을 같이 표현한다. 따라서 특정 기능에 대해 작동하는 객체들과 메시지들의 흐름을 파악할 수 있다. 협력도를 이용한 테스트 케이스 추출 기법[9]은 이러한 특징을 이용하여 기능 중심으로 협력도를 작성하고 객체들의 관계와 메시지들의 흐름을 대상으로 테스트 케이스를 추출한다. 다시 말하면 특정 기능 중심으로 오퍼레이션의 호출 순서에 따라 입출력 데이타를 정의하고 테스트 케이스를 추출한다. 그림 2는 협력도를 이용한 테스트 케이스 추출 기법을 나타내고 있다. 그림 2는 협력도를 통해 메소드의 흐름을 파악하고, 파악된 메소드의 진행 상태에 따라 테스트 케이스를 추출하고 있다. 협력도를 이용하여 테스트 케이스를 추출하는 방법은 관련 메소드의 상호 작용 순서를 통해 해당 객체의 신뢰성과 테스트의 완전성을 파악할 수 있다. 그리고 오류가 많이 발생하는 지점을 찾을 수 가 있기 때문에 오류와 관련이 있는 객체들을 검사하고 관리할 수 있다는 장점이 있다.

### 2.3 Object-Z를 이용한 케이스 추출 기법

Object-Z[14]는 객체지향 언어 정형 명세에 적합하도록 기존의 Z를 응용하였다. 그리고 Chen은 Object-Z를 이용하여 객체지향 소프트웨어를 테스트하는 방법을 제안했다[10]. Object-Z를 이용하여 테스트 케이스를 추출하는 방법은 그림 3과 같이 주요 클래스를 Object-Z로 명세한 후 메소드를 중심으로 상태 다이어그램을 구현한다. Object-Z는 객체의 정적인 상태만을 나타낸다. 따라서 상태도를 이용하여 이벤트에 의한 객체의 상태 변화를 파악하는 것이 중요하므로 상태도를 이용하여 객체의 변화를 파악하고 테스트한다.

그리고 상태도에서 추출 가능 경로를 구하고 이 경로를 토대로 시나리오를 작성하여 입, 출력 값을 정의함으로써 테스트 케이스를 추출한다. Object-Z는 상태기반 명세가 가능하고 구현된 소프트웨어가 가져야 할 기능

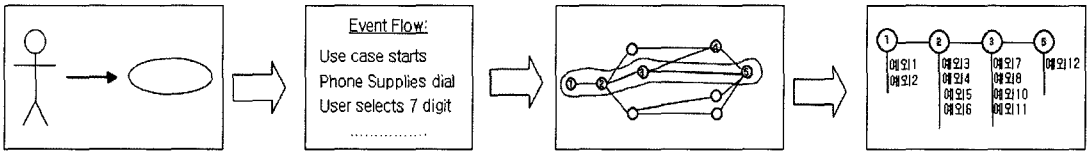



그림 1 단순 사용사례를 이용한 테스트 케이스 추출 절차

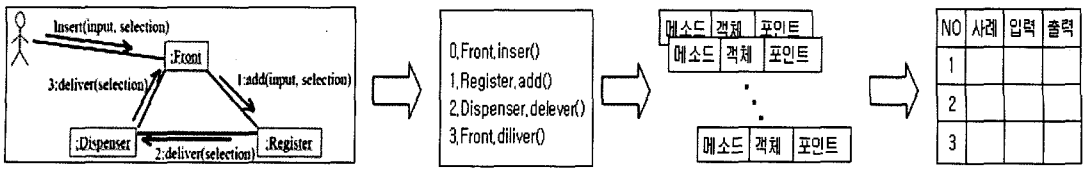



그림 2 협력도를 이용한 테스트 케이스추출 절차

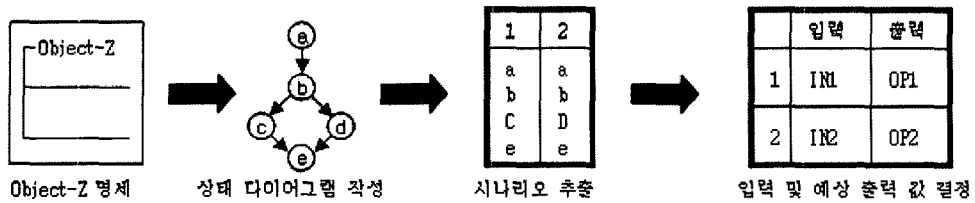



그림 3 Object-Z 기반의 테스트 케이스 추출 절차

적인 정보를 잘 나타낼 수가 있기 때문에 블랙 박스의 기능 테스트의 예측 결과를 찾아내기가 쉽다.

### 2.4 OCL을 이용한 케이스 추출 기법

UML로 표현한 설계 모델은 그래픽을 기반으로 한다. 그래픽 위주의 모델링은 사용자와 설계자, 프로그래머 등 프로젝트 관련자들 사이에 의사소통과 시스템에 대한 개념의 이해를 돕는데 유용하다. 하지만 UML의 사용사례와 같은 경우 사용사례에 해당하는 구체적인 내용과 제한적 요소는 그래픽적 요소만으로는 판단하기가 어렵다. 이를 보완하기 위해 OCL(Object Constraint Language)이 제안되었다. OCL은 선조건과 후조건을 사용하여 오퍼레이션과 메소드를 표현하기에 효과적인 방법이다. 또한 명확하고 애매모호하지 않은 객체의 의무와 권한을 나타낸다. 선조건과 후조건에 관한 표현 방식은 오퍼레이션 이름과 제한자 컨텍스트(Constraint Context)를 사용하여 표현한다. 이러한 선조건과 후조건 그리고 제한적 요소를 표현하는 의미적 내용을 통해 테스트 케이스를 추출한다.

테스트 케이스를 추출하는 단계는 그림 4와 같다. 먼저 테스트 될 기능의 도메인을 분할하고 OCL로 표현한다. 그리고 도메인 안에 있는 클래스들 간의 관계를 파악하고 이들의 속성, 초기값 등을 분할한다. 이렇게 분할된 각 요소들을 테이블로 표현한 후 진위값을 부여하여 테스트 케이스를 추출한다.

### 2.5 메시지 패스를 이용한 확장 사용사례 기반 테스트 케이스 추출 기법

확장된 사용사례 기반 테스트 기법[11](이하 '확장된 사용사례'라 함)에서 제안하고 테스트 케이스 추출 기법은 그림 5와 같다. 확장된 사용사례 기반 기법은 먼저 사용사례를 생성한 후 시나리오를 작성하여 구체적인 시나리오 인스턴스를 대입한다. 따라서 시나리오가 진행되는 동안 소스 코드 내에서 호출되는 변와 멤버 함수들을 구한다. 그리고 구해진 변수와 멤버 함수를 이용하여 액터의 입력 이벤트와, 예측된 출력 이벤트를 기초로 테스트 케이스를 추출한다.

또한 기능과 연관 있는 멤버 클래스와 매개변수를 추출하고 사용사례로부터 시나리오를 추출한다. 마지막으로 추출된 클래스와 매개변수를 시나리오와 매핑하여 이들로부터 구할 수 있는 경로 즉, MM-Path(Method/Message Path)를 구한 다음 경로(Path)를 이용하여 이

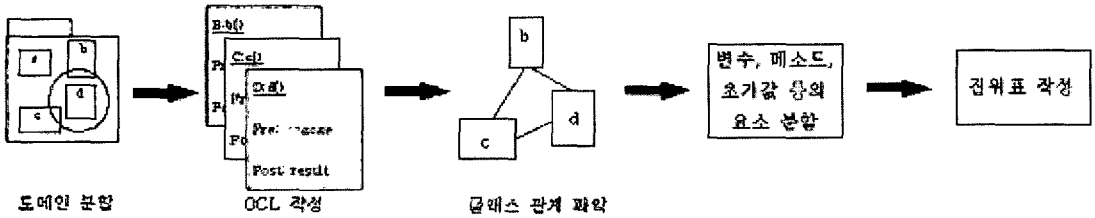



그림 4 OCL을 이용한 테스트 케이스 추출 절차

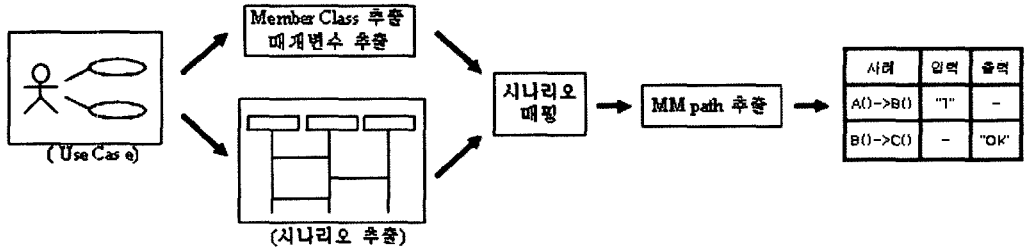



그림 5 확장된 사용사례를 이용한 테스트 케이스 추출 절차

벤트 입출력 자료를 중심으로 테스트 케이스를 추출하게 된다. 여기서 MM-Path는 메시지에 의해 연결된 연속된 메소드의 실행을 말한다.

