Journal of Next-generation Convergence Information Services Technology Vol.4, No.2, December (2015), pp. 105-122
차세대컨버전스정보서비스기술논문지

DbC 접근법과 동치클래스 분류기법을 이용한 객체지향 소프트웨어 테스트케이스 생성방법

A Object Oriented Software Method Using DbC Approach and Equivalence Class Classification

변완섭[1], 김용성[2]*

Wan-Seob Byoun[1], Yong-Sung Kim[2]*

요약

소프트웨어 개발 프로세스는 분석, 설계, 구현, 테스트 및 유지보수 단계를 거친다. 이러한 소프트웨어 개발 프로세스 단계 중에서 여러 단계의 검증도 중요하지만 실제 소프트웨어의 품질을 평가하는데 가장 중요한 단계는 분석과 설계 단계라고 볼 수 있다. 본 논문은 객체지향 소프트웨어 개발 프로세스 단계 중 분석과 설계 단계에서 제작되는 산출물 중 OCL 명세와 처리 문서와 과정을 나타내는 시퀀스 다이어그램에 대해 테스트 모델을 생성하고, 이에 대한 테스트케이스를 생성하여 테스트 드라이브에 활용하는데 목적이 있다.

핵심어 : 소프트웨어테스트, 테스트케이스, 테스트생성기법, DbC 접근법, 동적클래스 분류기법

Abstract

Various test methods are known today to efficiently detect the flaws in computer softwares. Many automated software test tools are also developed to support these test methods and test activities. However, performing a test effectively on a newly developed software system requires a prudent design, and it constitutes a project by itself.

In this paper, a new technique is proposed to create the object oriented software test case models and to generate test cases based on those test technique models, which are based on the sequence diagrams and OCL (Object Constraint Language) contract specifications. These are the by products in the procedure of analysis and design in the process of developing object oriented softwares. The test cases can be applicable not only for performing the test on the object oriented applications but also for writing the test driver for automating the tests.

Keyword : Software test, Test cases, Test generation technique, DbC approach, Equivalence Class Classification

1 Department of Computer Engineering, Chonbuk National University 664-14 DuckJinDong, DuckJin-Gu, Jeonju, Korea e-mail : ysbeon@jbnu.ac.kr

2 Department of Computer Engineering, Chonbuk National University 664-14 DuckJinDong, DuckJin-Gu, Jeonju, Korea e-mail: yskim@jbnu.ac.kr (Corresponding author)

* 이 논문은 변완섭의 2011년도 박사 학위논문에서 발췌 정리하였음

Received(October 08, 2015), Review(October 21, 2015), Accepted(December 02, 2015), Published(December 31, 2015)

ISSN: 2384-101X JNCIST

1. 서론

소프트웨어의 품질을 보장하기 위해 개발 전 소프트웨어 어플리케이션을 평가하고 이를 근거로 개선하는데 많은 노력을 기울이는 영역 중에 하나가 소프트웨어 테스트이다.

이러한 소프트웨어 테스트를 하는 주된 이유는 소프트웨어 개발단계에서 발행하는 문제들을 가능한 빨리 발견하고, 발견한 문제점을 수정확인 함으로써 양질의 소프트웨어를 생산하는데 있다.

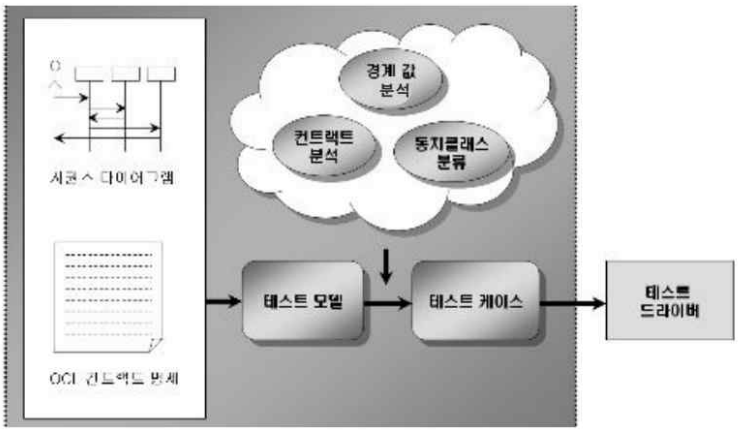
이러한 소프트웨어 테스트의 목적은 각 기능이 수행하는 단계가 옳고 그른지를 확신하는 것이며, 각 단계의 기능에 대해서 믿을 수 있는 테스크 오락클을 필요로 하며, 기능위주의 테스팅은 시스템의 가용성을 확인 하는 데는 커다란 장점을 갖고 있지만 실용성, 상호 작용성, 반복성 등에는 단점을 가지고 있다[1].

테스팅 기법은 소프트웨어를 테스트 할 때 무엇을 수행하는 것에 대해 기술한 일반적인 패턴이며 테스트 공간을 모델링함으로써 시작하게 되고, 그 다음에는 테스트 범위, 테스트 오라클, 활동들을 결정한다. 이후 테스트 중인 시스템을 구성하고, 운용하고, 관찰하며, 결과를 평가하는 일련의 과정을 의미한다.

때로는 이러한 과정들이 매우 빠르게 진행되며 직관적이기 때문에 수행하고 있는 것을 테스터는 알아차리지 못하기도 한다.

