블랙박스 테스트 케이스의 리엔지니어링

서 광 익[†] · 최 은 만[††]

요 약

소프트웨어를 블랙박스 테스트 하려면 대상 소프트웨어에 적절한 데이터를 주어 실행해 보아야 한다. 효과적인 테스트가 되기 위해서 테스트 케이스의 선택뿐만 아니라 테스트 케이스가 어떻게 표현되었는가가 중요하다. 또한 정적인 테스트 작업에도 테스트를 위한 체크리스트가 어떻게 작성되었는지에 따라 테스트 작업의 효율성이 좌우된다. 이 논문에서는 비효율적이며 문제가 있는 테스트 케이스와 체크 리스트들을 리엔지니어링 하는 방법을 제시하고 이를 실험 하였다. 임베디드 시스템의 일종인 디지털 방송수신 장치에 탑재된 소프트웨어를 대상으로 하여 이미 사용 중인 테스트 케이스의 효율성과 적합성을 따져보고 이를 리엔지니어링 하였다. 리엔지니어링 한 후의 테스트 케이스의 산출물이 테스트 시간과 커버리지 측면에서 얼마나 효과적인지를 살펴보았다. 또한 제품 계열 개념의 소프트웨어를 테스트하기에 적합하도록 테스트 케이스를 재사용 또는 재구조화 하는 방법도 연구하였다.

키워드 : 임베디드 소프트웨어, 소프트웨어 테스트, 테스트 케이스, 테스트 효율성

Reengineering Black-box Test Cases

Seo Kwang Ik[†] · Choi Eun Man[††]

ABSTRACT

Black-box testing needs to prepare fitting test data, execute software, and examine the result. If we test software effectively, not only selecting test cases but also representing test cases are important. In static testing effectiveness of testing activities also depends on how to represent test cases and checklist to validate. This paper suggests a method for finding ineffective critical test cases and reengineering them. An experiment of reengineering digital set-top box software shows the process and results of checking effectiveness and conformance of current test cases and patching test cases. The result shows how much save the test time and improve test coverage by reengineering test cases. Methods of reuse and restructuring test cases are also studied to fit into embedded product-line software.

Key Words : Embedded Software, Software Test, Test Case, Test Efficiency

1. 서 론

NIST의 조사보고에 의하면 소비자들이 구매한 소프트웨어에 포함된 오류를 50% 줄인다면 117억 달러나 절약할 수 있다고 한다[1]. 소비자들이 사용할 때 절약되는 것뿐만 아니라 개발자 입장에서도 오류를 먼저 찾아냄으로써 비용 절감의 효과를 가져 올 것이다. 이런 통계를 보더라도 현재의 테스트 기술에 더 많은 발전이 필요하며 오류를 효과적으로 찾아 이를 줄이려는 테스트 작업에 대한 노력이 중요하다.

소프트웨어 테스트 작업은 소프트웨어 프로덕트가 예상하는 것만큼 잘 작동되는지를 확신할 수 있는 좋은 방법 중의 하나이다[2]. 소프트웨어를 테스트 하려면 먼저 시험 대상 소프트웨어에 대한 요구를 잘 분석하여 어떤 테스트 데이터를 주었을 때 어떤 결과가 나와야 하는지를 찾아내야 한다. 이를 테스트 케이스라고 하는데 테스트 케이스를 어떻게 작성하는가는 테스트 작업의 효율성을 크게 좌우한다.

소프트웨어에 직접 입력을 주어 실행시키는 동적인 테스트 작업은 테스트 케이스가 얼마나 오류를 잘 발견할 수 있도록 최소화 되었는가가 중요한 관건이다. 입력 자료값을 주고 수행결과를 확인하는 동적인 테스트뿐만 아니라 소프트웨어의 외관과 실행시키기 위하여 사용자가 메뉴를 선택하고 여러 가지 정보가 전달되는 방법을 구체적으로 확인하는 작업도 매우 중요한 시험 작업이다. 이러한 측면에서 사용성(Usability)을 확인하기 위한 체크 리스트도 테스트 슈트라 볼 수 있다.

임베디드 시스템의 경우 소프트웨어와 하드웨어를 개발 과정에서 분리하여 테스트하기가 매우 어렵다. 서로 밀접히 관련되어 있어 같이 연동되는 부분이 많고 단위 시험에서 시뮬레이션 환경을 제공하여 테스트하였다 하더라도 타겟 환경에서 통합한 후 시스템 수준에서 기능과 사용성 측면의 테스트가 함께 이루어져야 컴포넌트들의 연동을 확인할 수 있다[3].

※ 이 논문은 동국대학교 논문게재 연구비의 지원으로 이루어졌음.
† 준 회 원 : 동국대학교 컴퓨터공학과 박사수료
†† 정 회 원 : 동국대학교 컴퓨터멀티미디어공학과 교수<교신저자>
논문접수 : 2006년 2월 14일, 심사완료 : 2006년 5월 31일

이론적인 테스트 기법에서는 테스트 케이스는 한번 사용하고 버리는 것으로 교육하고 받아들이는 경우가 많았다. 그러나 실제 개발 현장에서는 100% 새로운 요구가 아닌 유사한 부분이 많은 소프트웨어를 개발하고 있다. 특히 소프트웨어 제품 계열(Software Product Line) 성격의 시스템은 비슷한 구성요소들 중에 일부만을 바꾸어 새로운 소프트웨어, 또는 탑재 시스템으로 완성하는 경우가 많다[4]. 이런 경우 테스트 케이스를 모두 다시 만드는 것은 비효율적이며 지루한 작업이다.