확장사용사례를 이용하여 테스트를 실행할 때에는 시스템의 기본 기능들의 수행여부를 시험하기 위해 각 객체들 사이의 메시지 교환이 제대로 이루어지는지 테스트하는 작업이 필요하다.

## 3. 테스트 기법의 적용 실험

본 연구를 위해 사용자와 시스템간의 정보의 입력과 선택의 전달 과정이 빈번한 ATM(Automated Teller Machine)을 실험 적용하였다. 고객은 두 개의 예금 계좌를 가지고 있고 고객번호와 개인 비밀번호(PIN)를 가지고 있다고 하자. 고객은 모두 현금 인출과 같은 은행 업무를 위해서는 고객 번호와 비밀번호가 필요하다.

그림 6과 같이 시스템의 전체적인 내부 구조를 보면 KeyPay는JPannel로부터 상속을 받아 입력 화면을 구성한다. ATM은 Jframe으로부터 상속을 받아서 프레임을 구성하고 KeyPad, Bank, Customer, BankAccount를 이용한다. Bank는 Customer를 이용한다. 연관관계는 Bank를 이용해서 Customer를 접근할 수 있고, Customer를 이용해서 고객의 계좌를 접근할 수 있다.

### 3.1 사용사례 기반 테스트 케이스 추출 실험

외부 사용자의 관점에서 시스템의 기능을 파악하기 위해 작성한 사용사례에서 테스트하려는 기능을 추출한다.

그림 7은 시스템의 기능들 중 인출을 테스트하기 위한 사용사례이다. 이 사용사례에서 예금 인출 기능을 사용할 때 인출하려는 금액이 충분하거나 잔액보다 적을 수도 있다. 이러한 기능에 대한 정상적인 경우와 예외사항을 고려하면서 테스트 케이스를 만든다. 인출 사용 사례에 대한 테스트 케이스는 표 1과 같다.

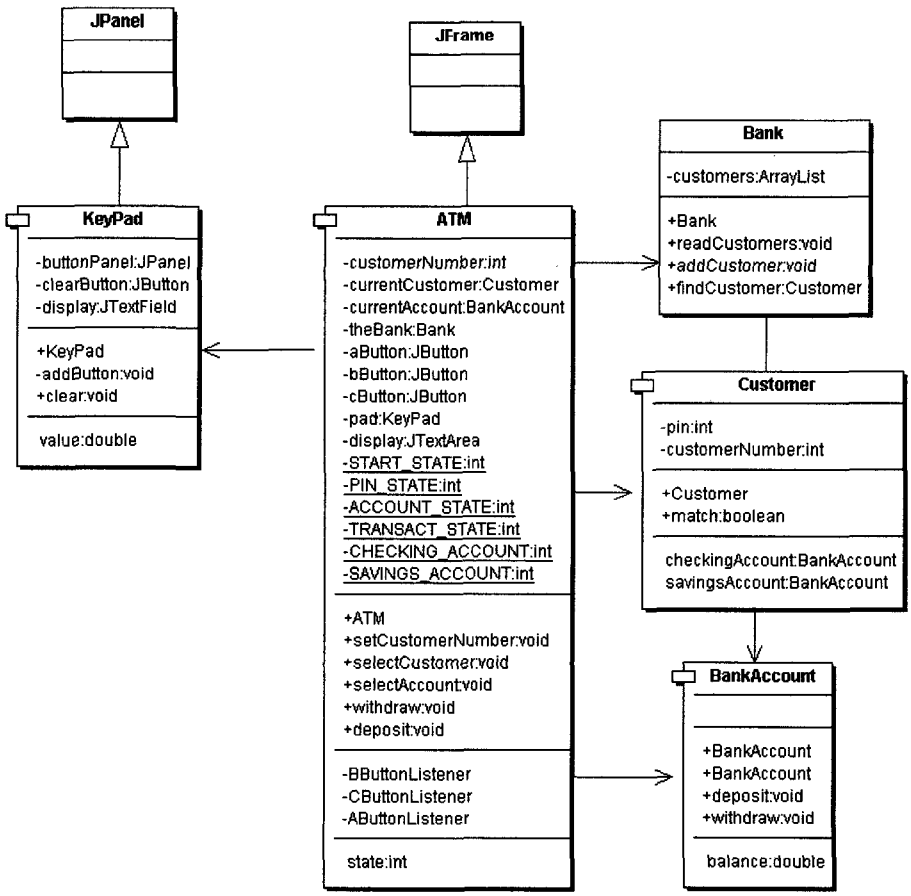
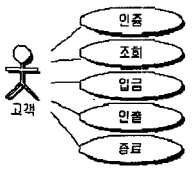



그림 6 ATM 클래스 다이어그램




| 사용사례이름 | 인출 |
|---|---|
| 참여 액터 | 고객 |
| 시작조건 | 정상적으로 고객이 인증을 완료한다. |
| 정상적 사건 흐름 | 1. 고객이 예금 인출 기능 버튼을 클릭한다.<br>2. 원하는 인출 금액을 입력한다.<br>3. 현금이 인출되고 잔액을 화면에 보여준다. |
| 예외사항 | 3.1 잔액이 부족하여 인출할 수 없다는 오류 메시지를 화면에 보여준다. |
| 종료조건 | 정상적으로 고객이 작업을 완료한다. |

그림 7 사용사례 기반 테스트 케이스

표 1 예금 인출 테스트 케이스

| 상태 | 입력 | 예상 결과 | 비고(잔액상태) |
|---|---|---|---|
| 정상 | 30만원 | 1. 현금 30만원<br>2. 잔액 메시지 20만원 | 작업 전: 50만원<br>작업 후: 20만원 |
| 비정상 | 80만원 | 1. 잔액 부족 메시지 | 작업 전: 50만원<br>작업 후: 50만원 |

### 3.2 협력도를 이용한 테스트 케이스 추출 기법

테스트 대상 시스템에 대한 협력도를 작성한다. ATM 시스템에 대한 협력도는 그림 8과 같다. 고객이 ATM 객체 생성을 시작한 후 시스템이 작업을 준비하고 고객에게 제공할 기능에 따라 협력이 필요한 객체들과 메시지를 주고받는 순서들이 표현되어 있다. 이러한 협력도를 바탕으로 테스트하기 위한 기능에 대해 메소드를 파악하고 자료의 예상 입출력을 파악하여 테스트 케이스를 추출한다. 테스트 케이스의 최종 자료는 표 2와 같다.

### 3.3 Object-Z 기반 테스트 케이스 추출 실험

테스트 할 대상에 대한 요구명세서 또는 코드 명세서를 이용하여 Object-Z로 정의한 후 그림 9와 같이 시스템의 상태 변화를 표현하기 위해 상태 변이 다이어그램(State Transition Diagram)을 작성한다. 그리고 그래프의 순서를 이용하여 시나리오를 만들고 각 시나리오를 입력 값과 예상 출력 값을 기준으로 정의한다. 이것이 표 3의 최종 테스트 케이스가 된다.

### 3.4 OCL 기반 테스트 케이스 추출 실험

먼저 테스트할 대상 객체들에 대하여 카테고리를 분할한다. 테스트 대상 시스템에 따라 다중도의 범위가 다양할 수 있지만 ATM의 경우에는 한번에 한명의 고객과 트랜젝션만 가능하므로 하위 경계LB(Low Boundary)가 1이다. 표 4는 이러한 분할 관계 내용을 나타내고 있다.