일반적으로, 테스트 케이스를 디자인하는데 유일한 한 가지 규칙은 모든 기능을 커버해야 하지만, 너무나 많은 케이스들을 만들지 않아야 한다는 점이다. 그리고 테스터는 매트릭스 기반의 테스트 시트를 사용해 필요한 모든 기능을 테스트하여 볼 수도 있고, 많이 사용하는 동치 클래스 분류 기법과 경계 값 분석을 적용해 중복적인 테스트 케이스들을 제거 할 수도 있다. 또한, 필요할 때는 상태 변이 모델, 결정 테이블, 데이터 흐름 모델을 기반을 적용하여 또 하는 다른 테스트 방법을 사용하기도 한다.

본 논문은 ‘Warm’ 이 제안한 Dbc 접근법과 ‘Blae’ 이 제안한 동치 클래스 분류기법을 기반으로 소프트웨어 개발 프로세스 중에서 분석과 설계 단계에서 발생하는 산출물에 대한 테스트케이스를 생성하여 테스트 드라이브를 생성하는 방법을 제안하는 것을 목적으로 하며, 내용과 범위는 다음 [그림 1]과 같다.



[그림 1] 연구 내용 및 범위
[Fig. 1] Research contents and scope

2. 관련연구

2.1 테스팅 기법 개요

소프트웨어 테스트 기법은 프로그램을 실행시키지 않고 테스팅을 수행하는 정적 테스팅 (Static Testing) 과 프로그램을 실행시켜 테스팅을 수행하는 동적 테스팅 (Dynamic Testing) 으로 크게 구분할 수 있다. 동적 테스팅은 다시 블랙박스 테스팅과 화이트박스 테스팅으로 나눌 수 있으며, 블랙박스 테스팅은 기능적 테스트라고도 하며, 프로그램의 구조를 고려하지 않고 프로그램의 요구사항 명세로부터 테스트 데이터를 선정하여 각 요구사항 부분을 테스트를 하는 것이 특징이다. 반면에 화이트박스 테스팅은 구조적 테스트라고도 하며, 구현된 프로그램 내부 구조를 보고 테스트 데이터를 선정하여 처리결과를 테스트 한다. 화이트 박스 테스트 방법은 이론적이며 분석적이며, 그리고 프로그램의 소스 코드를 이해하면서 동시에 프로세스의 흐름을 파악해야 하므로 단위 테스트 이상의 규모에서 적용하기에는 한계가 있는 단점이 있다.

따라서 실제 현장에서 테스트 기법들이 테스트 업무에 적절하게 적용되도록 다양한 테스트 기법들을 선별하고 비교해서 활용해야 한다.

2.2 테스트기법의 분류

테스트 기법은 소프트웨어를 테스트 할 때 수행할 내용을 기술한 일반적인 패턴으로, 테스트 공

간을 모델링함으로써 시작하게 된다. 그 다음 커버리지, 오라클 (문제를 인식할 수 있도록 해주는 원칙이나 메커니즘), 활동들을 결정하게 되며 테스터는 테스트 중인 시스템을 구성하고, 운용하고, 관찰하고 최종적으로 결과를 평가하게 된다.

많이 활용하는 테스트 기법을 분류하고 정리해보면 다음과 같은 몇 가지로 나눠볼 수 있다[2].

[표 1] 테스팅 기법의 분류
[Table 1] Classification of Testing Techniques

종 류	의 미
기능 테스팅	각각의 단위기능을 확인하고 독립적으로 테스팅 하는 것으로 각 기능의 오락클이 필요하다.
도메인 테스팅	시스템과 관련이 있는 것들을 분류함으로써 테스팅을 단순화 명확화 하는 것으로 영역내 존재 여부 등을 평가한다.
스트레스 테스팅	시스템에 과도한 데이터나 없는 데이터를 입력하여 평가하는 테스팅이다.
흐름 테스팅	정지와 재구성 없이 시스템의 동작시켜 상호작용하는 절차와 흐름을 평가하는 테스팅이다.
시나리오 테스팅	시스템과 유저사이에 스토리를 전개하는 과정으로 문제점을 평가하는 테스팅이다.
클레임 테스팅	누군가 또는 어떤 시스템에 대해 주장에 대한 정량성을 테스팅 한다.
유저 테스팅	유저와 유저모델을 가지고 구축된 시스템을 테스팅 한다.
위험기반 테스팅	위험이 있는 요소들에 대한 테스팅을 한다.
자동화 테스팅	시간과 변이를 초월해서 테스트 할 수 있는 환경을 의미한다.

2.3 동치클래스 분류와 경계 값 분석기법

이론적으로는 테스트하기 원하는 모든 부분들에 대하여 모든 가능한 파라미터들의 조합을 가지고 테스트 케이스들을 생성해야 하는 문제점이 있기 때문에 이를 개선하는 여러가지 방법들이 연구되고 있다. 테스트할 유닛을 몇몇 부분들로 나누는 동치 분류 전략은 한 파티션의 모든 멤버들을 테스트하는 대신 적절한 파티션의 하나의 멤버를 테스트 한다는 기준을 기반으로 한다. 이것은 만약 파티션의 한 멤버를 사용한 테스트 케이스가 실패한다면 이 파티션의 멤버들을 이용한 모든 테스트 케이스들은 실패한다는 것을 의미하며, 유사하게 파티션의 한 멤버를 사용한 테스트 케이스가 성공한다면 이 파티션의 멤버들을 이용한 모든 테스트 케이스가 성공한다는 것을 말한다. 이 접근법의 성공은 명세된 파티션의 품질 전체를 함께 파악할 수 있는 특징을 갖고 있다[1][2].

