객체지향 정보시스템의 테스팅을 위한 확장된 유스 케이스의 사용과 계층적 상태 기반 테스팅 방법

박 광 호*

A Use of Extended Use Cases and Hierarchical State-Based Testing Methods for the Testing of Object-Oriented Information Systems

Kwang-Ho Park*

Abstract

Object-oriented development methodologies require consistent and seamless object-oriented paradigm to be applied from analysis to testing. Testing must focuses on the state of aggregated objects. This paper suggests testing methods that satisfy such requirements. In order to confirm appropriate implementation of the user requirements, the methods apply extended use cases[Jacobson et al., 1992] that are prepared from analysis stage. Testing must be performed based on the use cases because the user requirements are formally documented in the use cases. The notations of the original use case are modified for the state-based testings. The testing methods consist of a unit testing and *four-level of integration testings. Particularly, the level 0 testing is based on FREE state machine* [Binder, 1995, 1996]. The testing methods have been applied to 3 projects and proved their practicability.

* 한양대학교 경영학과

1. 서 론

일반적으로 클라이언트/서버 환경의 객체지향 정보시스템은 2~3계층 구조로 개발된다. 각 계층에는 다수의 객체가 존재하며 이러한 객체들의 집합적 행동으로 정보시스템이 주어진 기능을 수행한다. 그러나, 3계층 가운데 사용자가 직접적으로 접촉하는 것은 1계층의 폼이다. 폼은 요구 사항 분석에서 정의된 유스 케이스(Use Case)[Jacobson et al., 1992]의 가시적 구현체로서 테스팅의 대상이 된다. 따라서, 일반적인 객체지향 소프트웨어의 테스팅 방법[Perry and Kaiser, 1990, 정인상, 1996, 황현숙, 박만곤, 1995, 최병주, 1995]과는 달리 객체지향 정보시스템의 테스팅은 개별 객체보다는 다수의 객체의 집합체인 폼을 중심으로 하는 것이 현실적이다. 유스 케이스는 요구 분석 단계부터 작성되고 분석, 설계, 구현 단계에서 공통으로 사용되는 도구로서 사용자 요구 사항의 정확한 구현 여부를 테스트하는데 가장 효과적인 기준을 제공한다. 그러나, 테스팅을 위한 유스 케이스는 분석, 설계 단계에서 작성된 유스 케이스와 차이가 있다. 분석, 설계 단계의 유스 케이스는 주로 처리 단계나 비즈니스 로직에 대한 서술적 표현인데 반해 테스팅을 위한 유스 케이스는 상태 중심, 데이터 중심으로 구체적이며 결과 지향적이라는 점이다. 또한, 개별적 유스 케이스에 대한 테스팅뿐만 아니라 다수의 유스 케이스들의 집합적 행동에 대한 테스팅도 강조되므로 유스 케이스간의 연관 관계가 중요하게 대두된다.

정보시스템 품질을 측정하기 위한 11개 특성[DeGrace and Stalhl, 1993]이 있는데, 이 가운데 신뢰성은 테스팅과 직접적으로 관련된 특성이다. 신뢰성이란 1) 시스템이 사용자 요구사항, 즉 기대하는 기능을 정확히 수행하는가를 체크하는 정확성 평가, 2) 에러가 발생하지 말아야 하며 사용자의 미숙한 사용에도 대처할 수 있는 에러 대응성에 대한 평가, 3) 사용자의 사용 편리성을 평가하기 위한 단순성으로 구성된다.

이상의 테스팅 기준을 만족시키기 위한 테스팅 방법은 가장 가능성이 높은 오류에 관심을 두고 테스트 케이스를 생성할 수 있어야 한다. 정상 유스 케이스 뿐만 아니라 비정상, 예외 유스 케이스까지 테스팅할 수 있는 테스트 케이스를 생성해야 정확성 평가가 실현될 것이다. 테스트 케이스를 생성하기 앞서 선행되어야할 작업은 목표 행동에 대한 결과 설정이다. 객체지향 정보시스템에 있어 행동에 대한 판단은 상태에 기초한다. 즉, 어떤 객체가 초기 상태에서 출발하여 이벤트에 의한 전이가 발생했을 때 나타나는 객체의 상태 변화가 목표 행동 결과에 부합되는가를 연속적으로 체크하는 작업이라 할 수 있다. 특히 GUI 기반 정보시스템은 다음 2가지 이유로 상태 기반 테스팅에 적합하다고 알려져 있다. 첫째, 클래스나 유스 케이스의 행동은 유한 상태 기계(Finite State Machine)로 모델화 될 수 있다[Chow, 1978, Howden, 1987, Hopcroft and Ullman, 1997]. 상태는 목표 행동 결과로 제약되므로 발생 가능한 상태는 논리적으로 유한하다고 할 수 있을 것이다. 즉, 실제로 발생 가능한 상태는 무한하나 목표 행동 결과의 경계선을 중심으로 정확성을 판단할 때 대표성을 가진 유한 테스트 케이스만으로도 테스팅 결과를 신뢰할 수 있는 것이다. 둘째, 테스팅 방법은 오류 발견을 지원해야 하는데 오류 발생의 원인을 발견하기 위해서는 유스 케이스를 구성하고 있는 개별 객체의 상태까지 테스팅 해야 하기 때문이다.

일반적으로 시스템 테스팅은 단위 테스팅(Unit Testing), 통합 테스팅(Integration Testing), 시스템 테스팅(System Testing)으로 구성된다[Jones, 1990]. 본 논문에서는 VisualBasic, PowerBuilder, Delphi 등과 같은 비주얼 개발 도구를 사용하여 개발한 객체지향 정보시스템의 단위 테스팅과 통합 테스팅 방법론을 제시한다. 본 논문에서 제시하는 테스팅 방법론은 FREE 상태 기계 모델[Binder, 1995, 1996]을 기반으로 한 레벨 0 테스팅(단위 테스팅)에서 출발하여 레벨 1~3 테스팅(통합 테스팅)까지 4단계로 구성된다.

정보시스템 테스팅 방법에 대한 국내외 연구는 일반적으로 테스팅 작업의 분류, 관점, 기준

등 비방법론적 측면에 초점을 두고 있다[Jones, 1990], [DeGrace and Stalhl, 1993], [Perry and Kaiser, 1990], [정인상, 1996], [황현숙, 박만곤, 1995], [최병주, 1995]. Binder[1995, 1996]의 연구는 유스 케이스 기반 테스팅의 이론적 기반을 제공하였으나 실용적으로 사용할 수 있는 테스팅 절차, 방법을 제시하지는 않고 있다. 따라서, 본 논문은 유스 케이스에 기반하여 실질적으로 정보시스템 테스팅에 적용할 수 있는 실용적 방법을 제시하고 있다. 대성그룹의 3개사, 정보시스템 개발 과정에서 테스팅 방법이 구체화되고 향상되었으며 향후 개발 예정인 다른 계열사에 적용될 계획이다.

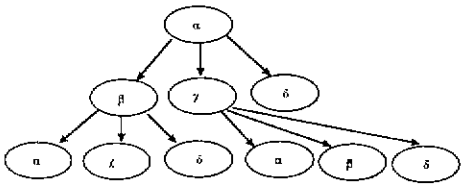
2. 테스팅 프레임웍

2.1 FREE 상태 모델과 테스트 케이스 생성

테스팅을 위한 상태 모델은 인스턴스 변수의 값, 예상 출력 값에 대한 상태들을 정의해야 하며 모든 허용된 메소드나 이벤트의 수행 순서를 지정해야 한다. FREE 상태 모델은 테스팅에 필요한 기반을 제공하고 있는데 메소드나 이벤트의 수행에 따른 객체 상태가 이 모델의 초점이다. FREE 상태 모델에서는 객체의 상태를 다음 4가지 경우로 정의한다.