따라서 이 논문에서는 제품 계열 개념의 소프트웨어를 테스트 할 때 이미 개발되어 테스트한 경력이 있는 요소들을 비용, 오류 추출율이라는 측면에서 과거 테스트 케이스를 개선하고 재사용하는 방법을 제안하였다. 또한 과거 테스트 케이스와 비교 시험하여 개선된 테스트 케이스가 과연 효율적인지를 보였다.

본 논문의 구성은 다음과 같다. 제2장에서는 블랙박스 테스트를 위한 좋은 테스트 케이스의 생성 기법에 대해서 알아보고 제3장에서는 테스트 케이스의 리엔지니어링 절차를 기술한다. 제4장은 테스트 케이스를 재사용하기 위한 규모를 알아보고 제5장에서는 기존 테스트 케이스와 시험 대상 시스템에 대해 분석한다. 제6장에서는 리엔지니어링을 적용한 실례를 기술했고 제7장은 적용 후의 효과를 분석했다. 그리고 마지막으로 제8장에서 결론에 대해 기술했다.

2. 블랙박스 테스트를 위한 좋은 테스트 케이스의 생성 기법

2.1 블랙박스 테스트 케이스

테스트 케이스에 관한 이론적 연구는 주로 단위 테스트 단계의 화이트박스 테스트 기법을 위주로 연구되어 왔다. 테스트 대상이 되는 원시코드의 지식을 바탕으로 실행경로와 테스트 데이터의 최적화라는 이슈가 주관심사 이었던 것이다. 화이트 박스 테스트는 프로그램 안에 확정된 실행 경로라는 정보를 기초로 이루어지므로 보다 확실하고 객관적인 테스트 케이스를 만들 수 있다. 또한 자동화도 수월하고 커버리지 모니터링도 가능하다.

반면에 기능의 잘못된 구현이나 데드코드는 발견하기 어렵고 인터페이스나 다중 프로세스, 스레드, 인터럽트 등의 동적인 부분에 대한 시험이 어렵다. 화이트 박스 테스트 기법의 결정적인 한계는 원시코드의 규모가 커지고 실행 경로가 무수히 많은 경우가 많다는 점이다. 특히 단위 테스트의 규모를 벗어난 시스템 수준의 테스트에서는 블랙박스 테스트 기법을 주로 사용한다[5].

대규모 소프트웨어 개발 현장에서 주로 사용하는 블랙박스 테스트 기법을 조사 연구한 결과 다음과 같은 사실을 알 수 있었다.

1. 블랙박스 기법은 기능분할의 기준을 어느 것으로 삼느냐에 따라 테스트 케이스가 크게 달라진다. [3]에 연구한 바와 같이 요구명세를 기준으로 기능을 분할하더라도 요구 명세가 어떤 수준이냐에 따라 테스트 케이스 자체가 상이하다.
2. 사용사례나 정형적 기법인 Object-Z를 기준으로 기능을 분할하면 테스트 하려는 소프트웨어의 기능이 상당히 성글게 쪼개지는 반면 OCL(Object Constraint Language)이나 테스트 케이스를 고려한 확장된 사용사례를 기준으로 분할하면 더 잘게 쪼개진 테스트 케이스를 구할 수 있다.
3. 보다 자세한 명세를 사용하는 것이 테스트 케이스의 커버리지를 높이는 방안이 될 수 있고 이런 경우 블랙박스 방법에서 화이트박스 내부로 파고들어 가는 셈이 된다[6].

여기서 한 가지 중요한 사실은 테스트 작업에서 사용하는 테스트 케이스의 표현 형태가 테스트 작업의 시간과 효율성에 큰 영향을 미치는 것으로 발견되었다. 이 연구에서 조사 참여한 임베디드 소프트웨어의 시험 작업 과정에는 다음과 같은 여러 종류의 테스트 케이스 표현법과 테스트 케이스 단위를 고려하고 있었다.

- 기능위주로 테스트 케이스를 분류하되 시나리오를 구성하여 체크 리스트 형태로 만드는 방법
- 보고된 오류 케이스를 체크하기 위하여 시나리오를 작성하는 방법(흐름도로 작성하기도 함)[7].

블랙박스 테스트하기 위하여 각 기능을 체크 리스트 형태로 나열한 테스트 슈트는 모듈화가 가능하므로 임베디드 시스템이 프로덕트라인을 이루어 각 기능이 가변적일 때 테스트에 사용하기 좋다. 반면 테스트 슈트가 장황하게 되어 만들기가 번거롭고 중복이 많을 수 있다. 특정 오류를 발견하기 위하여 에러 케이스를 정의하고 이를 확인하기 위한 구동 절차를 흐름도로 작성하는 방법은 표현이 축약적이나 기능을 모두 커버하지는 않으며 재사용을 위한 모듈화가 어렵다.