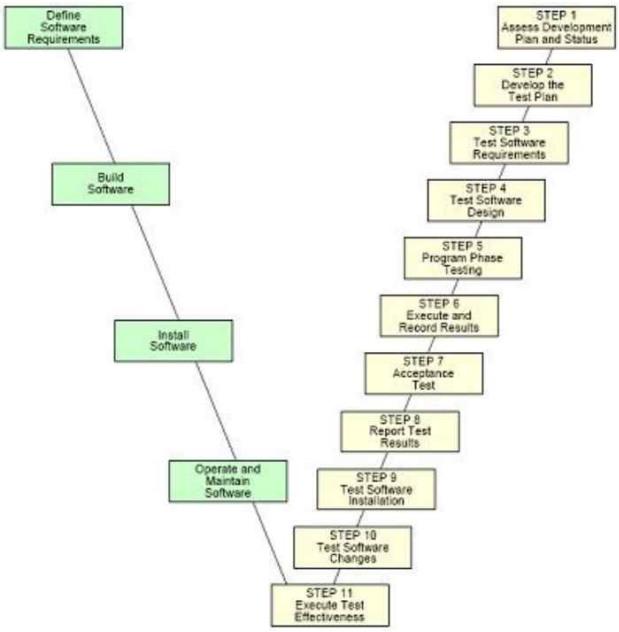
옵션들이 어떤 측면에서든 정렬되어 있을 경우에, 경계치는 동치 클래스 분류상에 존재한다. 다

시 말해서, 옵션이 다른 옵션들에 비하여 '더 크다'거나 '더 작다'라고 표현할 수가 있는 것이다. 이러한 동치분류들 경우에, 특별히 테스트 가능한 특정 요소들을 위한 옵션들을 지적해낼 수 있으며, 이 옵션들은 주로 경계 값 주변에서 발견된다.

경계란 시스템의 기대 행위가 바뀌는 곳을 말하며 경계는 입력치가 유효에서 무효치로 바뀌는 곳에 존재한다. 출력이 디스플레이 영역을 넘어설 수 있는 경우나 내부 프로세스 한계에 도달하는 곳에도 경계가 존재한다.

2.4 테스팅 프로세스

소프트웨어 테스팅 프로세스는 [그림 2] 와 같이 V자 형태의 11개 단계로 수행된다. 단계 1부터 5까지는 소프트웨어 요구사항 명세부터 프로그램 설계과정까지 소프트웨어 개발주기상의 원칙을 지키고 있는지를 검증 (verification) 하는 단계로, 단계 7부터 11까지는 주로 코딩결과의 유효성을 확인 (validation) 하는 단계이다. 유효성 검정결과에 따라 테스트 소프트웨어를 변경하여 코딩결과를 다시 테스트할 수 있다[3].



[그림 2] 소프트웨어 테스팅 프로세스

[Fig. 2] Software testing process

2.5 테스트케이스 추출방법

객체지향 기반의 시스템을 분석,설계하는 UML명세서를 통해 객체지향 언어로 구현한 시스템을 테스트하는 기법들에 대한 비교 연구는 미약한 실정이다. 그러나 소프트웨어 과학의 발전과 함께 소프트웨어의 복잡도가 급격히 높아짐에 따라서 개발 시스템의 이해와 의사소통을 돕기 위해 다양한 요구명세 기법 및 모델링 기법이 연구되고 있다.

요구명세를 이용한 테스트 기법들로는 UML에서 사용하는 유스케이스(Use Case) 이용 기법[4], 콜레보레이션 다이어그램 (Collaboration Diagram) 이용 기법, Object-Z 이용 기법[5], OCL이용 기법[6], 유스케이스를 확장하고 응용한 기법[7], 상태다이어그램 (StateChart Diagram) 이용 기법, 페트리넷 (PetriNet) 이용 기법[8] 등이 있다. 전통적 개발 방법에서 객체지향 개발 방법으로 소프트웨어 개발의 패러다임이 변화되었고 이로 인해 대부분 시스템 개발 구현의 언어로 객체지향 언어를 사용한다. 객체지향 관점에서 시스템을 모델링 하는 UML과 관련 있는 테스트 기법으로는 유스케이스와 콜레보레이션 다이어그램, 확장된 유스케이스, 상태도가 적용된 Object-Z 테스트 기법과 UML에서의 제약사항을 위해 고안된 OCL을 이용한 기법이 있다[9]. 일반적으로 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 하게된다. 따라서 자주 사용되는 UML을 이용한 테스트 케이스 추출 기법은 객체지향 관점에서의 개발에 있어서 중요한 부분이 되며, UML에서 정의하고 있는 여러 표현기법 중에서 유스케이스, 객체지향 소프트웨어를 위한 주요 블랙박스 테스트 기법들의 비교, 단순 유스케이스를 이용한 테스트 케이스 추출 절차와 콜레보레이션 다이어그램을 이용한 테스트 케이스추출 절차 OCL,협업도, 상태도를 통해 객체지향 언어 명세를 위해 수정된 Object-Z를 이용한 테스트 케이스 추출 기법은 UML이 자주 사용되므로 인해 주요한 테스트기법이라 할 수 있다.