- 알파(α) : 널(Null) 초기상태
- 세타(δ) : 소멸되는 최종 상태
- 베타(β) : 알파 상태로부터 갈 수 있는 합법적 상태
- 감마(χ) : 알파 상태로부터 갈 수 있는 비합법적 상태

FREE 상태 모델에서 상태는 인스턴스 변수 값들의 조합을 멤버로 하는 제한된 집합으로 정의되는데 FREE 상태 모델은 n 차원 상태 공간을 정의한다. 예를 들어, 3개의 인스턴스 변수를 가진 클래스의 상태 모델은 직육각형로 정의된다. FREE 상태 모델에 따라 순차적인 테스트 케이스가 생성될 수 있다. 테스트 케이스는 테스팅 대상 클래스의 순차적 행동에 따라 전이되는 4개 상태를 기반으로 생성된다. 즉, (그림 1)과 같은 상태 전이 나무에서 노드 사이의 경로의 조합으로 개별 테스트 케이스를 만들 수 있다.



(그림 1) 상태 전이 나무(State Transition Tree)

<표 1>은 상태 전이 나무를 기반으로 생성된 9개의 테스트 케이스 유형을 구분한 것으로 정상, 비정상 테스트 케이스로 구분되었다. 9개의 테스트 케이스 유형 가운데 <완료>, <초기-종료>는 정상적으로 작업을 수행하는 정상 테스트 케이스 유형이다. <완료>는 사용자가 해당 폼(기능)을 능숙하게 사용할 때 따르게 되는 절차로 테스팅에 있어 가장 중요한 테스트 케이스 유형이다. 또한, <초기-종료>는 폼의 초기화 상태만을 확인하기 위한 기초적인 테스트 케이스 유형이다. 9개의 테스트 케이스 유형 가운데 최종 상태가 종료가 아닌 나머지 5개 테스트 케이스 유형은 중도 테스트 케이스 유형으로서 후속 절차가 따르게 되어 있으나 후속 절차는 이미 정의된 테스트 케이스 유형 중 하나일 것이므로 무시하였다. 예를 들어, 최종 상태가 초기인 경우, 초기 상태로 돌아 갔으므로 다시 9개의 테스트 케이스 유형 중 하나로 진행될 것이다. 만일 중도 테스트 케이스 유형이 다시 종료 상태가 될 때까지 따르게 되는 경로를 테스트 케이스 유형으로 분류한다면, 테스트 케이스 유형은 유한하지 않으며 반복되는 경로를 테스팅해야 하므로 의미도 없다. <오류-복귀>, <오류-해결>, <오류-종료> 등 3개 테스트 케이스 유형은 각각, <정상-오류-복귀>, <정상-오류-해결>, <정상-오류-종료> 테스트케이스 유형에 포함되므로 테스트 케이스 유형에서 제외한다. 따라서, <표 1>에 정의된 9개 테스트 케이스 유형 중 <완료>, <초기-종료>, <정상-오류-복귀>, <정상-오류-해결>, <정상-오류-종료> 테스트 케이스 유형 등 5개 테스트 케이스 유형을

기반으로 테스팅을 추진하였다.

〈표 1〉 테스트 케이스 유형

테스트 케이스 유형	상태 경로	구분	최종 상태
복귀	$\alpha => \beta => \alpha$	비정상	초기
정상-오류 복귀	$\alpha => \beta => \chi => \alpha$	비정상	초기
정상-오류 해결	$\alpha => \beta => \chi => \beta$	비정상	진행
정상-오류 종료	$\alpha => \beta => \chi => \delta$	비정상	종료
완료	$\alpha => \beta => \delta$	정상	종료
오류 복귀	$\alpha => \chi => \alpha$	비정상	초기
오류 해결	$\alpha => \chi => \beta$	비정상	진행
오류 종료	$\alpha => \chi => \delta$	비정상	종료
초기-종료	$\alpha => \delta$	정상	종료

2.2 테스트 집합

본 논문에서 제시하는 테스팅 프레임웍은 유스 케이스, 테스트 케이스, 테스트 데이터의 3가지 차원으로 결정되는 테스트 집합을 가진다. 하나의 폼은 복합적 기능을 수행하도록 구현되어 있는데 정상, 비정상, 예외 등 여러 형태의 유스 케이스를 처리하게 된다. 따라서, 테스트 집합은 1차적으로 폼이 몇 개의 유스 케이스를 처리하는가에 달려 있다.

일단 유스 케이스들이 정의되면 각 유스 케이스에 대해 앞서 정의된 5개 테스트 케이스가 존재하게 된다. 유스 케이스별로 테스트 케이스가 중복될 수도 있지만 테스트 케이스가 테스트 집합을 결정하는 2번째 요인이 된다. 3번째 테스트 집합의 결정 요인은 테스트 데이터의 수이다. 특정 테스트 케이스에 대해 여러 개의 테스트 데이터가 사용되는 것이다.

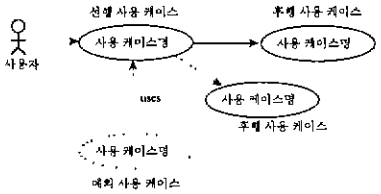
3. 유스 케이스 기반 테스팅

3.1 기본 방법 및 절차

유스 케이스는 Jacobson et al.[1992]에 제시된 요구 사항 분석 방법으로 객체지향 방법론의 중심적 도구 중의 하나로 널리 사용되고 있다. 유스 케이스에 대해서 다양한 정의가 있는데 본 논문에서는 유스 케이스를 폼, 윈도우로 구현되는 단위 기능으로 정의한다. 본 연구에서는 특히 통합 테스팅을 위해 유스 케이스의 기본 표기법을 유지하면서도 유스 케이스간의 흐름을 나타낼 수 있도록 일부 표기법을 수정, 보완하여 사용하였다. 또한, 회계정보시스템을 대상으로 제시된 테스팅 방법을 예시하기로 한다.

3.1.1 통합 테스팅을 위한 표기법

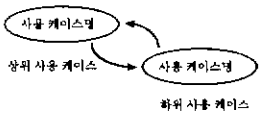
(그림 2)에서 보는 바와 같이 유스 케이스 타원형 내에 유스 케이스명을 표기하여 나타내며 선, 후행 관계를 나타내기 위해 화살표를 사용한다. 그러나, 후행 유스 케이스가 선택적으로 수행되는 경우라면 점선 화살표를 사용하여 나타낸다. 예외 유스 캐이스(Exceptional Use Case)는 유스 케이스간의 유전 관계(Inheritance)를 나타내는데 점선 타원형을 사용하며 유스 케이스에 대한 사용자(Actor)는 점선 화살표를 사용하여 연결한다. 추상 유스 케이스(Abstract Use Case)는 유스 케이스명을 기울임꼴로 표기하여 나타낸다.