두 가지 방법이 보완적이며 임베디드 시스템의 블랙박스 테스트를 위하여 사용되고 있는 방법이나 이 논문에서는 테스트 케이스의 재사용 및 리엔지니어링 차원의 작업에 초점을 두었으므로 모듈화된 테스트 케이스를 만들기 위하여 전자의 방법이 더 적합한 것으로 판단하여 이를 채택하였다.

2.2 좋은 테스트 케이스

IEEE에서 발행한 소프트웨어 관련 용어집[8]에 의하면 테스트 케이스는 "특정 프로그램의 경로나 소프트웨어의 요구사항을 확인 등 일정한 목적을 위하여 개발된 테스트 입력, 실행 조건, 예상 결과의 집합체"라고 정의하고 있다. 그러나 실제 소프트웨어를 세밀하고 체계적으로 테스트하려면 입력과 예상 결과 이외에도 테스트가 이루어지기 전까지의 상태와 환경, 테스트 케이스가 커버하는 기능이나 품질 요소도 중요한 요소이다.

〈표 1〉 좋은 테스트 케이스가 되기 위한 속성

오류추출성 (Readable)	테스트 목적에 맞는 오류를 잘 드러내야 한다.
커버리지 (Coverage)	테스트 하려는 타겟이 테스트 케이스에 의하여 전부 확인될 수 있어야 한다.
경제성 (Minimize cost)	테스트 작업을 실행하는 데 소요되는 노력의 최소화.
수정가능성 (Up-to-date)	테스트 슈트가 수정 가능해야 한다.
구조성 (Structural)	재사용을 위하여 수직적으로 분해하고 합성할 수 있도록 잘 체계화 되어야 한다.
문서화 (Documented)	테스트 케이스가 작업에 드움이 되도록 잘 설명되어 있고 정리되어야 한다.

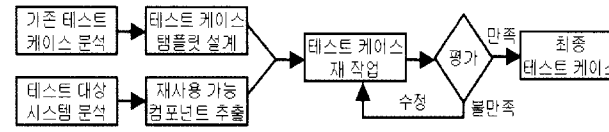
일반적으로 좋은 테스트 케이스란 테스트 목적에 맞는 오류를 잘 드러내고, 최소한의 비용과 노력으로 원하는 테스트를 커버하게 하여야 한다. 즉 오류추출율과 커버리지가 좋으면 테스트 케이스는 좋은 것이라고 이론적으로 알려져 있다. 그러나 실무 환경에서 실제 테스트 케이스를 작성해 보면 이외에도 다른 여러 가지 속성들을 만족하여야 한다. 특히 테스트 케이스를 실행시키기 위하여 필요한 조건이나 환경뿐만 아니라 테스트 케이스가 어떻게 표현되었는가도 테스트 작업의 성과를 좌우하는 중요한 측면이다.

테스트 케이스는 일종의 소프트웨어의 기능이나 성능에 대하여 던지는 일종의 질문이다[9]. 따라서 각 테스트 케이스는 테스트 엔지니어가 잘 이해하고 작업할 수 있도록 정리되어야 한다. 또한 임베디드 시스템과 같이 제품 계열 개념에 의하여 확장되거나 조금 다른 프로덕트를 테스트 할 때 쉽게 재구성할 수 있도록 구조화 되어야 한다.

특히 임베디드 시스템의 경우 제품계열 안에 포함되는 소프트웨어의 버전이 다르지만 제품 계열의 라이프사이클 동안 계속 지원되어야 하며 발전 재사용되어야 한다. 따라서 좋은 테스트 케이스는 수직적으로 잘 분할하고 합성할 수 있도록 모듈화 되어야 하며 테스트 작업의 기간을 단축할 수 있도록 대표성이 최적화되어야 한다.

3. 테스트 케이스 리엔지니어링 절차

소프트웨어의 리엔지니어링을 위해서는 이미 존재하는 도구나 시스템 그리고 문서와 같은 정보를 먼저 분석해야 한다[10, 11]. 이와 마찬가지로 테스트 케이스의 리엔지니어링 작업을 위해서 테스트 케이스와 테스트 대상 시스템에 대한 이해 작업이 필요하다. 먼저 테스트 케이스를 재구성하기 위해서 테스트 케이스를 이루고 있는 구성 요소와 정보에 대한 분석이 필요하다. 테스트 케이스가 필요한 정보를 모두 포함하고 있는지 아니면 불필요한 정보로 인해 작업의 효율성을 떨어뜨리고 있는지 분석한다. 그리고 테스트를 수행하는 실무자나 조직의 관리자가 원하는 정보를 제공하고 있는지에 대한 평가도 필요하다. 다음으로는 테스트 대상 시스템을 이해해야 한다. 해당 시스템에 대한 기능의 이해와 수행 절차 그리고 기능들 간의 구조와 시스템을 설치하는 방법 등 시스템에 대해 완전한 이해와 파악이 필요하다. 이러한 이해 작업이 있어야만 적절하게 재사용이 가능한 테스트 케이스를 추출할 수 있다. 이해 작업이 끝나면 기존 테스트 케이스를 분석한 결과와 테스트 대상 시스템의 분석 결과를 바탕으로 테스트 케이스의 서식을 설계한다. 서식의 설계는 테스트 케이스가 지녀야 할 정보와 시험 작업의 효율성을 결정하기 때문에 중요한 업무라 할 수 있다[17]. 그리고 새로 작성된 테스트 케이스의 내용과 서식은 실무자들에 의해 평가되고 만족을 충족시킬 때까지 계속해서 수정한다. 이러한 결과를 통해 최종 테스트 케이스가 생성된다.