2.6 DbC (Design by Contents) 접근법

DbC 접근법은 이러한 소프트웨어의 신뢰성 측면에서 이점을 안고 있기 때문에 여러 방법론에 채택되고 사용되고 있다. 과거 Fusion[10] 에서는 오퍼레이션 정의를 위해 사전조건 및 사후조건이 사용되었으며, UML에서는 사전조건, 사후조건, 불변조건뿐만 아니라 이를 표현하는 OCL이라는 정형명세 언어를 사용하고 있다. 다른 명세 언어가 많은 수학적 지식을 필요로 하는 반면,OCL언어는

많은 수학적 지식이 없어도 DbC 접근법을 이용하여 소프트웨어를 정형적으로 명세할 수 있는 언어로 다음과 같은 특징을 가지고 있다[1][11].

첫째, 요구사항에 대한 제약사항이나 비즈니스 룰을 표현하는 것이 가능하다. 둘째, 테스트 케이스 선정을 위한 경계 값을 알아낼 수 있다. 셋째, 수학적 지식은 필요하지만 다른 명세 언어에 비해 사용이 어렵지 않다. 넷째, 분석가가 의도한 사항이 프로그래머에게 정확히 전달될 수 있다. 다섯째, 시뮬레이션과 테스트, 일관성 체크 및 코드생성을 위해 사용될 수 있다.

3. Dbc 명세를 기반으로 한 테스트모델 생성

테스트 관점에서 최근 중요한 이슈로 등장하고 있는 것 중 하나는 테스트의 기반이 되는 모델의 구성에 관한 것이다[12]. 기존 테스트의 경우 자연어를 기반으로 하는 매우 추상화된 명세와 소스코드를 이용하였다. 그러나 자연어는 테스트 모델로는 적절하지 못하다. 그 이유는 모든 필요한 테스트 가공물을 유추하기에 충분할 만큼 정형적이지 못 할뿐만 아니라 소스코드를 이용하는 화이트박스 테스트의 경우 수반되는 엄청난 노력을 고려한다면 효율적인 모델이 될 수는 없기 때문이다.

3.1 테스트 모델 정의

모델이란 많은 컨텍스트에서 사용되며, 소프트웨어에서 구축에서 사용되는 모델을 다음과 같이 정의하였다.

- 모델은 특성들과 제약사항을 갖는 일관된 모델 구성 요소들의 집합이다.
- 모델 저장소는 자동화된 도구에서 모든 모델 구성요소들의 저장소이다.
- 모델 구성요소들의 시각적 뷰 (view) 는 다이어그램 형태로 나타낸다.

> *[정의 3.1] 테스트 모델은 특성들과 제약사항을 갖는 객체와 객체간의 상호작용들의 집합이며, 자동화된 도구에서 사용 가능하도록 모델 구성 요소들의 저장 형식이 정의되고, 다이어그램 형태의 시각적 뷰로 나타낸다. 테스트 모델은 테스트 케이스 추출을 위한 기반이 되는 모델이다.*

3.1.1 모델의 구성요소

본 논문에서 제안하는 DbC 명세를 기반으로 한 테스트 모델은 유스케이스 수행에서 표현되는 유한개의 객체들과 상호작용들로 이루어진다.

> *[정의 3.2] 개개의 유스케이스는 한단위의 테스트 모델을 구성하며, 테스트 모델의 구성은 유한개의 객체들(Objects)과 객체들간의 유한개의 상호작용(Interactions)으로 구성된다.*
>
> $M = (O, I)$
>
> O : $Objects$
>
> I : $Interactions$

상호작용은 객체들이 교환하는 메시지를 의미하며, 메시지는 7개의 튜플로 구성된다.

> *[정의 3.3] 상호작용은 객체들간에 교환되는 메시지를 의미하며,*
> *메시지는 7개의 튜플로 구성된다.*
>
> $I = (m_n, s_o, q_o, i_p, o_p, C_{pre}, C_{post})$
>
> 여기서, m_n : $message\ sequence\ number$
>
> s_o : $start\ \ object$
>
> q_o : $end\ \ object$
>
> i_p : $inputs$
>
> o_p : $outputs$
>
> C_{pre} : *preconditions*
>
> C_{post} : *postconditions*

3.1.2 시각적 뷰 요소

테스트 모델의 시각적 뷰는 테스트 모델의 구성요소를 그래픽 요소로 표현하며 방향 그래프 G = (N, E, M) 로 다음과 같이 정의한다.

> *[정의 3.4] 테스트 모델의 시각적 뷰는 UML의 콜레보레이션 다이어그램과 유사한 형태의 방향 그래프 G=(N, E, M)로 나타낸다. 노드 N은 메시지 교환의 주체가 되는 객체를 나타내고, 간선 E는 객체들간의 상호작용을 의미하고, 메시지 M은 실제 객체간 상호 호출되는 객체의 오퍼레이션을 의미한다.*

3.1.3 저장형식

테스트 모델을 자동화 도구 내에서 활용하기 위해서는 테스트 모델을 자동화 도구 내에 저장하기 위한 저장 형식이 필요하다. 본 논문에서는 XML을 이용하여 테스트 모델의 저장 형식을 다음과 같이 정의한다.