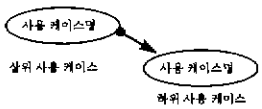
(그림 2) 유스 케이스 - 기본표기법

(그림 3)은 유스 케이스간의 종속 관계를 나타내고 있는데 상위 유스 케이스의 수행 중 하위 유스 케이스로 이동하여 작업 수행을 마치면 다시 상위 유스 케이스로 되돌아 오는 관계를 나타내고 있다. (그림 4)는 유스 케이스간의 생성 관계를 나타내는데 상위 유스 케이스가 하위 유스 케이스를 자동으로 수행하여 사용자가 하위 유스 케이스를 직접 수행할 필요가 없는 경우이다. (그림 5)는 유스 케이스간의 생성 관계의 예외 형태를 나타내는데 상위 유스 케이스가 하위 유스 케이스 작업의 일부만 수

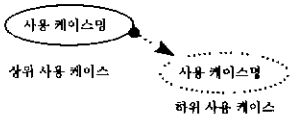
행하고 나머지는 사용자가 직접 수행하여 하위 유스 케이스 작업을 완료하도록 하는 경우이다.



(그림 3) 종속 관계

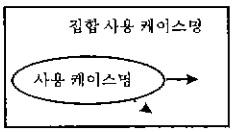


(그림 4) 생성 관계



(그림 5) 생성 관계의 예외 형태

(그림 6)은 유스 케이스 간의 집합 관계(Aggregate Relationship)를 나타내는 것으로 다수의 유스 케이스와 이들 간의 관계를 집합적으로 나타내기 위해 사용한다.



(그림 6) 집합 유스 케이스

3.1.2 테스팅 레벨

유스 케이스 기반 테스팅은 단위 테스팅에서 통합 테스팅까지 4 레벨로 수행된다. 단위 테스팅은 일반적으로 단위 모듈에 내재하는 오류를 발견하기 위한 목적으로 수행된다. 정보시스템 테스팅에 있어 단위 테스팅은 개별 유스 케이스의 기능이 정확히 구현되었는가를 검증하는 것으로 레벨 0 테스팅이다. 전표 등록, 전표승인처리, 영업입금(현금)처리 등과 같은 단위 유스 케이스에 대한 테스팅인 것이다.

통합 테스팅은 일반적으로 단위 테스팅을 통과한 모듈간의 관계상 오류를 발견하기 위한 목적으로 수행된다. 정보시스템 테스팅에 있어 통합 테스팅은 집합 유스 케이스, 즉 다수의 유스 케이스 간의 상호 관계로 구현된 기능이 정확히 구현되었는가를 검증하는 것으로 레벨 1에서 3까지로 구성된다. 통합 테스팅은 단위 테스팅이 완료되었을 때 가능하며 레벨이 올라 감에 따라 내부 구조에 대한 테스팅 보다는 기능 위주의 블랙박스 테스팅으로 전환된다.

레벨 1 테스팅은 프로세스를 대상으로 하는데, 개별 프로세스내에서 수행되는 작업들의 순차적 처리에 대한 기능 테스팅인 것이다. 그러나, 레벨 1 테스팅은 프로세스의 복잡성에 따라 다시 계층적으로 구분될 수 있다. 예를 들어, 전표처리(전표등록⇒전표승인처리) 프로세스에 대한 테스팅은 전표 등록과 이에 대한 전표 승인 처리까지 연결작업의 정확성을 포함하게 된다. 본사-지점전표처리(전표처리⇒본지점피발전표내역등록⇒전표처리) 프로세스에 대한 테스팅은 전표처리라는 하위 프로세스에 대한 테스팅을 포함하고 있다. 영업입금처리-본지점(영업입금(현금)처리⇒본사-지점전표처리) 프로세스는 영업부의 입금처리와 이에 따라 발생하는 본사-지점전표처리 프로세스인데, 내부적으로 본사-지점전표처리 프로세스를 포함하고 있다. 마지막으로 영업입금관리 프로세스({영업입금(현금)처리, 영업입금(받을어음)처리, 영업입금(보통/당좌)처리}⇒{영업입금조회, 법인번호별채권조회, 만기일자별어음현황조회, 입금명세서출력})에 대한 테스팅은 영업입금처리 하위 프로세스를 포함하게 된다. 이와 같이 레벨 1 테스팅은 프로세스의 복잡성에 따라 다양한 수준의 유스 케이스가 정의될 수 있으므로 단계적으로 집합 유스 케이스를 정의하여 순차적으로 테스팅해야 할 것이다.

레벨 2 테스팅은 기능 영역 집합을 대상으로 기능 영역내의 프로세스들의 집합적 행동에 대한 테스팅이다. 일결산({전표관리, 부가세관리, 영업입

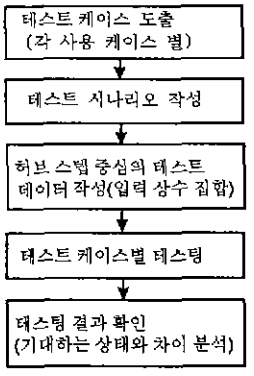
금관리, 받을어음관리, 지급어음관리, 차입금관리, 예적금관리, 대여금관리,...)⇒(일계표출력, 분개장출력, 현금출납장출력, ...)) 집합 유스 케이스는 모든 전표관련 프로세스의 일마감 후, 결산 작업을 수행하는 프로세스로 회계 기능 영역에 존재하는 프로세스들의 집합적 행동에 대한 테스팅인 것이다.

레벨 3 테스팅은 연계 기능 영역 집합을 대상으로 다수의 기능 영역간의 집합 프로세스들의 순차적인 집합적 행동에 대한 테스팅이다. 월마감(납품⇒정산⇒세금계산서발행⇒영업입금-채권) 집합 유스 케이스는 수급, 영업, 회계 등 3개 기능영역간의 관련 집합 프로세스들에 대한 집합적 행동에 대한 테스팅이다.

4단계의 테스팅 레벨에서 유스 케이스의 개별 행동의 오류와 집합적 행동의 오류를 발견하여 신뢰성 있는 정보시스템 구축을 보장할 수 있을 것이다. 레벨 0 테스팅은 개별 행동을 대상으로, 레벨 1 테스팅은 프로세스에 관련된 연계 순차 행동을 대상으로, 레벨 2~3 테스팅은 프로세스 수준을 넘어선 기능영역 수준의 집합적 행동을 대상으로 한다.

3.1.3 테스팅 절차

테스팅은 (그림 7)에서 보는 바와 같이 테스트 케이스의 도출로 시작한다. 테스트 케이스가 확정되면 이에 대한 테스트 시나리오를 작성한다. 테스트 시나리오가 작성되면 테스트 데이터를 준비



(그림 7) 테스팅 절차

해야 한다. 테스트 데이터가 모두 준비되면 실제 테스팅에 들어가게 된다. 테스팅은 테스팅 결과 확인으로 종료된다.

(1) 테스트 케이스 도출

각 유스 케이스에 대한 테스트 케이스는 FREE 상태 모델에서 제시한 상태 기반 기계로 생성하며 앞서 설명한 바와 같이 5개의 테스트 케이스 유형을 기준으로 정상, 비정상 케이스 유형으로 정의하게 된다. 그러나, 9개 테스트 케이스 유형을 반드시 모두 포함할 필요는 없으며 더욱이 비정상 케이스는 단위 테스팅 단계인 레벨 0 테스팅에만 국한한다. 이는 레벨 0 테스팅만이 내부 구조의 오류 발견을 주 대상으로 하지 상위 테스팅은 기능 위주의 블랙박스 테스팅을 대상으로 하기 때문이다. 이는 또한 Petschenik[1985]이 실용적 테스팅을 위해 제시한 "컴포넌트 보다 기능 테스팅이 중요하다.", "비정상 보다 정상 케이스에 대한 테스팅이 우선되어야 하다." 등의 가이드라인에 따른 것이다. 따라서, 레벨 0 테스팅은 FREE 상태 모델에서 생성될 수 있는 가능한 모든 테스트 케이스를 사용하며 레벨 1~3 테스팅은 정상 테스트 케이스를 중심으로 실시한다.