(그림 1) 테스트 케이스 리엔지니어링 절차

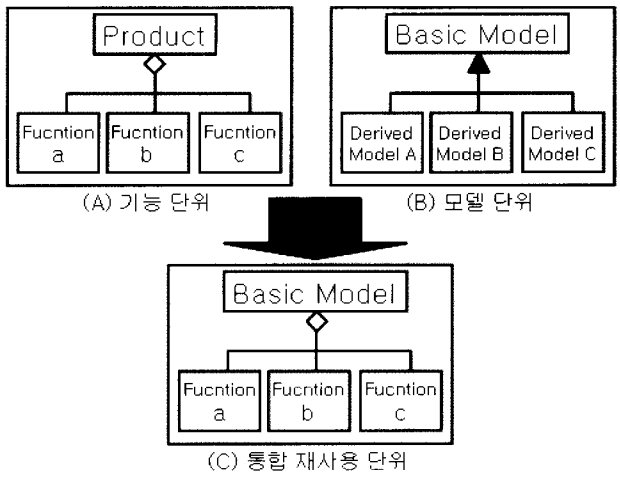
(그림 1)은 이러한 작업 과정을 보이고 있다.

4. 테스트 케이스의 재사용 단위

제품을 개발할 때마다 테스트 케이스를 설계하고 작성하는 것은 비효율적인 일이다. 제품 계열(Product Line)의 성격을 가진 제품을 개발할 경우에 소프트웨어 뿐 만 아니라 테스트 데이터나 테스트 케이스도 재사용 할 수 있다[12]. 그 중 테스트 케이스를 재사용하기 위해서는 테스트 케이스의 재사용 단위를 고려해야 한다.

테스트 케이스를 재사용 할 수 있는 단위는 두 가지로 나누어 볼 수 있다. 첫 번째는 특정 기능 단위로 재사용 하는 것이다[13]. 예를 들면 시스템의 기본 설정을 변경하거나 중요한 속성을 변경할 경우 사용자 검증을 위한 비밀번호 입력과 같은 일부 기능에 대한 테스트 케이스가 재사용의 단위가 될 수 있다. 그리고 기능보다 더 큰 단위로써의 특정 모델 또는 제품에 대한 테스트 케이스가 재사용의 단위가 될 수 있다. 예를 들면 특정 모델을 기반으로 일부 기능들을 추가하여 후속 모델이나 제품들이 개발하는 경우가 있다[14, 15]. 실제로 새로운 제품을 개발할 때는 이미 개발된 기능을 조합하거나 기존의 개발된 모델에 일부 기능을 변경하거나 추가 또는 삭제하여 개발할 때가 많다. 이러한 이유로 인해 본 연구에서는 (그림 2)와 같이 두 가지 경우를 모두 고려하여 테스트 케이스를 설계하였다.

(그림 2)의 (A)는 기능들을 조립하여 제품을 개발하는 경우 재사용의 단위로써 각 기능에 대한 테스트 케이스를 표현하고 있다. 서로 다른 독립적인 기능들을 조립하여 하나의 제품을 구성하기 때문에 조립된 각 기능들에 대한 테스트 케이스를 재사용할 수 있다. 여기서 어떤 기능이 다른 제품에서도 재사용 된다면 마찬가지로 해당 기능에 대한 테스트 케이스도 재사용 될 수 있다. 그리고 (그림 2)의 (B)는 각 모델의 기본이 되는 제품을 표현하고 있다. 모델 A, B,



(그림 2) 테스트 케이스 재사용 단위

C는 기본이 되는 모델(Basic Model)에서 파생되었기 때문에 파생된 모델 A, B, C의 기능이나 구조는 기본 모델의 기능과 구조를 가지고 있다. 따라서 기본 모델에 대한 테스트 케이스는 파생 모델 A, B, C의 시험에 모두 응용할 수 있다. 그러나 기본 모델이라 하더라도 실제 제품을 개발할 때에는 다양한 변수가 있어 기본 모델의 특정 부분을 변경하거나 추가 또는 삭제하여 파생 모델을 만들게 된다. 따라서 기본 모델에 대한 전체 테스트 케이스를 그대로 재사용하기 어렵다. (그림 2)의 (C)는 재사용이 가능한 단위로 기능과 기본 모델을 모두 표현하고 있다. 즉 기본 모델을 구성하고 있는 기능 중 필수적인 기능만을 재사용의 단위로 이용한다. 왜냐하면 기본 모델의 특정 기능은 파생 모델에 따라 추가되거나 삭제될 수 있다. 따라서 본 연구에서 재사용을 위한 단위의 의미는 기본 모델에 속하면서 변경이나 삭제가 없는 필수 기능에 대한 테스트 케이스로 삼는다[16, 17].