> *[정의 3.5] 테스트 모델을 저장하기 위한 객체간 메시지 전달(메소드 호출) 정보($m_n, s_o, q_o, i_p, o_p, c_{pre}, c_{pst}$)는 XML 로 나타낸다.*

3.2 테스트 모델 생성기법

3.2.1 생성절차

테스트 모델 생성 과정은 시퀀스 다이어그램과 OCL 컨트랙트 명세를 입력으로 하여 [정의 3.2], [정의 3.3]에 정의한 구성요소들을 완성하였다.

산출물을 만들어 내는 과정은 다음과 같다. [규칙 3.1]은 시퀀스 다이어그램의 구성 정보를 [정의 3.2]와 [정의 3-3]에서 정의한 모델 구성요소로 매핑하기 위한 규칙이다.

> *[규칙 3.1] 시퀀스 다이어그램에서 객체들은 O. 메시지들은 I로 매핑하며, 메시지 순서는 m_n, 메시지 전송 객체는 s_o, 메시지 수신 객체는 q_o, 메시지 입력 파라미터는 i_p, 출력 파라미터는 o_p로 각각 매핑한다.*

[규칙 3.2]는 OCL 컨트랙트 명세를 [정의 3.3]에서 정의한 모델 구성요소로 매핑하기 위한 규칙이다.

> *[규칙 3.2] OCL 컨트랙트 명세에서 사전조건은 C_{pre}, 사후조건은 C_{post}로 매핑한다.*

[규칙 3.1]과 [규칙 3.2]에 의해 생성된 노드, 간선, 메시지를 이용하여 방향성 그래프를 [규칙 3.3]에 의해서 작성한다.

> *[규칙 3.3] 노드는 객체, 간선은 객체간의 관계, 메시지는 오퍼레이션으로 대응시켜 방향성 그래프로 매핑한다.*

객체간 메소드 수행정보와 [규칙 3.3]에서 작성된 방향성 그래프를 이용하여 XML 작성하는 규칙은 [규칙 3.4]와 같다.

> *[규칙 3.4] [규칙 3.1]의 메시지 수행정보와 [규칙 3.3]의 방향성 작성하기위한 관계를 XML 에 매핑한다.*

3.2.2 테스트 케이스 생성 알고리즘

테스트 모델에 관한 정의와 규칙에 따라서 테스트 모델 생성 알고리즘은 다음과 같다.

Input : 시퀀스 다이어그램, OCL 컨트랙트 명세
Output : 테스트 모델의 시각적 뷰와 XML 로 테스트 모델 생성

Step1. 입력 프로세스
- 1-1. 시퀀스 다이어그램 생성
- 1-2. OCL 컨트랙트 명세 생성

Step2. 테스트 모델 생성 프로세스
- 2-1. 시퀀스 다이어그램에서 객체, 메시지, 메시지순서, 메시지 전송 객체, 메시지 수신 객체, 메시지 입력 파라미터, 메시지 출력 파라미터를 추출하여 I={ m_n, s_o, q_o, i_p, o_p}를 구성한다.
- 2-2. OCL 컨트랙트 명세에서 사전조건 Cpre, 사후조건 Cpost를 추출한다.
- 2-3. 객체간의 상호작용 I={ $m_n, s_o, q_o, i_p, o_p, c_{pre}, c_{post}$}를 구성하여 객체간의 상호작용 관계를 구성한다.
- 2-4. 객체간의 상호작용 구성요소를 이용하여 객체들 간의 유한개의 상호작용으로 테스트 모델 M(O, I)를 구성한다.

Step3. 출력 프로세스
- 3-1. 노드는 객체, 간선은 상호작용을 하는 메시지, 메시지는 객체의 오퍼레이션으로 대응시켜 방향성 그래프를 구성한다.
- 3-2. 객체간 메소드 수행정보와 방향성 그래프를 이용하여 XML 인 테스트 모델을 출력한다.

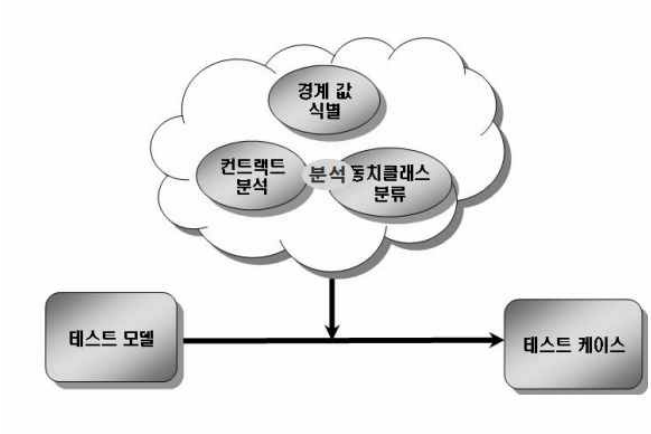
4. DbC 명세를 기반으로 테스트케이스 생성

DbC 접근법 이용하여 시스템을 명세하는 접근법을 말한다. 컨트랙트는 서비스의 공급자(서버)와

사용자 (클라이언트) 사이의 책임 및 의무를 정확히 명세한 것을 말하는 것으로[12], 이 DbC 접근법은 특정 클래스 (서버) 의 오퍼레이션을 사용하기 위한 클라이언트의 책임은 오퍼레이션의 사전조건으로 명세하고, 클라이언트가 책임 (사전조건) 을 완수하였을 경우 클래스가 지켜야 할 약속은 오퍼레이션의 사후조건으로 명세하며, 클래스 자체가 일반적으로 유지해야 할 속성들은 불변조건으로 명세하는 접근법이다[12].