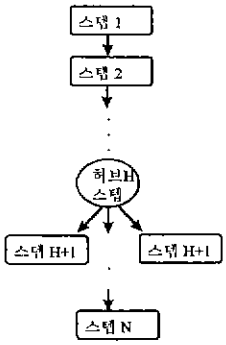
(2) 테스트 시나리오 작성

테스트 시나리오는 구현된 폼이나 윈도우에 대한 작업 순서를 기술한 것으로 유스 케이스의 시나리오가 충분히 구체적이라면 대체할 수 있다. 그러나, 테스팅 작업은 실제로 구현된 컨트롤이나 폼, 윈도우를 지칭해야하므로 재작성하는 것이 바람직하다. 정상 테스트 케이스에 대한 테스트 시나리오를 먼저 작성하고 이를 바탕으로 비정상 테스트 케이스에 대한 테스트 시나리오를 작성하는 것이 현실적이다.

(3) 테스트 데이터 작성

일단 테스트 시나리오가 작성되면 테스트 데이터 집합을 준비해야 하는데 이를 위해서는 우선적으로 허브 스텝(Hub Step, 그림 8)을 기준으로 준

비하는 것이 효과적이다. 허브 스텝이 결정되면 허브 스텝에서 입력될 컨트롤의 값의 집합을 결정하고 이를 중심으로 다른 컨트롤들의 상수 값을 결정하여 테스트 데이터 상수 집합을 완성하게 되는 것이다.



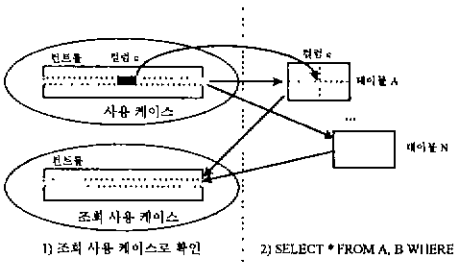
(그림 8) 허브 스텝을 중심으로 한 테스트 데이터 생성

테스팅 대상 객체의 상태를 변경시키는 것은 메소드나 이벤트이다. 순차적 이벤트들의 변수에 대한 모든 상수 집합을 포함하는 테스트 데이터 작성은 현실적으로 불가능할 뿐 아니라 실효성도 없다. 따라서, 논리적으로 다양한 이벤트 발생을 유발시키는 허브 스텝에 존재하는 변수에 대한 상수를 대상으로 테스트 데이터를 작성하게 된다. 결국 테스트 데이터 집합은 허브 스텝에 의해 도출되는 대체 데이터 값들로 정의되는 것이다.

(4) 테스팅 결과 확인

테스트 케이스 시나리오에 따라 테스팅 작업을 수행한 후, 그 결과에 대한 정확성 평가가 수반되어야 한다. 그러나, 작업 결과에 대한 테스팅은 관련된 객체 레벨의 테스팅까지 요구한다. 유스 케이스의 수행 결과, 상태가 전이되는 객체가 기대하는 상태가 되는지 체크하는 것이다. 이는 관계형 데이터베이스의 경우, 테이블에 터플(Tuple)이 삽입, 수정, 삭제되는 상태를 가지고 확인될 것이다. 따라서 경우에 따라서는 유스 케이스의 개별 컨트롤의 상태가 특정 터플의 컬럼 상태로 전달되는 상세 레벨까지의 테스팅이 요구된다.

(그림 9)에서와 같이 테스트 케이스의 수행 결과에 대한 테스팅은 우선, 객체 레벨까지 내려 가지 않고 조회 유스 케이스가 있을 경우, 이를 수행시켜 결과를 확인해 볼 수 있다. 일차적으로 조회 유스 케이스를 통해 결과를 확인하고, 만일 예상하지 못한 결과가 나올 경우, 2차적으로 SQL 문을 수행하여 해당 테이블의 상태를 확인해 볼 수 있다.



(그림 9) 유스 케이스 수행 결과 확인 방법

3.1.4 컴포넌트 테스팅과 테스팅 패턴

유스 케이스 기반 테스팅은 많은 시간을 요구하는 작업으로 정보시스템 개발 프로젝트에 있어 위험 요소가 된다. 따라서, 시간을 절약하면서도 완벽한 테스팅을 완수할 수 있는 방법이 요구된다. 테스팅 작업을 최적으로 수행하기 위해 두가지 방법을 사용하였다. 첫째로, 컴포넌트 테스팅이다. 테스팅이 완벽하게 수행된 컴포넌트를 재사용하여 정보시스템을 개발하게 되면 적어도 사용된 컴포넌트에 대한 테스팅은 피할 수 있게 된다.

테스팅 작업을 최적화하기 위한 두번째 방법은 테스팅 패턴의 사용이다. 테스팅 패턴은 테스팅 과정에서 반복되는 일련의 작업을 일반화시켜 정의한 것이다. 테스팅 패턴이 정의되면 테스팅 세부 작업을 정형화시켜 동일 패턴을 갖는 테스팅 작업을 일관성 있게 수행하게 된다. 레벨 0에서의 테스팅 패턴은 도메인 클래스[Forte, 1995]와 같이 개별 컨트롤 레벨의 패턴에서 템플릿[박광호, 1998]과 같이 폼 레벨까지 존재하게 된다. 또한, 상위 레벨의 집합적 행동에 대한 테스팅 작업이 수행될 때에도 유사한 작업을 반복해서 수

〈표 2〉 테스팅 패턴

테스팅 패턴	테스팅 유스 케이스
선업무처리-부분분개-전표등록완성-당점전표	영업입금처리, 받을어음등록처리, 지급어음발행처리, 지급어음만기결재처리, 지급어음취소처리, 예적금불입처리, 예적금인출처리, 차입금등록처리, 차입금상환처리, 차입금이자상환처리, 외화차입금평가처리, 차입금유동성부채처리, 받을어음할인처리, 할인어음만기결재처리, 받을어음만기결제처리, 받을어음부도처리, 받을어음기타결제처리
선업무처리-부분분개-전표등록완성-본지점전표	영업입금처리, 받을어음등록처리, 지급어음발행처리
선업무처리-부분분개-전표등록완성-지점지점전표	영업입금처리, 받을어음등록처리

행하게 되는 경우가 발생하게 된다. <표 2>는 레벨 1단계에서 발견되는 테스팅 패턴이다. 테스팅 패턴이 정의되면 테스트 시나리오의 작성이나 테스트 케이스 생성에 있어 반복되는 작업을 줄일 수 있을 뿐만 아니라 일관성 있는 작업을 수행할 수 있다.