5. 테스트 케이스 및 시험 시스템 분석

5.1 기존 테스트 케이스의 문제점

협력 업체에서 재사용되고 있는 테스트 케이스는 여러 제품에서 기본적으로 갖추고 있어야하는 기능들에 대한 공통 시험 항목들을 포함하고 있어야 한다. 하지만 실제 시험을 할 때 기존 테스트 케이스를 사용하는 데 있어서 다양한 문제점들이 드러나고 있다. 테스트 대상 시스템을 시험할 때 사용하는 기존의 테스트 케이스의 명세 방법과 내용에 대하여 발견한 문제점 몇 가지는 다음과 같다.

첫 번째로 테스트 케이스는 기본적인 기능을 충분히 반영하고 있지 못하고 있다. 기능을 시험하는 테스트 케이스의 명세가 추상적이거나 요약되어 있고 어떤 기능들은 누락된 것들도 있었다. 하지만 이러한 부분들이 테스트 케이스에 반영되지 못하고 있어서 정확한 테스트를 수행했는지에 대한 신뢰성이 떨어지게 된다.

두 번째로 서로 다른 테스트 요소들이 있음에도 이들이 하나의 체크리스트에 혼재되어 있다. 그 예로 하나의 테스트 체크리스트에 기능 테스트와 UI 테스트 등이 중복되어 있다. 이러한 경우 하나의 체크리스트를 통해 기능 테스트와 UI 테스트를 함께 할 수 있다는 장점이 있지만 만약 오류가 발견 된다면 오류를 기록할 때 시험 항목의 내용과 발견된 오류의 매핑이 어려워지게 된다. 또한 이러한 문제는 테스트 케이스를 재사용하거나 재구성이 필요할 경우 수정을 어렵게 한다. 어떤 모델에서 특정 기능이 삭제된다면 테스트 케이스에서도 해당 기능에 대한 부분을 삭제해야 한다. 기능 테스트와 UI 테스트가 분리되어 있을 경우는 해당 기능에 대한 체크리스트만 삭제하면 되지만 기능 테스트와 UI 테스트가 함께 정의된 경우에는 해당 테스트 케이스를 찾아 검토하고 수정해야만 한다.

세 번째로 작성된 테스트 케이스는 오랜 시간 동안 여러 사람이 테스트 케이스를 기술하고 확장하였다. 따라서 테스트 케이스를 작성하기 전에 명칭이나 기능에 대한 용어를 통일하기 위한 정의가 필요했다. 하지만 이러한 작업을 생략함으로 인해 동일한 명칭이나 기능임에도 불구하고 테스트 케이스에서 쓰이는 용어가 서로 다르게 사용되고 있다. 예를 들면 문자를 입력하는 작업 창을 'Text Edit 창' 또는 '키보드 창' 또는 'Edit 창' 등으로 불린다. 이는 테스트 케이스에 사용된 용어의 일관성을 결여시키기 때문에 테스트 엔지니어에게 혼란을 초래할 수 있다.

네 번째로 테스트 케이스는 시험 내용과 그에 대한 예상 결과를 함께 기술해야 한다. (그림 3)은 현재 실무에서 사용하는 테스트 케이스의 일부이다. (그림 3)에서 보는 바와 같이 예상된 테스트 결과를 정의하는 곳이 없다. 따라서 테스트 결과 실패와 성공의 정확한 기준이 테스트 엔지니어마다 다를 수 있다.

다섯 번째로 테스트 케이스가 기능 위주로 기술되어 있고 그 기능에 접근하는 절차나 과정에 대한 시험 내용이 없다. 특정 기능이 오류 없이 잘 동작한다 하더라도 그 기능을 접근하는 과정에 오류가 발생할 수 있다. 그리고 특정 기능을 접근하기까지는 다양한 경로가 있을 수 있고 그 중에 특정 경로에는 오류가 잠재되어 있을 수 있다. 따라서 이러한 경로에 대한 시험이 필요하다.

이 밖에도 다양한 부가 정보나 발견된 에러에 대한 관리를 지원하는 정보가 부족하다. 예를 들면 오류의 심각성이나 테스트 수행 날짜나 테스트 엔지니어에 대한 기록과 같은 사항에 대한 내용이 부족하다. (그림 3)의 테스트 케이스에는 관련 기능과 식별 번호 그리고 그 결과를 적게 되어 있으나 테스트 데이터나 환경, 전제 조건 등에 대한 내용은 찾아 볼 수 없다. 이러한 문제점을 보완하기 위해 본 연구에서는 테스트 케이스 서식을 새롭게 설계하고 테스트 작업 내용을 작성하는 방법에 대한 실례를 보였다.

Item / No	B.1.4 Installation		Result
	Test Segment	Test Description	
B.1.4.4	Manual Search	1. Transponder에서 User Define 선택 시 Edit Transponder Menu로 이동되어야 한다.	
		2. Transponder에서 User Define 선택 시 Edit Transponder Menu로 이동 Frequency 항목에 커서 이동 Arrow Right/OK Key 누르면 입력 창 출력되어야 하며, 숫자 키 입력과 동시에 다음 자리에 커서 입력이 되어야 한다.	