4.1 적용 도메인과 절차

본 논문에서 제안하는 테스트 케이스 생성기법은 제3장에서 제안한 테스트 모델을 기반으로 하여 테스트 모델이 포함하고 있는 정보들을 토대로 하여 컨트랙트 분석, 동치 클래스 분류, 경계 값 분석을 통하여 테스트 케이스를 생성하는 기법이다. [그림 3] 은 테스트 모델로부터 테스트 케이스를 생성해 나가는 과정을 표현하고 있다.



[그림 3] 테스트 모델로부터 테스트 케이스 생성
[Fig. 3] Generate test cases for the test model

적용사례로 사용할 도메인은 유명한 "Royaland Loyal System"의 예제를 활용한다[Warm 03]. 예제 애플리케이션은 온라인상에서 카드 회원 관리시스템으로 객체지향 소프트웨어이다.

4.2 테스트 케이스 생성

본 절에서는 3장에서 제안한 테스트 모델 기법을 적용 도메인 R&L애플리케이션에 적용한 각 산출물을 몇 가지 예를 기술한다.

(1) 'getService'메소드

LoyaltyProgram 클래스와 ProgramPartner 클래스 간에 회원가입과 탈퇴, 인증 등에 처리를 수행하며 파라미터 타입은 string이다. 따라서 나타날 수 있는 테스트 케이스들은 다음 [표 2]와 같다.

[표 2] 테스팅 기법의 분류
[Table 2] Methods of test cases getService

Test No.	Test Item	Inputs	Expected Results	Procedural Requirements	Intercase dependencies
1.1	getService	YSKim	valid name	no	
1.2	getService	YS1234	valid password	no	
1.3	getService	성남시 분당구 율동 323-3	valid address	no	
1.4	getService	전주시@덕진동	invalid address	no	
1.5	getService	$Kim02	invalid password	no	
1.6	getService	02-324-27954	invalid phoneno	no	
1.7	getService	20000324	valid date	no	
1.8	getService	20100229	invalid data	no	

(2) 'makeTransaction'메소드

'LoyaltyProgram'클래스와 'Service'클래스 간에 트랜잭션, 즉 슈퍼마켓, 주유, 차량대여, 항공여행 등의 사건처리에 관한 메소드로 서브클래스는 'Earning'과 'Burning'이다.'makeTransaction'의 파라미터 타입은 (int,Date) 이며, 나타날 수 있는 테스트 케이스들은 다음 [표 3]과 같다.

[표 3] makeTransaction 메소드의 테스트 케이스들
[Table 3] Methods of test cases makeTransaction

Test No.	Test Item	Inputs	Expected Results	Procedural Requirements	Intercase dependencies
2.1	makeTransaction	(25, 2000325)	valid Transaction	yes	1.X
2.2	makeTransaction	(100, 2010425)	valid Transaction	yes	1.X
2.3	makeTransaction	(25.4, 2011514)	invalid amount	yes	1.X
2.4	makeTransaction	(150, 20120230)	invalid date	yes	1.X
2.5	makeTransaction	(200, 20131312)	invalid month	yes	1.X
2.6	makeTransaction	(YSKim, 20120215)	invalid amount	yes	1.X
2.7	makeTransaction	(150000, 20120219)	invalid too high amount	yes	1.X
2.8	makeTransaction	(-300, 20141204)	invalid too low amount	yes	1.X

(3) 'setDate'메소드

'setDate'메소드는 트랜잭션이 발생할 때마다 Date를 체크하는 메소드로 데이터 타입은 Date 이고, 나타날 수 있는 테스트 케이스들은 다음 [표 4]와 같다.

[표 4] setDate 메소드의 테스트 케이스들
[Table 4] Methods of test cases setDate

Test No.	Test Item	Inputs	Expected Results	Procedural Requirements	Intercase dependencies
4.1	setDate	99990204	invalid too high Year	yes	2.X
4.2	setDate	20101302	valid Transaction	yes	2.X
4.3	setDate	20110402	invalid amount	yes	2.X
4.4	setDate	Date	invalid date	yes	2.X
4.5	setDate	년월일	invalid date	yes	2.X

그밖에 'setAmount' 메소드, 'setPoint'메소드, 'AddTreansaction'메소드, 'adjustpoint'메소드는 생략한다.

4.3 결과 분석

본 절에서는 테스트 케이스 생성의 예시를 경계 값 분석, 동치 클래스 분류 기법, 그리고 본 논문에서 제안한 테스트 케이스 생성 기법을 테스트 케이스의 수와 테스트 케이스의 생성 노력도를 비교 고찰한다.