3.2 레벨 0 테스팅 - 단위 유스 케이스

3.2.1 접근 방법

레벨 0에서의 테스팅은 개별 기능의 정확성에 초점을 맞추기 때문에 기능 구현에 사용된 인터페이스 객체, 비즈니스 객체에 대한 코드 레벨까지의 테스팅을 포함하게 된다. 즉, 레벨 0에서의 테스팅 대상은 윈도우, 폼과 같은 인터페이스 객체, 테이블, 저장 프로시져, 데이터 무결성 관련 기능, 인덱스, 뷰, 트리거 등과 같은 비즈니스 객체가 포함된다. 그러나, 본 연구에서 제시하는 테스팅 방법은 유스 케이스에 기반하는 기능 중심의 테스팅이다. 따라서, 비즈니스 객체에 대한 테스팅은 인터페이스 객체의 테스팅 과정에서 연계하여 테스팅될 수 있으므로 인터페이스 객체를 주 대상으로 테스팅을 실시한다. 그러나, 테스트 케이스에 대한 수행 결과 확인을 위해서 비즈니스 객체에 대한 테스팅을 실시할 것이다.

윈도우, 폼과 같은 인터페이스 객체(Interface Object : IO)는 버튼, 텍스트 박스 등 다수의 컨트롤로 조합된다. 따라서, 인터페이스 객체의 상태는 조합된 컨트롤의 값으로 결정되며, 특히, 컨트롤 가운데 값을 가지는 컨트롤(Value Containing Control : VCC)이 IO의 상태를 결정하게 된다. 개별 VCC는 합법적 값의 범위에 대한 제약 조건을 가지게 된다.

VCC에 값을 입력하는 방법을 결정하는 값 입력 모드에는 외부 입력, 내부 상호 연계 입력, 외부 상호 연계 입력 등이 있다. 외부 입력은 사용자에 의한 직접적인 값의 입력으로 값을 가지게 되는 경우를 말한다. 내부 상호 연계 입력은 내부 컨트롤과 상호 연계된 값을 갖게 되는 다수의 컨트롤을 말한다. 마지막으로, 외부 상호 연계 컨트롤은 외부 객체(데이터 베이스)와 연계된 값을 갖게 되는 다수의 컨트롤을 말한다.

레벨 0 테스팅은 기본적으로 폼 중심으로 수행되며 보다 완벽한 테스팅을 위해서는 개별 컨트롤에 대한 테스팅까지 수행된다. 그러나, 개별 컨트롤 자체에 대한 테스팅까지 포함하게 된다면 테스팅 작업은 매우 많은 시간을 요구할 것이다. 개별 컨트롤에 대한 테스팅을 가능한 줄이기 위해서는 완벽하게 테스팅이 완료된 컴포넌트를 재사용하는 방법을 사용할 수 있다. 따라서, 적용 프로젝트에서 레벨 0 테스팅에서 실제로 개별 컴포넌트에 대한 테스팅을 수행하기 않았다. 컴포넌트와 폼에 대한 테스팅 방법을 다음에 설명한다.

3.2.2 컴포넌트 테스팅

컴포넌트에 대한 일반적 테스트 케이스는 FREE 상태 모델에 따라 다음과 같이 정의된다.

- 초기화 테스팅(α 테스팅):
  초기 값이 채워지는가?
- 최종 테스팅(δ 테스팅):
  최종 전이가 발생했을 때 값이 재 초기화되는가?
- 외부 입력 테스팅(β 테스팅):
  합법적 값에 대한 제약조건이 지켜지는가?
- 외부 입력 테스팅(χ 테스팅):
  비합법적 값이 입력될 때 적절한 메시지가 나타나는가?
- 내부 연계 입력 테스팅(β 테스팅):
  해당 이벤트가 발생했을 때, 연계 컨트롤에 적절한 값이 채워지는가?
- 외부 연계 입력 테스팅(β 테스팅):
  해당 이벤트가 발생했을 때, 연계 컨트롤에 적절한 값이 채워지는가?

3.2.3 폼 테스팅

폼은 유스 케이스의 실질적인 구현체로서 단순히 하나의 기능을 수행하는 폼부터 다양한 기능을 수행하는 폼까지 다양한 형태를 가지고 있다. 일반적으로 폼에서 수행하는 작업은 신규등록, 조회, 수정, 삭제, 출력, 처리 트랜잭션 중에 하나일 것이다. 등록 폼은 6개 트랜잭션을 모두 수행하는 경우도 있으며 조회, 출력, 처리 폼은 단순히 해당 기능만을 수행한다. 하나의 폼에 대한 테스트 케이스는 이와 같이 최대 6개 유스 케이스에 대해 <표 1>에서 정의된 정상, 비정상 테스트 케이스들로 구성될 것이다. 따라서, 폼 하나에 대한 테스트 케이스는 경우에 따라, 수십 개가 될 수도 있다. 각 테스트 케이스에 대한 정상, 비정상 테스트 케이스를 도출하면서 다음과 같은 테스트 케이스에 포함되어야 할 공통적인 부분을 발견하고 이를 중심으로 테스트 케이스를 도출하였다.

- 초기 상태
  폼이 처음 열리거나, 재 초기화되었을 때의 합법적인 상태를 의미하는데, 예를 들어, 윈도우가 처음 열리거나 신규 버튼 클릭 이벤트가 발생했을 때 컨트롤들의 초기 값이 적절히 채워지는가?
- 트랜잭션 수행 전
  트랜잭션 수행 전 상태에 대한 합법적, 비합적인 상태
- 트랜잭션 수행
  트랜잭션 수행시 나타나야 할 시스템의 반응 상태로 예를 들어, 합법적 값에 대한 제약 조건이 지켜지는가? 비합법적 값이 입력될 때 적절한 메시지가 나타나는가?
- 트랜잭션 수행 후
  트랜잭션 수행 후 나타나야 할 시스템의 반응과 결과로 적절한 메시지가 나타나는가? 데이터베이스에 정확히 반영되었는가?
- 종료 상태
  폼이 닫힐 때 합법적인 상태로 예를 들어, 종료 버튼 클릭 이벤트와 같은 최종 전이가 발생했을 때 데이터가 수정되었을 때 이를 저장할 것인지, 아니면 무시할 것인지를 확인하는 메시지가 나타나는가?

3.2.4 레벨 0 테스팅 예

레벨 0 테스팅의 예로 전표 등록 폼에 대한 테스팅을 설명하기로 한다. 전표 등록 폼에는 승인여부 체크박스가 있는데 이 컴포넌트에 대한 테스팅은 다음과 같이 구성된다.

δ 테스팅 - 미승인으로 체크되는가?
δ 테스팅 - 신규버튼을 클릭했을 때 다시 미승인으로 체크되는가?
χ 테스팅 - 해당 없음
β 테스팅(내부 연계) - 해당 없음
β 테스팅(외부 연계) - 승인된 전표 조회의 경우, 승인으로 체크되는가?

전표등록 폼에 대한 유스 케이스는 다음과 같이 도출되었다.

- 정상 케이스
  입금, 출금, 대체 전표 등록
- 비정상 케이스
  분개의 일부만이 채워진 전표 등록
- 예외 케이스

보통예금, 당좌예금 계정관련 전표 등록 경우, 신규 계좌 개설일 경우, 관리 번호 테이블에 신규로 계좌번호가 추가되는 경우와 같이 전표 등록에 따라 예외적으로 특정 테이블에 터플이 추가되는 경우

이상의 유스 케이스 중 입금 전표 등록에 대한 테스트 케이스는 다음과 같이 구성되었다.