다음으로 관련된 클래스를 모두 OCL로 명세하고 ATM 시스템의 상태를 나타내는 state 변수를 각 클래스의 작용에 맞도록 분할한다. 그리고 각 분할된 조건이 True 또는 False가 되도록 결과 값을 정의한다. 표 5는 OCL 기반으로 추출된 최종 테스트 케이스이다.

### 3.5 메시지 패스를 이용한 확장 사용사례 기반 테스트 케이스 추출 실험

첫 번째 단계는 사용 사례와 시나리오의 준비 단계로써 고객이 ATM을 통해 고객 번호를 입력하고 비밀번호를 입력한 후 당좌예금(Check Account) 계좌를 선택하는 시나리오를 예로 구성한다. 그리고 예금을 인출하고 거래를 종료하면 처음 상태로 돌아가 새로운 고객의 번호를 입력 받기 위한 초기 상태로 전환하는 사용 사례를 표 6과 같이 작성한다. 또한 시나리오를 중심으로

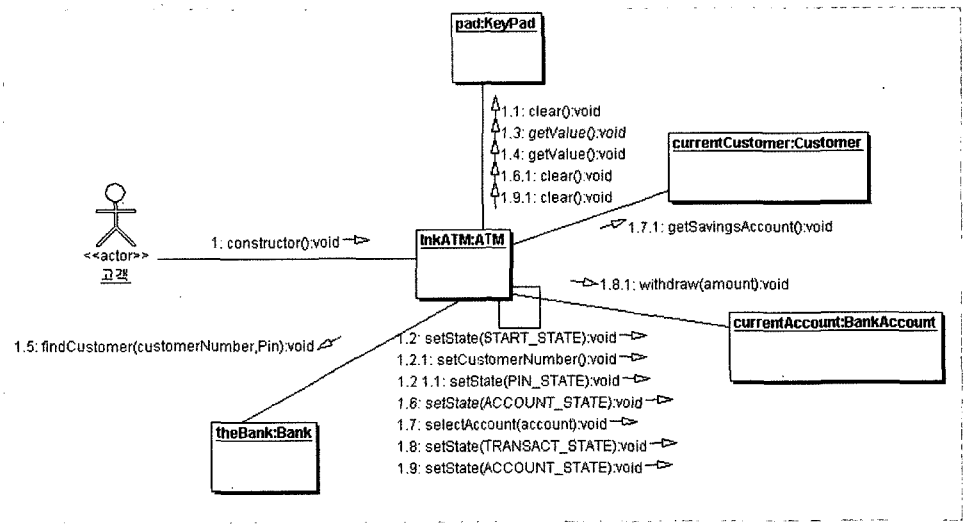



그림 8 협력도

표 2 협력도 기반 테스트 케이스

| NO | 테스트 케이스 | 코드삽입 | 예상 결과 |
|---|---|---|---|
| 1 | ATM.ATM | Instrumented code | actionName:constructor<br>linkEndOgjectTypeName:ATM<br>next : KeyPad.clear |
| 2 | Pad.clear | Instrumented code | actionName:clear()<br>linkEndOgjectTypeName: KeyPad<br>next : ATM.setState(START_STATE |
| 3 | ATM.setState(STAR T_STAT) | Instrumented code | actionName:setState(START_STATE)<br>linkEndOgjectTypeName: ATM<br>next : ... |
| ... | ... | ... | ... |
| 16 | currentAccount.withd raw | Instrumented code | actionName: withdraw()<br>linkEndOgjectTypeName: BankAccount<br>next : ATM.setState(ACCOUNT_STAT |
| 17 | setState(ACCOUNT_ STAT | Instrumented code | actionName: setState(ACCOUNT_STATE)<br>linkEndOgjectTypeName: ATM<br>next : KeyPad.clear |
| 18 | Pad.clear | Instrumented code | actionName: clear()<br>linkEndOgjectTypeName: KeyPad<br>next : NU |

표 3 Object-Z 기반 테스트 케이스

| | 입력 | | | 예상 결과 | 비고 |
|---|---|---|---|---|---|
| | 고객번호 | 비밀번호 | 인출 또는 입금 금액 | 잔액 | |
| 시나리오 | 1. 정상적인 인출 시나리오 | | | | 잔액 0 |
| | 1 | 1234 | 인출:300 | -300 | |
| | 2. 정상적인 입금 시나리오 | | | | 잔액 0 |
| | 1 | 1234 | 입금:300 | 300 | |
| | 3. 비밀번호 오류 시나리오 | | | | |
| | 1 | 1234 | | 오류 메세지 | |
| | 4. select Account 취소 시나리오 | | | | |
| | 1 | 1234 | | 초기 화면 | |
| | 5. 인출 후 입금 시나리오 | | | | 잔액 0 |
| | 1 | 1234 | 인출:300<br>입금:100 | -200 | 200 감소 |
| | 6. 입금 후 인출 시나리오 | | | | 잔액 0 |
| | 1 | 1234 | 입금:300<br>인출:100 | 200 | 200 증가 |

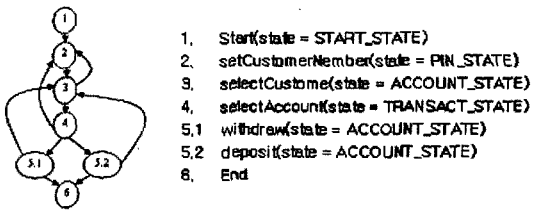



그림 9 ATM 상태 변이 다이어그램

변수와 메소드를 추출하고 사용 사례의 인스턴스를 매핑한다.

시나리오로부터 메소드에서 메시지가 전달되는 경로를 구하고 이를 바탕으로 매핑된 인스턴스의 입력값과 출력값을 정의한다. 이것이 표 7의 최종 테스트 케이스이다.

표 4 객체 카테고리 분할

| 클래스 | LB (Low Boundary) | LB+1, … HB−1 | HB (Hihg boundary) |
|---|---|---|---|
| ATM | 1 | N/A | N/A |
| KeyPay | 1 | N/A | N/A |
| Bank | 1 | N/A | N/A |
| Customer | 1 | N/A | N/A |

### 3.6 수강신청 시스템의 테스트 기법 적용 사례

실험의 신뢰성을 더하기 위해 ATM 시스템보다 큰 규모의 시스템을 대상으로 각 기법을 적용하였다. 실험

표 5 OCL 기반 테스트 케이스

| 사용사례 | 테스트 데이타 | 테스트 스크립트 | 예상결과 |
|---|---|---|---|
| PsetCustomerNumber() | Pstate1 | pad.setValue() | F |
| | Pstate2 | | T |
| | Pstate3 | | F |
| | Pstate4 | | F |
| PsetCustomer()1 | Pstate1 | theBank.findCustomer(customerNumber,pin)-customer | F |
| | Pstate2 | | F |
| | Pstate3 | | T |
| | Pstate4 | | F |
| PsetCustomer()2 | Pstate1 | theBank.findCustomer(customerNumber,pin)<>NULL | T |
| | Pstate2 | | F |
| | Pstate3 | | F |
| | Pstate4 | | F |
| PsetCustomer()3 | Pstate1 | currentCustomer.getCheckingAccount() | F |
| | Pstate2 | | F |
| | Pstate3 | | F |
| | Pstate4 | | T |

표 6 ATM 현금 인출 사용사례

| 에이전트 | 액션 |
|---|---|
| ATM | 고객 번호 입력 요구 화면 출력 |
| 사용자 | 고객 번호 입력 |
| ATM | 비밀번호 입력 요구 화면 출력 |
| 사용자 | 비밀번호 입력 |
| ATM | 계좌 선택 |
| 사용자 | 현금 선택 |
| ATM | 금액과 작업 선택 화면 출력 |
| 사용자 | 금액과 작업 선택 |
| ATM | 고객에게 현금 전달 |
| 사용자 | 현금 인출 |
| ATM | 계좌 선택 화면 출력 |
| 사용자 | 종료 버튼 |
| ATM | 고객 번호 입력 요구 화면 출력 |

대상 시스템인 수강신청 시스템은 9개의 클래스 그리고 59개의 멤버 변수와 51개의 메소드로 이루어져 있는 약 1000라인 정도이다.