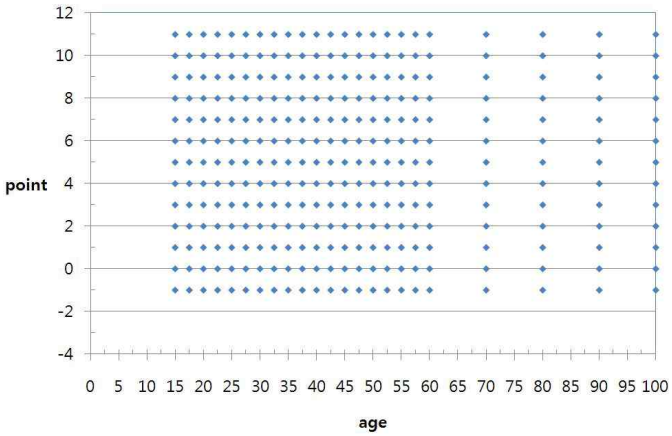
일반적으로, 테스트 케이스를 만드는 노력도는 테스트 케이스의 수에 반비례하는 성질을 갖고 있다. 즉, 테스트 케이스의 최적화는 테스트 목적에 상관없는 테스트 케이스가 얼마나 많이 포함되지 않았는가를 나타낸다. 따라서, 본 논문에서는 경계 값 분석과 동치 클래스 분류 테스팅의 최악의 경우를 아래와 같이 가정한 후 테스트 케이스수를 산정해서 비교해 본다.

연령별 카드 사용한도 포인트 (단위, 천포인트) 를 다음과 같이 구간을 설정한다.

$$A1=\{age: 16 \le age < 25\} \qquad P1=\{point=0,1\}$$
$$A2=\{age: 25 \le age < 35\} \qquad P2=\{point=2,3\}$$
$$A3=\{age: 35 \le age < 45\} \qquad P3=\{point=4,5\}$$
$$A4=\{age: 45 \le age < 60\} \qquad P4=\{point=6,7\}$$
$$A5=\{age: 60 \le age < 100\} \qquad P5=\{point=8,9,10\}$$

1) 경계 값 분석 기법을 적용한 테스트 케이스

경계 값 분석 기법을 적용하여 생성된 테스트 케이스를 좌표 평면에 도시하면 다음 [그림 4]와 같다.

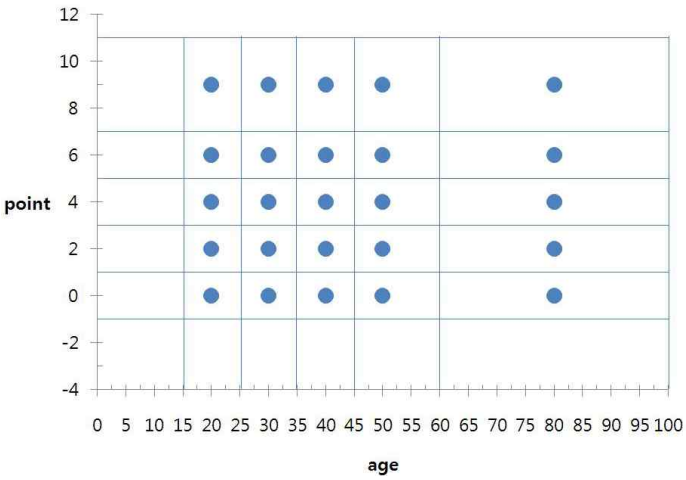


[그림 4] 경계 값 분석에 따른 테스트 케이스의 수

[Fig. 4] The number of test cases according to the boundary value analysis

2) 동치클래스 분류 기법을 적용한 테스트 케이스

동치클래스 분류 기법을 적용한 테스트 케이스를 좌표 평면에 도시하면 다음 [그림 5]와 같다.



[그림 5] 동치클래스 분류에 따른 테스트 케이스의 수

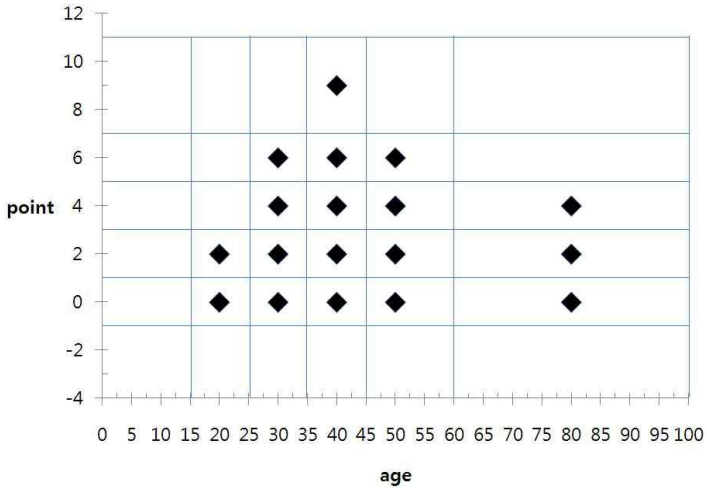
[Fig. 5] The number of test cases based on equivalence classification

[그림 4]의 테스트 케이스 수와 [그림 5]의 테스트 케이스 수를 비교해 보면 13×23=299개와 5×5=25개로 동치 클래스 분류기법이 현저하게 테스트 케이스가 적은 것을 알 수 있다.

즉, 25/299×100=8.3% 정도로 테스트 케이스가 줄어든 것을 알 수 있다.