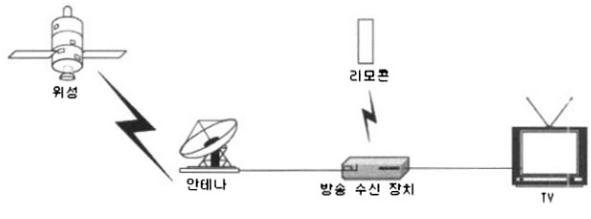
(그림 3) 기존 테스트 케이스

5.2 테스트 대상 시스템 분석

재사용 가능한 테스트 케이스 설계를 위해서는 먼저 기본 모델에 대한 기능 구조를 완전히 파악하는 것이 우선 되어야 한다[13]. (그림 4)는 현재 다양한 모델의 기본 모델이 되고 있는 시스템의 전체 구조이다. 기술의 발전과 사용자의 요구사항의 다양화로 인해 단순한 기능의 수신 장치로부터 복잡한 기능의 기술이 집적된 수신 장치로 발전하고 있어 다양한 파생 모델이 있다. 하지만 산학 협력 업체의 파생 모델에서 기본적으로 제공해야 하는 공통 기능은 (그림 4)와 같다. 이러한 기본적인 공통 기능을 제공하고 있는 모델을 기본 모델로 선정하고 이 모델에서 제공하는 필수적인 기능들에 관한 테스트 케이스를 재사용의 단위로 설정한다. 이와 같이 기본 모델의 필수 기능에 대한 테스트 케이스를 작성한다면 이를 이용하여 파생된 다양한 모델들에 적용 재사용이 가능할 것이다.

기본 모델의 기능 구조를 살펴보면 먼저 수신 장치에 탑재되어 있는 소프트웨어는 크게 일반과 메뉴 부분으로 나누어진다. 먼저 일반 부분은 다시 채널 탐색과 일정/예약으로 나뉜다. 채널 탐색은 (그림 4)에 나타나 있듯이 채널 리스트와 채널 잠금 등과 같은 기능을 제어하는 것들의 집합이며, 일정/예약은 프로그램 편성을 지원하기 위한 기능들의 집합이다. 메뉴 부분은 속성 설정, 채널 편집, 시스템 초기화, 장치 정보로 나뉘며 각각의 세부 기능들의 일부를 (그림 4)에 간단히 정리했다. 속성 설정은 보호자 설정과 언어 설정 등과 같은 기능들에 대한 집합이며, 채널 편집은 채널에 대한 편집 즉, 제어 및 설정을 변경할 수 있는 기능들의 집합이다. 시스템 초기화는 메뉴 부분에서 가장 중요하게 시험해야 할 부분이라 할 수 있다. 이는 안테나 설정, 위성에서 제공하는 채널 검색, 소프트웨어의 업데이트 및 초기화를 제어하는 기능들의 집합이다. 장치 정보는 설치된 소프트웨어의 정보 및 위성에 따른 시스템의 상태를 확인하는 기능들의 집합이다.

이와 같이 리엔지니어링을 하기 위해서는 이미 존재하고 있는 문서와 시스템에 대한 정보에 대한 분석이 필요하다. 그리고 이러한 분석을 바탕으로 테스트 케이스의 리엔지니어링과 재사용을 위한 기능들의 구체적인 레벨의 깊이가 결정된다. 여기서 레벨이란 피라미드 형태로 이루어진 기능의 상위수준부터 하위수준까지의 수평적인 그룹 계층을 의미한다. 어느 수준까지 테스트 케이스로 기술해야 하는지를 결정하는 것 또한 테스트 케이스의 재사용 단위로 결정하는데 중요한 업무라 할 수 있다.

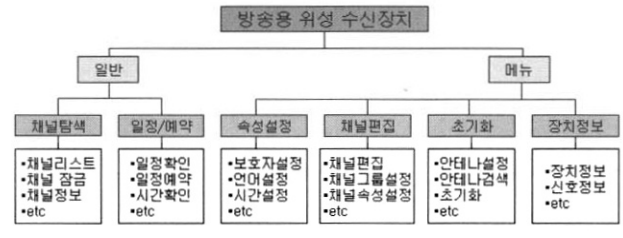


(그림 4) 기본 모델의 전체 기능 구조

5.3 시험 환경

실험 시스템은 디지털 방송 수신 장치로써 안테나로 수신한 디지털 방송 전파를 TV로 전송하기 위해 아날로그 신호로 변환해 주는 장치이다. 이러한 장치는 TV 내장용과 외장용으로 크게 나뉘는데 본 연구에서는 외장용을 사용했다.

(그림 5)는 시험 환경을 보이고 있다. 방송 신호 수신 장치는 안테나를 이용하여 위성으로부터 디지털 신호를 받아 TV에 아날로그 신호를 전송한다. 방송 수신 장치는 리모콘을 이용해 조작된다. 따라서 본 연구에서도 리모콘을 이용해 타겟 시스템의 기능을 시험했다. 타겟 시스템의 정확한 제품 모델명과 소프트웨어에 관한 정보는 산학협력 연구에 참여한 산업체의 보안 및 대외비 사정으로 인해 생략하기로 한다. 다만, 이미 개발이 완료되어 다수의 사용자가 본 제품 또는 유사한 파생 제품을 사용하고 있음을 밝힌다.