그림 10은 수강신청 시스템의 클래스도이다. 수강신청 관리자는 해당 학기에 필요한 과목들을 개설하고 학생들은 개설된 강좌에 수강을 신청한다. 수강신청을 하는 과정에서 해당 과목에 대한 선수과목 이수 여부와 잔여 좌석의 수 등을 검사하는 과정이 있다. 학생은 수강하고자 원하는 과목을 추가, 삭제할 수 있고 지금까지 이수한 과목들을 모두 볼 수 있다. 이와 같은 수강 신청에 관한 기본적인 기능을 구현하고 있는 시스템에 대해 위에서 제시한 다섯 가지의 테스트 기법들을 실험하였다.

그림 11은 수강신청 프로그램을 구동하고 사용자가 시스템에 접속하여 수강신청을 실행하고 있는 사용자 화면이다.

ATM을 대상으로 적용된 5가지의 테스트 기법이 수강신청 시스템에도 동일하게 적용되었다. 각 테스트 기법을 적용하고 난 후의 테스트 케이스와 관련 있는 변수, 메소드 그리고 이들이 접근하고 있는 시스템의 범위는 부록 2에 첨부하였다.

## 4. 테스트 기법의 테스트 케이스 추출 특성 비교

대부분의 테스트 케이스를 추출하는 기법은 각각의 고유한 방법과 절차를 따르고 있어 테스트 케이스의 형태나 자료의 양, 범위, 테스트 용이성 등에 영향을 미친다. 따라서 테스트를 수행하기 전에 각 기법의 특징들을 파악하는 일은 중요하다. 본 장에서는 ATM과 수강신청 시스템을 대상으로 테스트 케이스를 추출하면서 파악된 테스트 기법들에 대한 특징들을 정리였다.

테스트 케이스를 작성하면서 접근하는 시스템 영역을 그림 10에서 보여주고 있다. 그림 10은 시스템 영역을 시스템 외부 영역과 모델링 영역 그리고 프로그램 영역

표 7 확장 사용사례 기반 테스트 케이스

| 테스트 스크립트를 포함하는 사용 사례 | 입력 | 예상결과 | 비고 |
|---|---|---|---|
| ATM.ATM()->pad.clear() | | | 데이타 입력 전 |
| AButtonListener.actionPerformed()->ATM.setCustomerNumber()->pad.setValue() | "1" | | 데이타 입력 |
| ATM.setState()->pad.clear()->display.setText("Enter PIN A=OK") | | " " | 데이타 입력 후 |

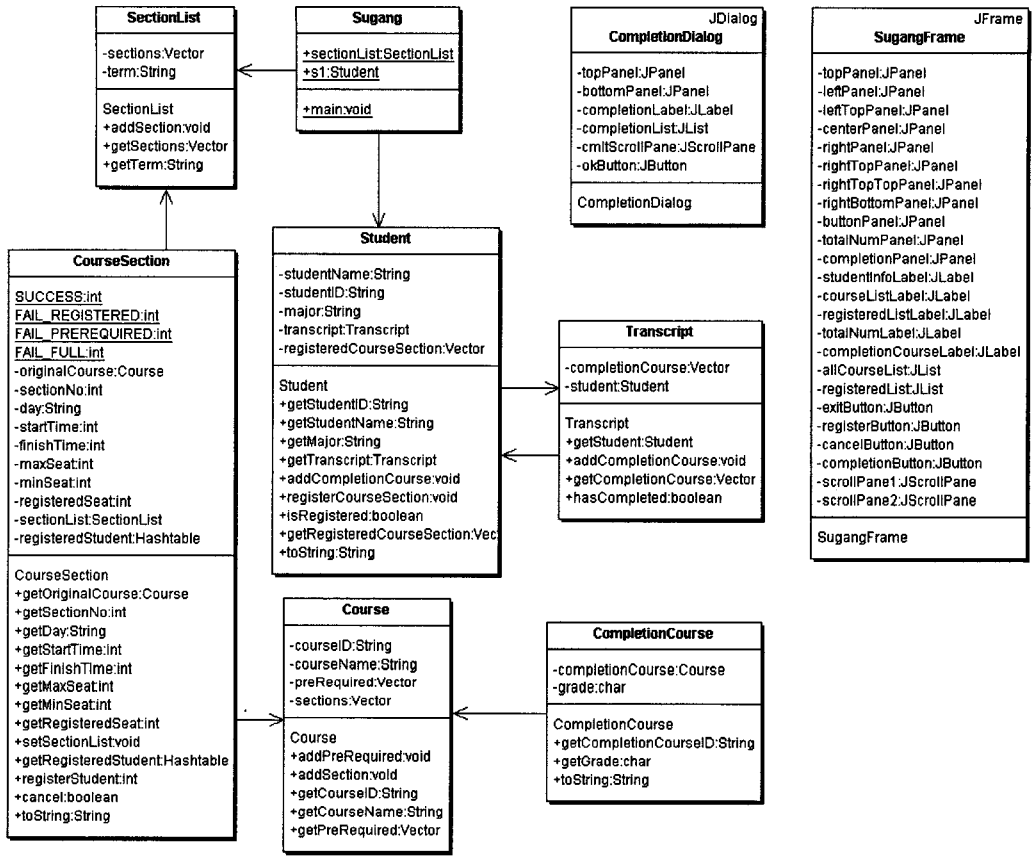



그림 10 수강신청 클래스 다이어그램

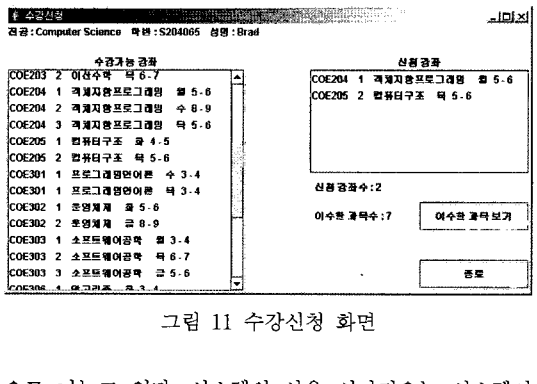



그림 11 수강신청 화면

으로 나누고 있다. 시스템의 사용 시나리오는 시스템이 제공하는 서비스에 따라 사용자가 사용 가능한 일련의 작업 흐름을 말한다. 사용자가 시나리오를 중심으로 시스템을 사용할 때 시스템 내부적으로는 시나리오의 기능 처리와 관련된 변수, 함수, 알고리즘, 자료구조 등이 쓰인다. 외부영역이란 여기서 내부의 자료 처리와 구조에 대한 정보를 고려하지 않고, 단순히 시나리오를 따라 작동하는 내부 멤버 변수와 메소드를 말한다. 즉, 시스템 외부에서 시스템 내부의 사항을 고려하지 않고 테스트 가능한 테스트 케이스 대상 영역을 말한다. 객체지향에서의 경계 클래스 또는 바운더리 클래스가 이에 속한다. 이러한 영역에서의 테스트 케이스 작성 기법은 대부분 블랙박스 유형이 된다. 왜냐하면 시스템 내부의 자세한 사항은 고려하지 않기 때문이다. 모델링 영역은 시스템 외부 영역보다는 자세한 시스템의 정보를 담고 있는 모델링 요구명세서를 통해 접근과 추출이 가능한 테스트 케이스의 대상 범위를 말한다. 시스템 내부의 영역은 화이트 박스의 유형처럼 시나리오가 진행되는 동안 사용되고 접근되는 소스 코드 수준의 테스트 케이스 대상 범위를 말한다. 사용 사례 기반 기법의 경우는 시스템 내부를 전혀 고려하지 않고 있다. 하지만 확장된 사용사례 기반 기법의 경우 테스트 엔지니어가 테스트하고자 하는 사용 사례를 결정하고 시나리오를 작성 한 후 발견된 문제점에 대해 프로그램 영역으로 자세히 접근하면서 시스템 내부의 세부적인 테스트 케이스를 추출 할 수 있다. 예를 들어 특정 시나리오를 기본으로 테스트 케이스를 작성하여 테스트를 수행하다가 노드 D에 오류

가 발견되었다면 노드 D에 대해 시스템 내부를 고려하여 소스 코드에 관련된 테스트 케이스를 또 다시 구체적으로 생성할 수 있다.