3) 제안한 방식의 테스트 케이스

본 연구에서 제안된 방식을 활용한 테스트 케이스를 좌표 평면에 도식화하면 다음 [그림 6]과 같다.



[그림 6] 제안한 방식에 따른 테스트 케이스 수

[Fig. 6] The number of test cases according to the proposed method

따라서, 다음과 같이 본 논문에서 제안된 방식이 동치 클래스 분류 방식보다도 테스트 케이스가 현저하게 적음을 알 수 있다.

$$\frac{8\times 6-(16+7+3+2)}{8\times 6}\times 100=\frac{20}{8\times 6}\times 100=41.7\%$$

5. 결론

오늘날 복잡한 소프트웨어들을 테스트할 경우 소프트웨어에 대한 완전한 기능 테스트는 테스트 케이스가 너무 많기 때문에 불가능하다. 따라서 불필요한 테스트 케이스를 줄이고 필요한 테스트

케이스만을 얻어낼 수 있는 방법에 관한 연구는 생산성도 높일 수 있고 노력도도 감소시킬 수가 있다.

본 논문에서 제안한 테스트 모델 및 테스트 케이스 생성기법의 의의는 다음과 같다.

첫째, 테스트 케이스 생성을 위해서는 기반이 되는 테스트 모델이 반드시 필요하다. 본 논문에서는 요구사항 분석 및 설계 단계의 산출물인 UML 시퀀스 다이어그램과 OCL 컨트랙트 명세를 이용하여 테스트 모델을 생성함으로써, 테스트 모델 생성시 요구되는 노력도를 최소화할 수 있는 테스트 모델 생성의 체계적인 프로세스를 제공하였다.

둘째, 객체지향 소프트웨어 테스트에 유용한 테스트 모델을 사용하여 불필요한 테스트 케이스를 제거할 수 있는 테스트 케이스 생성 기법을 제시하였다.

셋째, 소프트웨어에 대한 완전한 정보를 얻기가 어려운 복잡한 소프트웨어로부터 모든 종류의 입력을 제한하고, 관찰된 출력이 예상되는 출력과 일치하는지를 확인할 수 있도록 함으로써 객체지향 소프트웨어 자체가 지니고 있는 낮은 가시성을 극복할 수 있었다.

본 논문에서 제안한 객체지향 테스트 모델은 테스트 케이스 생성을 위한 프로세스의 입력으로 사용된다. 그러나 이러한 테스트 모델의 입력을 통하여 테스트 케이스와 같은 정보를 생성해 내는데 있어서 수작업은 인간의 오류나 수행 시 소모되는 시간 및 인력 낭비라는 문제 측면에서 자유롭지 못하다. 또한 객체지향 테스트 모델 생성을 위한 정보가 부정확한 경우 테스트 모델의 품질뿐만 아니라 생성되는 테스트 케이스의 품질에도 많은 영향을 미친다.

향후 연구 과제로는 본 논문에서 제안한 기법을 기반으로 테스트에 유용하게 활용할 수 있는 자동화된 객체지향 테스트 도구와 객체지향 테스트 모델 생성 시 부정확성을 줄일 수 있는 정형적인 소프트웨어 명세 방법들과 실증적 고찰에 관한 연구를 들 수 있다.

References

[1] H. -M. Noh and C. -J. Yoo, “Specification Technique of EJB - Based Application using Design by Contracts Approach”, Korea Information Science Society, vol. 29, **(2002)**, pp. 895-906.

[2] W. -S. Byoun, C. -J. Yoo and Y. -S. Kim, “Game Software Test Case Generation Technique Using Equivalence Class Partitioning and DbC Approach”, Journal of the Korean Society for Computer Game, vol. 4, no. 22, **(2010)**.

[3] Hans-Gerhard Gross, "Component-Based Software Testing with UML", Springer, **(2004)**.

[4] C. -J. Yoo and H. -M. Noh, “Test Case Generation Techniques based on Use Cases for Interoperability Test of Component-Based software”, Korea Information Science Society, vol. 36, no. 5, **(2009)**, pp. 361-375.

[5] C. -Y. Chen, "Constructing Usage-Based Testing On Object-Z Formal Specification Based Specification", Ph.D. Dissertation, Auburn University, **(1999)**.

[6] E. M. Choi, “Software Engineering: Generating Test Cases for Object - Oriented Design Specification”, Korea Information Processing Society, vol. 8-D, **(2001)**, pp. 843-852.

[7] E. M. Choi, "Use-Case Driven Test for Object-Oriented System", Proceedings of the IASTED International Conference, ACTA Press, **(2001)**, pp. 164-169.

[8] S. Ramaswamy and R. Neelakantan, "Software Design and Testing Using PetriNets: A Case Study Using a Distributed Simulation Software System", 2002 Performance Metrics for Intelligent Systems, National Institute of Standards and Technology, NIST Special Publications, **(2002)**.

[9] R. Duke, P. King, G. Rose and 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, **(1991)**.

[10] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes, "Object-Oriented Development: The Fusion Method", Prentice Hall, **(1994)**.

[11] J. Warmer and A. Kleppe, "The Object Constraint Language Second Edition – Getting Your Models Ready for MDA", Addison Wesley, **(2003)**.

[12] Object Management Group, "Model Driven Architecture – Resource Page", Technical Report, http://www.omg.org, **(2009)**.