- 정상 테스트 케이스
  - 완료 - 정상적으로 입금 내역에 대한 전표를 생성하는 경우로 존재하는 전표 번호를 입력하고 조회했을 때 전표 내용이 정확한가?; 대변 계정 정보를 입력했을 때 차변 내용이 자동적으로 채워지는가?
  - 초기 - 종료 사업장, 발의부서, 발의자, 발의일자, 승인여부, 순번 등의 초기 값이 적절히 채워지는가? 이 상태에서 닫기 버튼을 클릭하면 정상적으로 종료되는가?
- 비정상 테스트 케이스
  - 정상-오류-복귀 – 오류가 발생한 상태에서 신규 버튼을 클릭했을 때 초기 상태로 돌아가는가?
  - 정상-오류-해결 – 현금 계정에 대한 수정이 가능한가? 차대변 오차가 존재하는데 전표가 저장되는가?
  - 정상-오류-종료 – 오류가 발생한 상태에서 닫기 버튼을 클릭했을 때 문제 없이 종료되는가?

입금 전표 등록의 <완료> 테스트 케이스에 대한 시나리오는 다음과 같이 작성되었다.

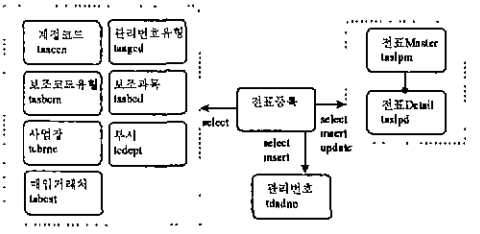
스텝 1. 전표등록 메뉴를 더블 클릭한다.
스텝 2. 사업장, 발의부서, 발의자를 확인한다.
스텝 3. 발의일자와 순번을 확인한다.
스텝 4. 전표구분을 1. 입금을 선택한다.
스텝 5. 순번을 확인한다.
스텝 6. 계정코드를 EDIT하거나 자료사전을 사용하여 입력한다.
스텝 7. 해당 계정코드별 필수 보조, 관리 번호 입력항목이 나타나면 각각 값을 입력한다.
스텝 8. 적요를 기입한다.
스텝 9. 금액을 기입한다.
스텝 10. 계정코드에 대한 필수 입력항목에 대해 각각 값을 입력한다.
스텝 11. 확인 버튼을 클릭한다.
스텝 12. 추가 입력 분개 내역이 있으면 스텝 5로 간다. 아니면, 스텝 13으로 간다.
스텝 13. 차대오차가 0임을 확인하고 저장 버튼을 클릭한다.

이상의 테스트 시나리오에서 스텝 6은 허브 스텝이다. 입력된 계정 코드에 따라 입력 내용이 달라지기 때문이다. 따라서, 스텝 6을 중심으로 테스트 데이터를 준비하였다.

테스팅 결과에 대한 확인은 이는 일종의 테스팅라 볼 수 있는데, 이미 제시한 바와 같이 조회 유스 케이스를 통한 방법과 관련 객체를 통해 확인하는 방법이 있다. 조회 유스 케이스를 통한 방법은 윈도우상의 VCCs와 이에 대응하는 조회 유스 케이스 윈도우의 VCCs와 값을 비교하는 것이다. 전표 등록의 경우, 전표조회/출력 유스 케이스를 사용하여 테스트 케이스 결과를 확인할 수 있다.

관련 객체를 통한 방법은 윈도우상의 VCCs와 이에 대응하는 테이블의 해당 터플의 컬럼과 값을 비교하는 것이다. 전표등록 유스 케이스의 경우, (그림 10)에서 보는 바와 같이 taslpm, taslpd,

```
SELECT * FROM taslpm WHERE slip_date='oct 15 1997' AND slip_no='00001'
SELECT * FROM taslpd WHERE slip_date='oct 15 1997' AND slip_no='00001'
SELECT * FROM tasadno WHERE bojo_sml_code = '111-11-1111'
```



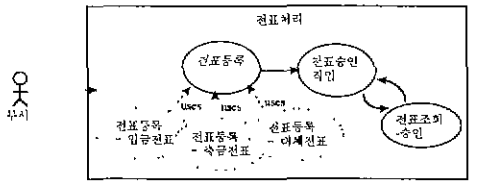
(그림 10) 전표등록에 관련된 객체도

tdadno 등과 같은 테이블의 해당 컬럼과 전표 등록 윈도우의 VCCs를 상호 비교하여 확인할 수 있다. 이를 위해 테스트 케이스 시나리오의 최종 작업으로 다음과 같은 SQL문을 수행할 수 있다.

3.3 레벨 1 테스팅 프로세스 유스 케이스

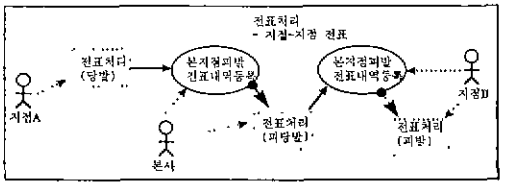
3.3.1 접근 방법

레벨 1에서의 테스팅은 레벨 0에서 통과한 유스 케이스들의 단순 연속 작업이 정확하게 수행되는가를 테스팅하는 것이므로 레벨 0에 비해 코드레벨에 대한 테스팅보다는 기능 중심의 테스팅에 중점을 둔다. 그러나, 앞서 설명한 바와 같이 레벨 1 테스팅은 프로세스의 복잡성에 따라 테스팅 작업이 다시 계층적으로 구분될 수 있다. (그림 11)에서 보는 바와 같이 전표처리 집합 유스 케이스에 대한 테스팅은 전표등록 유스 케이스와 전표승인작업 유스 케이스의 순차적 작업에 대한 테스팅이다.



(그림 11) 단순 순차 유스 케이스 - 전표처리 - 집합 유스 케이스

(그림 12)는 지점간 전표처리 집합 유스 케이스로서 (그림 11)의 전표처리 유스 케이스를 포함한 상위 집합 케이스이다.



(그림 12) 지점간 전표처리 집합 유스 케이스

이상과 같은 다수의 하위 프로세스를 포함하는 상위 집합 유스 케이스의 경우, 선행 유스 케이스의 완료에 의해 발생되는 후행 유스 케이스가 항상 결정되어 있는 것은 아니다. 유스 케이스의 수행 결과에 따라 후행 유스 케이스의 발생이 변경될 수 있기 때문이다. 이렇게 수행 결과에 의해 후행 유스 케이스의 발생이 변경되는 유스 케이스를 허브 유스 케이스(Hub Use Case)라 한다. 예를 들어, 영업입금(받을어음)처리 유스 케이스는 허브 유스 케이스로서 입력된 내용에 따라 후행으로 발생하는 전표처리가 1) 당점 전표 처리, 2) 지점-지점간 전표 처리, 3) 본사-지점간 전표 처리 등으로 달라질 수 있다.

3.3.2 테스트 케이스 작성

이미 지적한 바와 같이 레벨 1 이상의 테스팅에서는 정상 테스트 케이스를 대상으로 테스트 케이스를 작성하게 된다. 테스트 케이스를 작성할 때 주의할 점은 후행 유스 케이스의 테스트 케이스가 선행 유스 케이스의 테스트 상수 집합에 의존적이라는 것이다. 선행 유스 케이스 수행 결과가 정확하다는 가정하에 후행 유스 케이스를 테스팅 작업을 수행함에 있어 선행 유스 케이스에서 사용된 상수 집합이나 일부가 재사용되는 것이다. 선 후행 유스 케이스 수행은 각 유스 케이스에 관련된 객체의 상태를 전이시킨다. 따라서, 테스트 케이스의 수행 결과를 확인하기 위해 해당 테이블에서 생성되거나 수정된 터플의 상태를 체크해야 한다.