반면 Object-Z의 경우는 시스템의 내부 영역에 있는 메소드를 통해 이들의 관계성을 파악하여 생성 가능한 시나리오를 구하고 각 시나리오를 시험한다. 그림 10에서 생성 가능한 시나리오는 (A, B, D, E)와 (A, C, D, E)이므로 두 시나리오의 경우를 테스트할 수 있다. 하지만 OCL의 경우는 전혀 시나리오와 관계없이 테스트 케이스를 추출한다. 즉, 특정 상태 또는 메소드나 멤버 변수 그리고 클래스 간의 관계들만으로 테스트 케이스를 추출한다. 다시 말하면 테스트 엔지니어가 테스트하고자 하는 대상과 관계를 어떻게 분할할 것인가에 따라 테스트 케이스의 양이 결정된다고 할 수 있다.

또한 테스트 케이스를 추출하기 위해 접근하는 시스템 영역의 방향에 따라 테스트 케이스 기법의 특징도 차이가 난다. 예를 들어 협력도 기반 기법과 Object-Z 기반 기법은 모두 모델링 영역에서 테스트가 시작되지만 각 기법을 적용하면서 접근하는 영역은 서로 다르다. 협력도 기반 기법은 프로그램 영역인 소스 코드 레벨로 진행이 되고 Object-Z의 경우는 시스템 기능을 위한 시스템 테스트 레벨로 진행된다. 따라서 테스트 자료가 클래스 다이어그램 중심의 모델링 자료 위주이면서 시스템의 기능을 중심으로 테스트하고자 한다면 Object-Z 기반 기법을 사용할 수 있고 반면 모델링 자료를 이용하여 프로그램 소스의 구체적인 흐름과 작동을 테스트하고자 한다면 협력도 기반 테스트를 선택할 수 있다.

## 5. 커버리지 비교

테스트 기법들 중에서 적절한 테스트 기법을 결정하기 위해서는 테스트 케이스 생성의 효율성, 테스트 대상이 위치하는 시스템 내의 수준, 개발 프로세스 관점에서의 진행 단계 등 다양한 관점에서의 검토가 필요하다. 그 중 테스트 커버리지는 테스트 업무의 효율성과 관련이 깊다. 테스트 커버리지는 테스트 케이스를 이용하여 테스트를 수행할 때, 테스트 케이스에 의해 테스트 대상이 되는 시스템의 범위를 말한다. 즉, 테스트 케이스를 작성하는 방법에 따라 테스트 작업에 영향을 받는 시스템의 범위는 달라진다. 예를 들어, 동일한 시스템에 대해 화이트 박스에서의 문장 검증 기준, 선택 검증 기준, 경로 검증 기준 등 각각의 기법을 이용하여 테스트 케이스 생성하고 그 테스트 케이스가 접근하는 시스템의 변수 또는 프로그램 경로의 개수 등을 구해서 비교하면 그 크기는 달라진다. 적은 테스트 케이스로 넓은 시스템의 테스트 커버리지를 얻을 수 있다면 테스트 작업은 효율적이라 할 수 있다. 테스트 기법의 효율성을 판단하기 위해서는 테스트 케이스의 커버리지의 비교가 객관적인 판단 기준이 된다. 이러한 관점에서 실험 대상으로 사용한 5개의 테스트 기법에 대한 테스트 케이스의 커버리지를 비교하였다.

### 5.1 전체 커버리지에 대한 분석

본 논문에서는 비교 기준에 동일한 기준을 적용하기 위해 그림 9의 상태 변환 다이어그램에서 생성할 수 있는 (1, 2, 3, 4, 5.1, 6)의 시나리오를 대상으로 멤버 변수와 메소드를 추출하고 ATM 시스템에서 차지하는 이들의 비율을 구했다. ATM 시스템에서 시나리오를 따라 진행되는 프로그램을 대상으로 사용되는 모든 변수와 메소드를 테스트 자동화 도구를 사용하지 않고 수동으로 찾았다. 그리고 각 테스트 기법에서 추출된 테스트 케이스의 테스트 데이타가 되는 멤버 변수와 메소드들의 개수를 계산했다. 그리고 테스트 데이타의 전체에 대한 각 테스트 케이스의 멤버 변수와 메소드의 비율을 계산하였다. 테스트 케이스에서 쓰인 멤버 변수와 메소드를 가려내고 계산한 내용은 부록에 추가하였다.

표 8을 통해 확장 사용 사례의 경우는 84%, 그리고 협력도와 Object-Z의 경우는 44%를 나타내고 있다. 확장 사용 사례의 경우 Object-Z나 협력도에 비해 2배 정도의 높은 커버리지가 나왔다. 그 이유는 확장 사용 사례 적용의 경우 블랙박스에서 화이트 박스, 즉 코드 단위 레벨로 하향식 테스트를 진행하므로 시스템 내부에서 발생하는 논리적인 프로그램의 흐름뿐만 아니라 모든 이벤트들을 관찰할 수 있다. 하지만 후자의 경우는 시나리오를 추출하는 방법이 상태 변화 또는 객체간의 관계에 근거하여 블랙박스 테스트 레벨로 진행되어 확인할 수 있는 테스트 요소들의 범위가 감소하기 때문이다.

이와는 다르게 테스트 요소, 즉 클래스간의 관계 또는 데이타 멤버와 메소드, 데이타 멤버와 멤버, 메소드와 메소드 등을 중심으로 의존도와 다중도를 분할하고 있는 OCL의 적용 기법의 경우는 74%의 접근 범위를 나타내고 있다.

그리고 사용사례 기반 기법은 커버리지가 27%로 가

표 8 동일 시나리오에 대한 커버리지

| | | 사용사례 | 협업도 | Object-Z | OCL | 확장사용사례 |
|---|---|---|---|---|---|---|
| 적용 대상 | 변수 (24개) | 10 | 10 | 11 | 21 | 23 |
| | 메소드 (27개) | 4 | 12 | 11 | 16 | 19 |
| 총51개 중 (개) | | 14 | 22 | 22 | 37 | 42 |
| 커버리지 (%) | | 27% | 44% | 44% | 74% | 84% |

장 낮게 분석되었다. 그 이유는 표 8에서도 알 수 있듯이 사용사례기법은 완전한 블랙박스 기법으로 시스템 내부와의 경계인 UI(User Interface)와 관련된 요소에만 커버리지가 미치고 있기 때문이다.

### 5.2 KeyPad 클래스에 대한 분석

표 9를 통해 각 기법들 중 KeyPad 클래스에 대한 커버리지의 분석 결과는 각 기법에 대한 차이점을 보여주고 있다. 확장 사용 사례의 경우와 협력도 그리고 OCL의 경우는 커버리지가 모든 데이타 멤버 변수와 메소드를 거의 접근하고 있지만 Object-Z의 경우에는 0%이다. KeyPad 클래스는 시스템이 외부에 있는 사용자로부터 데이타 값을 입력 받기 위해 사용자에게 보여주는 인터페이스를 위한 클래스이다. 이러한 사용자 인터페이스에 대한 테스트 케이스의 접근이 Object-Z의 경우에는 전혀 나타나지 않고 있다. 왜냐하면 KeyPad는 시스템의 상태 변화를 일으키는 외부 데이타의 입력만을 위해 쓰이고 내부적으로는 시스템의 작동을 나타내는 상태 표현과는 상관이 없기 때문이다. 따라서 사용자 인터페이스 구성과 같은 KeyPad 클래스의 경우는 Object-Z의 기법을 통해 생성된 테스트 케이스로는 접근할 수 없음을 알 수 있다.

또한 표 9에서 두드러진 특징은 사용사례 기반 기법과 확장 사용사례 기반 기법의 커버리지 차이이다. 이러한 차이점은 같은 블랙박스 기법이면서 동시에 사용사례를 기반으로 테스트를 시작 한다 하더라고 테스트 케이스 추출 과정에서 시스템 내부의 프로그램 구성 요소에 대해 고려했는지의 여부에 근거가 있다 하겠다.