(그림 5) 시험 환경

6. 리엔지니어링 적용 사례

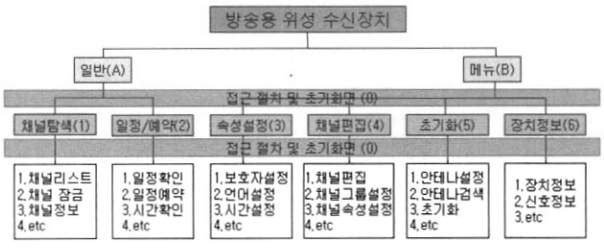
새로 작성한 테스트 케이스는 5.1절에서 언급했던 문제점들을 모두 해결하도록 수정하였다. 먼저 재사용 단위가 가능한 기본 모델의 모든 기능들에 대해 테스트 케이스를 작성하였다. 또한 하나의 체크 리스트는 최소 기능을 설명하는 목적어 한 개와 서술어 한 개를 기본으로 하였고 UI 시험의 경우는 한 개의 아이콘이나 한 개의 아이템을 체크리스트의 단위로 정의했다. 각 체크리스트가 기능 시험인지 또는 UI 시험인지 각 특성을 분류할 수 있도록 구별하였다. 또한 테스트 케이스에서 사용되는 용어들을 통일하여 정리하였고 다양한 부가 정보들을 포함 할 수 있도록 테스트 케이스 서식을 설계하였다. 기능의 동작과 그 기능에 접근하는 과정 및 초기 상태에 대한 시험을 구별하여 시험 범위를 확장하였다. 이 외에도 테스트 케이스의 고유 번호를 통해 테스트 케이스의 시험 내용을 직관적으로 식별할 수 있도록 구성했다.

6.1 테스트 케이스 작성 실례

이 절에서는 협력 업체의 시스템을 대상으로 테스트 케이스 리엔지니어링 하는 과정에서 테스트 케이스를 작성하는 실례를 기술적으로 보안이 필요한 부분을 제외하고 정리하였다.

6.1.1 테스트 케이스 식별 번호 작성 방법

테스트 케이스의 식별 번호는 현재 사용하고 있는 시험의



(그림 6) 테스트 케이스 식별 번호 체계

내용과 진행 과정에 대한 정보를 제공하기 때문에 아주 중요한 작업이라 할 수 있다. 테스트 케이스의 식별 번호는 상위 기능부터 하위 기능의 순서에 따라 각각의 레벨과 기능을 구분할 수 있도록 식별 번호를 부여했으며 각 기능의 초기 화면이나 접근 절차에 대한 시험이 필요한 경우 번호 '0'을 사용하여 이를 구별했다. (그림 6)은 5.2절에서 파악한 시스템의 기능 구조에 따라 주어진 테스트 케이스의 식별 번호를 보이고 있다.

a. 초기상태(Test NO. '0')는 아래의 세 가지에 해당된다.
   - 접근절차 테스트 케이스(Test NO. '0-1'): 해당 기능의 접근 절차에 대한 테스트 케이스이다.
   - UI 테스트 케이스(Test NO. '0-2'): 해당 기능의 초기화면인 UI에 대한 테스트 케이스이다.
   - 테스트 케이스(Test NO. '0-3'): 화면에서 해당 기능을 선택하기 위한 방향키 조작에 대한 테스트 케이스이다.

b. Toggle(버튼을 눌렀을 경우 화면에 팝업 형태의 정보가 나타났다가 동일한 버튼을 다시 눌렀을 때 사라지는 기능)과 같은 형태의 기능은 초기 상태에 포함한다.
   - 예를 들어 메뉴에서 Left Key 또는 Back Key를 사용할 경우 화면의 작업 창이 사라지는 것과 같은 기능은 초기 상태의 절차('0-1') 또는 초기 상태('0')에 작성한다.

c. 절차 중 비밀번호를 확인하는 기능이 있는 경우 접근절차 테스트 케이스(Test NO. '0-1')에 자세히 정리한다.

6.1.2 시험명의 작성 방법

시험명은 Test-case NO.를 참고로 하여 (그림 7)과 같이 테스트 케이스의 식별 번호와 일치하도록 작성한다. 이러한 방법은 현재 작업하고 있는 시험에 대한 위치와 정보를 쉽게 확인할 수 있도록 도와준다. 테스트 케이스의 식별 번호 분류 체계와 시험명의 분류 체계가 동일하다.

6.1.3 시험사례 설명 작성 방법

a. (그림 7)과 같이 시험사례 설명은 시험명의 마지막 단위를 중심으로 작성한다.

Test-model		Test-case NO.	A-2-5-1-1
S/W Version		H/W Version	
Load Version		System ID	
시험명	General - Guide - Schedule - 예약 추가 - 예약 정보 입력		
시험사례 설명	Schedule 화면에서 예약을 추가하는 기능을 확인한다.		

(그림 7) 시험명 및 시험사례 설명

b. 서술어는 '확인한다.'로 통일한다.

6.1.4 시험 절차 작성 방법

a. 시험 절차는 초기상태(ex. A/V 상태)에서부터 시험이 이루어지는 전 과정을 작성한다.

b. 절차의 선후 관계는 (그림 8)과 같이 '->'(하이픈 + Right Angle Bracket)으로 표시한다.