이와 같은 상위 단계의 레벨 1 테스팅을 위한 테스트 데이터는 허브 유스 케이스를 중심으로 작성된다. 허브 유스 케이스의 허브 스텝에 입력될 수 있는 값을 기준으로 테스트 데이터를 작성하면 된다. 예를 들어, 영업입금(받을어음)처리에서 허브 유스 케이스는 영업입금처리작업이다. 영업입금처리작업시 입력되는 거래처와 상대 계정은 후속으로 발생하는 전표처리를 결정하는 허브 스텝인 것이다. 따라서, 거래처와 상대 계정의 상수를 기준으로 테스트 케이스를 작성하게 된다.

3.4 레벨 2 : 기능 영역 집합 유스 케이스

3.4.1 개 념

레벨 1 테스팅이 종료되면 기능영역별로 집계, 마감으로 대표되는 프로세스에 대한 테스팅이 요구된다. 레벨 1 테스팅은 영업, 생산, 회계 등 단위 정보시스템에 대한 모든 프로그램의 개발이 완료되었을 때 가능하다. 예를 들어, 회계정보시스템의 일마감은 일계표, 분개장, 현금출납장의 출력이라고 할 수 있는데, 이는 일별 전표관리, 부가세관리, 영업입금관리, 받을어음관리, 지급어음관리, 차입금관리 등 프로세스를 포함하는 회계 기능영역의 집합 사용케이스인 것이다.

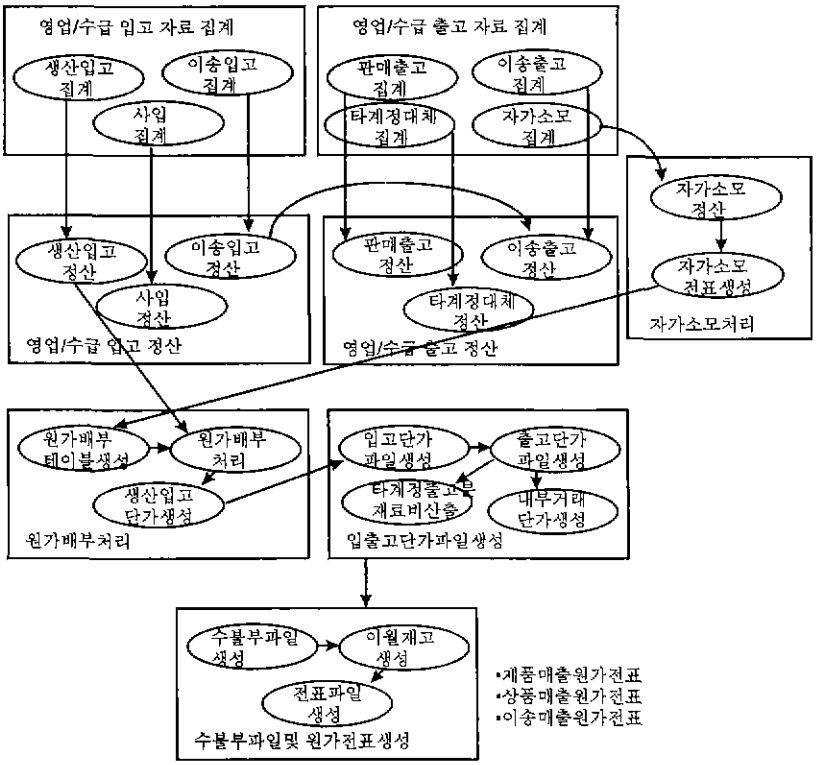
3.4.2 테스트 케이스 작성

레벨 2 테스팅을 위한 테스트 케이스는 실질적으로 일, 월별 등과 같이 정해진 기간내에 발생하는 실 업무 처리 결과를 대상으로 검증하는 것이 현실적이다. 따라서, 레벨 2 테스팅은 병행처리 작업이라고 볼 수 있다. 즉, 기존에 사용하고 있던 시스템과 새로 개발하고 있는 시스템에 병행으로 데이터를 입력하여 두 시스템의 처리 결과를 비교해 보는 것이다. 만일, 기존에 개발된 시스템이 없다면 수기로 작성된 문서와 비교해 볼 수 있다. 중요한 것은 레벨 2 테스팅부터 테스팅 작업은 가상의 데이터를 가지고 하는 것이 아니라 실제로 발생했거나 발생하는 데이터를 가지고 테스팅한다는 것이다.

3.5 레벨 3 : 연계 기능 영역 집합 유스 케이스

3.5.1 개념

레벨 2 테스팅이 완료되면 기능영역별 프로그램에 대한 테스팅이 완료된다. 레벨 3 테스팅은 기능영역별로 완료된 프로그램을 전사적으로 집합적으



(그림 13) 연계 기능영역 집합 유스 케이스 예 - 원가처리

로 처리하는 복수 기능에 대한 테스팅이다. 일반적으로 레벨 2 테스팅이 부서 중심의 마감, 집계 작업에 대한 테스팅이라면 레벨 3 테스팅은 전사적 통합 마감, 집계 작업에 대한 테스팅이라고 볼 수 있다. 레벨 3 테스팅은 다수의 유스 케이스를 포함하는 매우 복잡한 집합 유스 케이스에 대한 테스팅이므로 오류 발생시, 이에 대한 원인 분석이 매우 어렵게 된다. 따라서, 반드시 레벨 0~2 테스팅이 완료된 후 실시해야 한다.

3.5.2 테스트 케이스 생성

레벨 3 테스팅을 위한 테스트 케이스도 레벨 2와 같이 실 데이터를 기준으로 테스트하는 것이 현실적이다. 다만, 레벨 3 테스팅에서는 관련 유스 케이스의 취소와 같은 비정상 유스 케이스에 대한 테스트가 강조된다. 예를 들어 (그림 13)에서 납품 취소처리는 정산 취소처리 후에만 가능하며 정산 취소 처리는 다음과 같은 비즈니스 규칙에 따라 수행되므로 이에 대한 테스트 케이스가 작성되어야 한다. 이와 같이 레벨 3 테스팅은 내부 유스 케이스 사이의 연관 관계를 규정하는 비즈니스 규칙을 요구하게 된다.

- 정산 취소 처리 비즈니스 규칙

1) 세금 계산서 발행 유무
   해당 정산번호에 대한 세금 계산서 발행 취소 후에 정산 취소 처리 가능
2) 정산 구분에 따른 처리 방법
   이월, 부분이월의 경우 :
   정산번호에 해당하는 정산 번호 삭제
   현재 정산 구분 해제('2'로 지정)
   정산내역을 납품내역과 동일하게 맞춤
   물량, 단가, 금액, VAT의 경우 :
   정산내역을 납품내역과 동일하게 맞춤
   정산 구분 해제('2'로 지정)

4. 테스팅 방법의 타당성과 적용 결과

객체지향 방법론의 중심 사상은 개발 전과정에 걸쳐 일관성 있고 연속성 있는 패러다임의 적용이다. 이런 점에서 분석 단계에서 작성하는 유스 케이스는 개발의 마지막 단계인 테스팅 단계까지 확장되어 사용되어야 한다. 따라서, 유스 케이스 중심의 테스팅 방법은 객체지향 방법이 지향하는 일관성과 연속성을 완성한다고 볼 수 있다. 다만 분석 단계의 기능 중심에서의 관점이 테스팅 단계에서는 상태, 데이터 중심의 관점으로 변화하므로 유스 케이스를 확장하는 것이 필요하다. 제시된 테스팅 방법은 상태 중심의 테스팅이 수행될 수 있도록 분석, 설계 단계에서 작성한 유스 케이스를 각 테스팅 레벨에 따라 적절히 확장하는 방법을 제시하였다.