그리고 사용사례 기반 기법이 KeyPad의 메소드 커버리지에서 0%가 나왔다. 이러한 특징의 원인은 사용사례 기반 기법과 확장 사용사례 기법의 커버리지 차이점에서 잘 알 수 있다. 동일한 블랙박스 기법이면서 동시에 사용사례를 기반으로 한 두 기법이지만 테스트 케이스 추출 과정의 접근 방법이 다르다. 사용사례 기반의 경우 표 9의 빗금 영역에서 보듯이 시스템 내부는 전혀 접근하고 있지 않다. 즉 KeyPad가 화면을 구성하는 동작에 대해서는 접근하지 않고 있다. 따라서 Object-Z 기반 기법과 같이 메소드 커버리지는 0%가 나왔다.

### 5.3 수강신청 시스템 커버리지 분석

실험을 위한 시나리오는 학생이 수강신청 시스템에 접근한 후 그림 12에서 보이는 왼쪽의 수강가능강좌에서 오른쪽의 수강신청강좌로 등록하는 과정이다. 이와 같은 시나리오를 각 기법에 동일하게 적용하였고 그에 대한 테스트 결과의 커버리지는 부록에 첨부하였다. 그리고 테스트 기법을 적용하는 과정에서 테스트 데이타 또는 테스트 대상으로 접근되는 멤버 변수와 메소드를 추출하고 그 커버리지에 대한 실험 결과를 표 10에 정리하였다.

표 10에서 보는 바와 같이 단순사용사례 기반 테스트

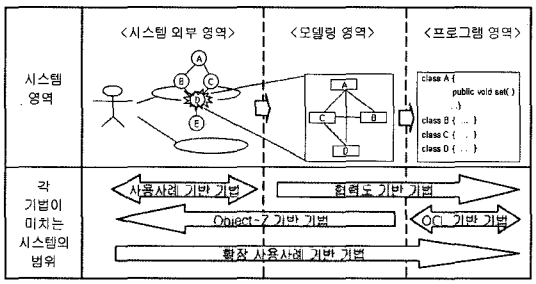



그림 12 각 기법의 시스템 접근 방법의 차이

표 10 수강신청 시스템 커버리지

| | | 사용사례 | 협업도 | Object-Z | OCL | 확장사용사례 |
|---|---|---|---|---|---|---|
| 적용 대상 | 변수 (59개) | 33 | 25 | 23 | 47 | 53 |
| | 메소드 (51개) | 12 | 26 | 30 | 26 | 36 |
| 총110개 중 (개) | | 45 | 51 | 53 | 73 | 89 |
| 커버리지 (%) | | 41% | 46% | 48% | 66% | 81% |

표 9 클래스에 대한 커버리지 분석

| | 적용 기법 | 클래스 | | | | | 합계 |
|---|---|---|---|---|---|---|---|
| | | ATM | KeyPad | Bank | Customer | BankAccount | |
| 데이타 멤버 변수 | 사용사례 | 7/17 | 2/3 | 0/1 | 0/2 | 0/1 | 9/24 |
| | 협업도 | 6/17 | 3/3 | 0/1 | 1/2 | 0/1 | 10/24 |
| | Object-Z | 11/17 | 0/3 | 0/1 | 0/2 | 0/1 | 11/24 |
| | OCL | 14/17 | 3/3 | 1/1 | 2/2 | 1/1 | 21/24 |
| | 확장사용사례 | 16/17 | 3/3 | 1/1 | 2/2 | 1/1 | 23/24 |
| 메소드 | 사용사례 | 3/10 | 0/4 | 0/4 | 0/4 | 0/4 | 3/26 |
| | 협업도 | 5/10 | 3/4 | 1/4 | 1/4 | 2/4 | 12/26 |
| | Object-Z | 5/10 | 0/4 | 2/4 | 2/4 | 2/4 | 11/26 |
| | OCL | 5/10 | 3/4 | 2/4 | 3/4 | 3/4 | 16/26 |
| | 확장사용사례 | 8/10 | 3/4 | 2/4 | 3/4 | 3/4 | 19/26 |

41%, 협력도 기반 테스트 46%, Object-Z 기반 테스트 48%, OCL 기반 테스트 66%, 확장사용사례 기반 테스트 81%로 ATM 시스템 테스트 실험 결과와는 약간의 차이가 있지만 각 테스트 케이스가 차지하고 있는 커버리지 범위는 단순사용사례 기반 테스트 기법에서부터 확장사용사례 기반 테스트로 가면서 더욱 더 넓어지는 것을 알 수 있다. 이러한 차이는 이미 5.1절에서 언급한 바와 같이 각 테스트를 실험할 때 접근되는 시스템의 영역이 서로 다르고 테스트 목적 또한 다르기 때문이다.

## 6. 커버리지 최대화를 위한 테스트 기법 분석

표 8과 표 10을 통해 커버리지의 수치적 결과는 차이는 있지만 각 기법이 나타내는 커버리지 대소의 비교는 그 결과가 "단순 사용 사례 < 협력도 < Object-Z < OCL < 확장사용사례"와 같다. 특히 OCL은 약 70%가 되고 확장 사용 사례의 경우는 약 80% 정도가 된다. 따라서 만약 테스트 계획 단계에서 테스트 목적으로 넓은 커버리지를 고려해야 한다면 확장 사용 사례 또는 OCL 기반 테스트 기법을 참고할 수 있을 것이다. 그리고 소프트웨어 시스템을 효율적으로 테스트하기 위한 방법으로는 확장 사용 사례와 OCL의 조합을 생각해 볼 수 있다. 커버리지의 범위가 가장 넓은 OCL과 확장 사용 사례 기반 테스트 기법은 3장과 4장에서 본 바와 같이 테스트 케이스를 추출하는 과정에서 서로 다른 관점을 가지고 있다. 확장 사용사례 기반 테스트의 경우는 시스템의 시나리오를 기반으로 블랙박스 테스트가 이루어진다. 이는 시스템 내부의 프로그램의 논리적인 흐름에 대한 테스트를 포함하고 있다. 그러나 OCL 기반 테스트 기법은 논리적인 흐름에 대한 검증이라기 보다는 논리적인 흐름에 참여하고 있는 멤버 변수나 메소드 또는 객체들의 관계들에 대한 테스트이다. 따라서 확장 사용사례를 기반으로 하여 시스템의 논리적 흐름을 대표하는 시나리오 인스턴스를 블랙박스로 테스트하고 오류가 발생했을 경우에 발생 지점을 중심으로 OCL 기반 테스트 기법을 적용하면서 오류를 추적한다면 효율적인 테스트 작업이 될 것이다. 이와 같이 확장 사용사례 기반 기법을 수행한 후 오류 발생을 확인하고 오류 발생 원인 후보에 대한 테스트를 OCL 기반 테스트 기법을 이용하여 집중적으로 테스트 한다면 해당 시스템의 테스트 담당자도 테스트 계획을 논리적으로 수립할 수 있고 한정된 자원도 효율적으로 사용할 수 있을 것이다.

## 7. 결 론

시스템을 구현하기 전에 사용자가 요구하는 목적의 기능들을 명시하는 요구명세서의 작성은 어떤 시스템이나 필수적이다. 또한 시스템을 구현하는 과정에서 최초에 의도한대로 기능을 하고 있는지 검증하기 위해서는 요구명세서를 기반으로 하는 방법이 가장 평이하면서도 용이하다. 이러한 테스트 방법들 중에 사용 사례 기반의 기법과 OCL 등 다섯 개의 테스트 방법 기법을 ATM과 수강신청 시스템에 적용하여 실험하였다. 그리고 테스트 케이스 추출과정에서의 테스트 기법의 특성과 추출된 테스트 케이스의 범위를 비교하였다. 그 결과 테스트 케이스를 추출하는 접근 방법이 기능 중심의 테스트인지 아니면 상태 변환 중심의 테스트인지의 관점에 따라 커버리지의 차이점을 잘 보여주고 있다.

그리고 각 기법들의 특성과 커버리지를 통해 확장 사용사례 기반 기법과 OCL 기반 기법의 조합을 통한 효율적인 테스트 방안도 고려해 보았다. 이와 같이 다양한 기법들에 대한 연구 결과가 풍부하게 잘 정립된다면 테스트 업무를 계획하고 수행하는데 많은 도움이 될 것이다.