제시된 방법론은 대성산소, 캠브리지필터, 대성계전 등 대성그룹 3개사의 통합정보시스템 개발 프로젝트에 적용되었다. 적용 과정에서 나타난 어려운 점은 다음과 같다.

- 유스 케이스 작성과 테스트 시나리오, 테스트 케이스 작성에 많은 시간이 소요되었다.
- 상위 레벨로 전개되면서 테스트 케이스 생성에 어려움이 있었다.
- 분석, 설계 단계에 작성된 산출물의 부정확성과 불완전성에 기인한 테스팅 자체의 오류가 발견되었다.

그러나 이상과 같은 어려움에도 불구하고 다음과 같은 적용 효과가 보고되었다.

- 분석, 설계, 프로그래밍, 테스팅의 개발 전 단계가 유스 케이스 중심으로 일관성있고 연속성 있게 진행되었다.
- 테스팅을 위한 유스 케이스의 작성 과정에서 전 단계에서 작성된 유스 케이스의 오류가 발견되었으며 다양한 테스팅 케이스 도출로 요구사항이 보다 구체화되어 결과적으로 기능상 오류가 사전에 수정될 수 있었다.
- 테스팅 작업이 조기에 착수될 수 있었으며 보다 철저한 테스팅 작업이 수행될 수 있었다.
- 분석, 설계 단계에서 작성된 유스 케이스 문서가 테스팅 단계에서 검증, 확장됨으로써 유스

케이스 중심의 단일 문서 관리가 실현되었다.
- 유스 케이스를 사용함으로써 분석가, 설계자, 프로그래머의 의사소통이 크게 향상되었다.

적용된 테스팅 방법이 품질이나 프로젝트 기간에 어느 정도 영향을 주었는지 검증하기에는 아직 이르다고 판단된다. 그러나 대성산소, 캠브리지필터, 대성계전으로 정보시스템 개발 프로젝트가 추진되면서 개발 기간이 점진적으로 단축되고 재사용율이 증가하고 있는 것은 간접적으로 테스팅 방법이 개발 생산성이나 품질면에 기여하고 있음을 시사한다고 볼 수 있다.

5. 결 론

본 논문에서 유스 케이스를 기반으로 한 객체지향 정보시스템 테스팅 방법을 제시하였다. 제시된 방법의 특징은 다음과 같이 요약될 수 있다.

- 분석 단계부터 사용하는 유스 케이스를 기반으로하여 일관성 있고 연속성 있는 테스팅 방법을 제시하였다.
- FREE 유한 상태 기계에 의해 테스트 케이스가 생성되는 상태 중심 테스팅 방법이다.
- 단위 테스팅에서 통합 테스팅까지 4단계 레벨의 테스팅을 단계적으로 수행한다.
- 테스트 케이스, 테스트 데이터 생성 방법, 테스팅 결과 확인 절차 등 구체적인 방법을 제시하였다.
- 테스팅을 위해 유스 케이스 표기법을 변형하여 사용하였다.

테스팅은 Jones[1990]이 지적한 바와 같이 전체 개발 일정 가운데 1/2을 소요해야 신뢰할 수 있는 시스템 개발을 보장할 수 있다. 그러나, 일반적인 프로젝트 관리에 있어 테스팅은 전체 일정의 1/5 정도로 일정이 잡혀 있는 것이 보통이다. 어차피 소요되어야했을 기간이 지난 다음에야 시스템의 정상 가동이 가능하다는 것을 인정한다면 아무리 많은 시간이 소요되어도 테스트 케이스를 완벽히 준비하고 철저히 테스팅 작업을 해야 할 것이다. 다만, 테스팅 작업을 효율적으로 하기 위한 방법에 대한 연구가 요구된다. 따라서, 테스팅 작업을 위한 재사용에 초점을 두고 유스 케이스와 테스트 케이스에 대한 재사용 방법에 대한 연구를 진행하고 있다. 제시된 테스팅 방법은 클라이언트 서버 환경에 GUI를 중심으로 개발된 정보시스템을 대상으로 적용될 수 있다. 그러나, 내부 구조보다는 기능에 중점을 둔 방법이므로 기존의 구조적 방법론이나 정보공학 방법론으로 개발된 정보시스템에도 적용될 수 있을 것이다.

참 고 문 헌

[1] 박광호, "정보시스템 개발을 위한 애플리케이션 프레임웍의 설계", 「경영정보학연구」, 제8권 제1호, pp.87-102, 1998.
[2] 정인상, "객체지향 프로그램의 테스팅 방법", 「정보과학회지」, 제14권 제10호, pp.55-65, 1996.
[3] 최병주, "소프트웨어 테스팅 시간 감축을 위한 병렬 테스트 모델", 「정보과학회논문지」, 제22권 제10호, pp.1533-1543, 1995.
[4] 황현숙, 박만곤, "데이터 흐름 분석 기법을 이용한 소프트웨어 테스팅 알고리즘", 「정보과학회논문지」, 제22권 제10호, pp.1425-1433, 1995.
[5] Binder, R., "State-Based Testing," *Object*, Vol.5, No.6, July-August, 1995.
[6] Binder, R., "The FREE Approach for System Testing : Use-Cases, Threads, and Relations," *Object*, Vol.6, No.2, February, 1996.
[7] Chow, T., "Testing Software Design Modeled by Finite State Machines," *IEEE Transactions on Software Engineering*, Vol.4, No.3, pp.173-186, 1978.
[8] Hopcroft, J., and Ullman, J., *An Introduction to Automata Theory*, Lanaguage, and Computation, Addison-Wesley, Reading, MA, 1987.
[9] Howden, W. E., *Fundamental Program Testing and Analysis*, McGraw-Hill, New York, NY, 1987.

[10] Jacobson, I. et al., *Object-oriented Software Engineering*, Addison-Wesley, Reading, MA, 1992.

[11] Jones, G. W., *Software Engineering*, John Wiley & Sons, New York, NY, 1990.

[12] Perry, D. and Kaiser, G., "Adequate Testing and Object-Oriented Programming," *Journal of Object-Oriented Programming*, Vol.2, No.5, pp.13-19, Jan/Feb, 1990.

[13] Petschenik, N. H., "Practical Priorities in System Testing," *IEEE Software*, Vol.2, No.5, pp.18-23, 1985.

■ 저자소개

박 광 호

저자(또는 공동저자) 박광호는 한양대학교 경영학과를 졸업하고, University of Iowa에서 석사, 박사학위를 취득하였으며 삼성SDS에서 AI, 객체지향 업무를 담당하였다. 현재 한양대학교 경상대학 경영학과 부교수로 재직하고 있으며 주관심분야는 AI, 정보시스템 개발방법론, 정보시스템 기획, 전자상거래 분야이다.