## 참 고 문 헌

[1] D. Wood, J. Reis, "Use Case Derived Test Cases," STAREAST *on Software Quality Engineering*, Conf., 1999.

[2] 최은만, "OCL로 기술된 개체지향 설계 명세의 테스트 케이스 생성", 정보처리학회 논문지(D), Vol. 8-D, pp.843-852, December 2001.

[3] E.J. Weyuker, S.N. Weiss, D. Hamlet, "Comparison of Program Test Strategies," ACM Proc. Symposium on Testing, Analysis, and Verification, Key West, Florida, pp.1-10, October 1991.

[4] V.R Basili, R.W. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transaction on Software Engineering, Vol.SE-13, No.12, pp.1278-1296, December, 1987.

[5] R. Hamlet, "Theoretical Comparison of Testing Methods," ACM Proc. Third Symposium on Testing, Analysis, and Verification, Key West, Florida, pp.28-37, December 1989.

[6] L.A. Clarke, A. Podgurski, D.J. Richardson, S.J. Zeil, "A Comparison of Data Flow Path Selection Criteria," *8th IEEE International Conference on* Software Engineering, pp.244-251, August 1895.

[7] L. Lauterbach and W. Randall, "Experimental evaluation of six test techniques, Proc. COMPASS'89, Gaitersburg, MD, pp.36-41, June 1989.

[8] S. Ntafos, "A Comparison of Some Structural Testing Strategies," IEEE Transaction on Software Engineering, Vol. 14, No.6, pp.868-874, June 1988.

[9] A. Abdurazik, J. Offutt, "Using UML Collaboration Diagrams for Static Checking and Test Generation." The Third International Conference on the

Unified Modeling Language (UML'00), Proc. pp.383-395, York, UK, October 2000.
[10] Chun-Yu Chen, "Constructing Usage-Based Testing On Object-Z Formal Specification Based Specification," Ph.D. Dissertation Auburn University, 1999.
[11] Eun Man Choi, "Use-Case Driven Test for Object-Oriented System," Proceedings of the IASTED International Conference, ACTA Press, pp.164-169, August 2001.
[12] Y. G. Kim, H. S. Hong, D. H. Bae and S. D. Cha, "Test cases generation from UML state diagrams," IEE Proc., Vol. 146, No. 4, pp.187-192, August 1999.
[13] S. Ramaswamy, R. Neelakantan, "Software Design and Testing Using Petri Nets: A Case Study Using a Distributed Simulation Software System," 2002 Performance Metrics for Intelligent Systems, Natinal Institute of Standards and Technology, NIST Special Publications, August 2002.
[14] R. Duke, P. King, G. Rose, G. Smith, "The Objects-Z specification Language. Version 1," Technical Report 91-1, Software Verification Research Center, Department of Computer Science, University of Queensland, Australia, May 1991.
[15] G.J. Myers, "The Art of Software Testing," John Wiley & Sons, Inc., 2004.
[16] Stefan Jungmayr, Reviewing Software Artifacts for Testability. Presented at EuroSTAR'99, Barcelona, November 1999.
[17] P. Hsia, "Formal Approach to Scenario Analysis," IEEE Software, pp.33-41, 1993.
[18] K. Koskimies, H. Mossenbok, "Scene: Using Scenario Diagram and Active Text for Illustrating Object-Oriented Programs," Proc. Of 18th International Conference. Software Engineering, pp.366-375, March 1996.
[19] P. Jorgensen et al, "Object-Oriented Integration Testing," Comm. Of ACM, Vol.37, No.9, pp.30-38, 1994.
[20] J. McGregor, T. Korson, "Integrated Object-Oriented Testing and Development Process," Comm. Of ACM, Vol. 37, No.9, pp.57-77. 1994.
[21] E. Weyuker, B. Jeng, "Analyzing Partition Testing Strategies," IEEE Transactions on Software Engineering, Vol. 17, No. 7, pp.703-711, 1991.

서 광 익
2002년 동국대학교 컴퓨터공학과(학사) 2002년 동국대학교 컴퓨터공학과(공학석사). 2005년 동국대학교 컴퓨터공학과(현재 박사 재학). 관심분야는 소프트웨어 테스트, 소프트웨어 품질, 임베디드 소프트웨어 테스트, 테스트 프로세스

최 은 만
1982년 동국대학교 전산학과(학사). 1985년 한국과학기술원 전산학과(공학석사) 1993년 일리노이 공대 전산학과(공학박사). 1985년~1988년 한국표준연구소 연구원. 1988년~1989년 데이콤 주임연구원. 2000년~2001년 콜로라도 주립대 전산학과 방문교수. 1993년~현재 동국대학교 컴퓨터멀티미디어공학과 교수. 관심분야는 객체지향 설계, 소프트웨어 테스트, 프로세스와 메트릭, Program Comprehension

## 부 록

### 1. ATM 테스트 커버리지

#### 1.1 동일 시나리오에 의해 접근 가능한 ATM의 변수 및 메소드

| 객체 | 분류 | 변수,메소드 | 확장사용 사례 | 사용사례 | OCL | Z | Collaboration |
|---|---|---|---|---|---|---|---|
| ATM | 변수 | state | ✓ | | ✓ | ✓ | |
| | | customerNumber | ✓ | ✓ | ✓ | ✓ | |
| | | pin | ✓ | ✓ | ✓ | ✓ | |
| | | currentCustomer | ✓ | | ✓ | ✓ | |
| | | currenAccount | ✓ | | ✓ | ✓ | |
| | | theBank | ✓ | | ✓ | ✓ | |
| | | aButton | ✓ | ✓ | ✓ | | |
| | | bButton | ✓ | ✓ | ✓ | | |
| | | cButton | ✓ | ✓ | ✓ | | |
| | | Pad | ✓ | ✓ | | | |
| | | Display | ✓ | ✓ | | | |
| | | START_STATE | ✓ | | ✓ | ✓ | ✓ |
| | | PIN_STATE | ✓ | | ✓ | ✓ | ✓ |
| | | ACCOUNT_STATE | ✓ | | ✓ | ✓ | ✓ |
| | | TRANSACT_STATE | ✓ | | ✓ | ✓ | ✓ |
| | | CHECKING_ACCOUNT | ✓ | | ✓ | ✓ | |
| | | SAVINGS_ACCOUNT | | | | | |

| 객체 | 분류 | 변수,메소드 | 확장사용 사례 | 사용사례 | OCL | Z | Collaboration |
|---|---|---|---|---|---|---|---|
| ATM | 메소드 | ATM | ✓ | | ✓ | ✓ | ✓ |
| | | setCustomerNumber | ✓ | | ✓ | ✓ | ✓ |
| | | selectCustomer | ✓ | | ✓ | ✓ | ✓ |
| | | selectAccount | ✓ | | ✓ | ✓ | ✓ |
| | | withdraw | ✓ | | ✓ | ✓ | |
| | | deposit | | | | | |
| | | setState | ✓ | | ✓ | | ✓ |
| | | AButtonListener | ✓ | ✓ | | | |
| | | BButtonListener | ✓ | ✓ | | | |
| | | CButtonListener | ✓ | ✓ | | | |
| KeyPad | 변수 | buttonPanel | ✓ | ✓ | ✓ | | ✓ |
| | | clearButton | ✓ | ✓ | ✓ | | ✓ |
| | | display | ✓ | ✓ | ✓ | | ✓ |
| | 메소드 | KeyPad | ✓ | | ✓ | | ✓ |
| | | addButton | ✓ | ✓ | ✓ | | |
| | | getValue | ✓ | | ✓ | | ✓ |
| | | clear | ✓ | | ✓ | | ✓ |
| Bank | 변수 | customers | ✓ | | ✓ | | |
| | 메소드 | Bank | ✓ | | ✓ | ✓ | |
| | | readCustomers | | | | | |
| | | addCustomer | | | | | |
| | | findCustomer | ✓ | | ✓ | ✓ | ✓ |
| Customer | 변수 | pin | ✓ | | ✓ | | ✓ |
| | | customer | ✓ | | ✓ | | |
| | 메소드 | Customer | ✓ | | ✓ | ✓ | ✓ |
| | | match | ✓ | | ✓ | | |
| | | checkingAccount | ✓ | | ✓ | ✓ | |
| | | savingsAccount | | | | | |
| BankAccount | 변수 | balance | ✓ | | ✓ | | |
| | 메소드 | BankAccount | ✓ | | ✓ | ✓ | ✓ |
| | | deposit | | | | | |
| | | withdraw | ✓ | | ✓ | ✓ | ✓ |
| | | getBalance | ✓ | | ✓ | | |

### 1.2 동일 시나리오에 대한 커버리지 결과

| | | 사용사례 | 협업도 | Object-Z | OCL | 확장사용사례 |
|---|---|---|---|---|---|---|
| 적용 대상 | 변수 (24개) | 10 | 10 | 11 | 21 | 23 |
| | 메소드 (27개) | 4 | 12 | 11 | 16 | 19 |
| 총51개 중 (개) | | 14 | 22 | 22 | 37 | 42 |
| 커버리지 (%) | | 27% | 44% | 44% | 74% | 84% |

## 2. 수강신청 시스템 테스트 커버리지

### 2.1 동일 시나리오에 의해 접근 가능한 수강신청 시스템의 변수 및 메소드

| 클래스 | 분류 | 변수,메소드 이름 | 단순사용사례 | 협력도 | Object-Z | OCL | 확장사용사례 |
|---|---|---|---|---|---|---|---|
| SugangFrame | 변수 | topPanel | ✓ | | | ✓ | ✓ |
| | | leftPanel | ✓ | | | ✓ | ✓ |
| | | leftTopPanel | ✓ | | | ✓ | ✓ |
| | | centerPanel | ✓ | | | ✓ | ✓ |
| | | rightPanel | ✓ | | | ✓ | ✓ |
| | | rightTopPanel | ✓ | | | ✓ | ✓ |
| | | rightTopTopPanel | ✓ | | | ✓ | ✓ |
| | | rightBottomPanel | ✓ | | | ✓ | ✓ |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | buttonPanel | ✓ | | | ✓ | ✓ |
| | | totalNumPanel | ✓ | | | ✓ | ✓ |
| | | completionPanel | ✓ | | | ✓ | ✓ |
| | | studentInfoLabel | ✓ | | | ✓ | ✓ |
| | | courseListLabel | ✓ | ✓ | | ✓ | ✓ |
| | | registeredListLabel | ✓ | ✓ | | ✓ | ✓ |
| | | totalNumLabel | ✓ | | | ✓ | ✓ |
| | | completionCourseLabel | ✓ | | | ✓ | ✓ |
| | | allCourseList | ✓ | | | ✓ | ✓ |
| | | registeredList | ✓ | | | ✓ | ✓ |
| | | exitButton | ✓ | | | ✓ | ✓ |
| | | registerButton | ✓ | ✓ | | ✓ | ✓ |
| | | cancelButton | ✓ | | | ✓ | ✓ |
| | | completionButton | ✓ | | | ✓ | ✓ |
| | | scrollPane1 | ✓ | | | ✓ | ✓ |
| | | scrollPane2 | ✓ | | | ✓ | ✓ |
| | 메소드 | SugangFrame | ✓ | | | ✓ | ✓ |
| CompletionDialog | 변수 | topPanel | | | | | |
| | | bottomPanel | | | | | |
| | | completionLabel | | | | | |
| | | completionList | | | | | |
| | | cmltScrollPane | | | | | |
| | | okButton | | | | | |
| | 메소드 | ComletionDialog | | | | | |
| Sugang | 변수 | sectionList | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | s1 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | 메소드 | main | ✓ | ✓ | ✓ | ✓ | ✓ |
| Student | 변수 | studentName | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | studentID | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | major | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | transcript | | | ✓ | ✓ | ✓ |
| | | registerdCourseSection | | ✓ | ✓ | ✓ | ✓ |
| Student | 메소드 | Student | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | getStudentID | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | getStudentName | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | getMajor | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | getTranscript | | | | | |
| | | addCompletionCourse | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | registerCourseSection | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | isRegistered | | | ✓ | ✓ | ✓ |
| | | getRegisteredCourseSection | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | toString | | ✓ | | ✓ | ✓ |
| SelectionList | 변수 | sections | | ✓ | ✓ | ✓ | ✓ |
| | | term | | ✓ | ✓ | ✓ | ✓ |
| | 메소드 | SectionList | | ✓ | ✓ | ✓ | ✓ |
| | | addSections | | ✓ | ✓ | ✓ | ✓ |
| | | getSection | | ✓ | ✓ | ✓ | ✓ |
| | | getTerm | | ✓ | ✓ | ✓ | ✓ |
| CourseSection | 변수 | SUCCESS | | ✓ | ✓ | ✓ | ✓ |
| | | FAIL_REGISTERED | | | | ✓ | ✓ |
| | | FAIL_PREREQUIRED | | | | ✓ | ✓ |
| | | FAIL_FULL | | | | ✓ | ✓ |
| | | originalCourse | | ✓ | ✓ | ✓ | ✓ |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | sectionNO | | ✓ | ✓ | ✓ | ✓ |
| | | Day | | ✓ | ✓ | ✓ | ✓ |
| | | startTime | | ✓ | ✓ | ✓ | ✓ |
| | | finishTime | | ✓ | ✓ | ✓ | ✓ |
| | | maxSeat | | ✓ | ✓ | ✓ | ✓ |
| | | minSeat | | ✓ | ✓ | ✓ | ✓ |
| | | registeredSeat | | | | ✓ | ✓ |
| | | sectionList | | | | ✓ | ✓ |
| | | registeredStudent | | | | ✓ | ✓ |
| | 메소드 | CourseSection | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | getOriginalCourse | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | getSectionNO | | ✓ | ✓ | ✓ | ✓ |
| | | getDay | | ✓ | ✓ | ✓ | ✓ |
| | | getStartTime | | ✓ | ✓ | ✓ | ✓ |
| | | getFinishTime | | ✓ | ✓ | ✓ | ✓ |
| | | getMaxSeat | | ✓ | ✓ | ✓ | ✓ |
| | | getRegisteredSeat | | | ✓ | ✓ | ✓ |
| | | getsetSectionList | | | ✓ | ✓ | ✓ |
| | | getRegisteredStudent | | | ✓ | ✓ | ✓ |
| | | registerStudent | | | ✓ | ✓ | ✓ |
| | | Cancel | | | | | |
| | | toString | | | | | ✓ |
| Course | 변수 | courseID | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | courseName | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | preRequired | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | Sections | ✓ | ✓ | ✓ | | ✓ |
| | 메소드 | Course | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | addPreRequired | | ✓ | ✓ | | ✓ |
| | | addSection | | ✓ | ✓ | ✓ | ✓ |
| | | getCourseID | | ✓ | ✓ | ✓ | ✓ |
| | | getCourseName | | ✓ | ✓ | ✓ | ✓ |
| | | getPreRequired | | ✓ | ✓ | | ✓ |
| CompletionCourse | 변수 | completionCourse | | ✓ | ✓ | | ✓ |
| | | grade | | ✓ | ✓ | | ✓ |
| | 메소드 | CompletionCourse | | | | | ✓ |
| | | getCompletionCourseID | | | | | ✓ |
| | | getGrade | | | | | ✓ |
| | | toString | | | | | ✓ |
| Transcript | 변수 | completionCourse | | | | | |
| | | Student | | | | | |
| | 메소드 | Transcript | | | | | |
| | | getStudent | | | | | |
| | | addCompletionCourse | | | | | |
| | | getCompletionCouse | | | | | |
| | | hasCompleted | | | | | |

### 2.2 동일 시나리오에 대한 커버리지 결과

표 10 수강신청 시스템 커버리지

| | | 사용사례 | 협업도 | Object-Z | OCL | 확장사용사례 |
|---|---|---|---|---|---|---|
| 적용 대상 | 변수 (59개) | 33 | 25 | 23 | 47 | 53 |
| | 메소드 (51개) | 12 | 26 | 30 | 26 | 36 |
| 총110개 중 (개) | | 45 | 51 | 53 | 73 | 89 |
| 커버리지 (%) | | 41% | 46% | 48% | 66% | 81% |