시험 절차	A/V 상태 -> Menu Key 사용 -> 'Installation 항목 선택 -> Password 입력 -> Installation Menu로 이동 -> '4. Other Search'선택 -> '1. Advanced Search' 선택

(그림 8) 시험 절차

6.1.5 시험내용 작성방법

a. 시험내용을 기술하는 명세의 서술어는 의문형을 사용한다.
   - 출력되는가? 표시되는가? 동작되는가? 선택되는가? 갱신되는가? 분류되는가? 등.

b. (그림 9)와 같이 한 문장에 2개 이상의 기능이 결합되어 있는 경우 각각 분리한다.
   - 예를 들어 '이름으로 찾기의 기본 동작: Blue Key를 누르면 Find 초기화면이 출력되고 다시 Blue Key를 누르면 일정표 화면으로 돌아가야 한다.'와 같은 기능이 있을 경우 2 개의 기능으로 분리한다.

c. 동일한 입력으로 2개 이상의 결과가 기대되는 테스트의 경우 각각의 결과마다 다른 시험 항목으로 작성해야 한다.
   - 예를 들어 (그림 10)과 같이 탐색 실행 중 임의로 멈출 경우 탐색된 결과가 있으면 현재까지 진행된 결과만을 보여주고, 탐색된 결과가 없으면 'No channel..'이라는 내용의 정보 창이 출력 되어야 한다.

No.	시험 내용	기대 결과
1	Guide 상태에서 사용자가 원하는 프로그램을 찾기 위한 Find 대한 절차가 올바르게 작동하는가?	Find Programme by Name'에 대한 Text Edit 창이 표시된다.
2	Find 화면에서 Blud Key를 누르면 다시 Guide Tabel 화면으로 돌아가는가?	Guide 기능으로 돌아간다.

(그림 9) 분리된 테스트 케이스

탐색 중 임의로 멈춘 경우 탐색된 결과가 있는 경우 현재까지 진행된 결과가 출력되는가?	임의로 멈춘 시점까지의 결과가 출력된다
탐색 중 임의로 멈춘 경우 탐색된 결과가 없는 경우 'No Channel..'이라는 내용의 정보 창이 출력되는가?	No Channel.. 이라는 내용의 정보 창이 출력된다

(그림 10) 동일한 입력에 복수의 결과가 기대되는 경우

6.1.6 기대결과 작성방법

올바르게 나타나야 하는 기대결과를 작성한다.

6.1.7 시험 데이터 기술 형식의 통일

a. 시험 데이터 작성 방법
   - (그림 11)과 같이 숫자를 입력하는 시험 데이터는 경계 값 시험을 기준으로 하며 각 경계 값에 대해 정상 값과 비정상 값을 정의한다.

25	비밀번호을 입력한 후 예약 작업이 완료되는가? - Test Data +경계값:0000 => 정상값:0000, 0001, 비정상값: 없음 +경계값:9999 => 정상값:9998, 9999, 비정상값: 10000

(그림 11) 숫자 입력 테스트 데이터

6	끝나는 시간이 정상적으로 수정되는가? + Test Data - 정상값 : 1/1(~12), 2/1(~12), 29/4(6,9,11), 30/1(~12, 단 2월은 예외), 31/1(3,5,7,8,10,12), 27/2, 28/2 - 비정상값 : 0/1(~12), 31/4(6,9,11), 32/1(3,5,7,8,10,12)	정상값일 경우 : 정상적으로 숫자가 수정되고 다음항목으로 이동한다. 비정상값일 경우: 입력 오류 메시지가 출력된다.

(그림 12) 날짜 관련 시험 데이터

b. 날짜 관련 시험 데이터 작성 방법

- (그림 12)와 같이 일/월의 형태로 유효한 일(정상값)과 유효하지 않은 일(비정상 값)을 구분하여 시험한다.
- 1/1(~12)의 의미는 1일/1월(~12월) 즉 1월 1일, 2월 1일, 3월 1일, ... 12월 1일까지 테스트해야 함을 의미한다.
- 윤년의 경우 2월 29일까지 존재하기 때문에 2월 1, 2일과 28, 29일을 정상 값으로 2월 30일을 비정상 값으로 한다.

6.1.8 중복되는 기능 시험의 처리 방법

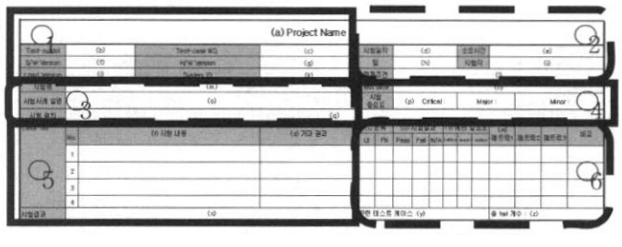
시험 대상 기능에 접근하는 동안 반복 되어 출력되는 창(비밀번호 창, 텍스트 편집 창 등)에 대한 시험은 최초의 테스트 케이스를 제외하고 나머지는 음영으로 처리하여 구분한다.

6.2 테스트 케이스 문서 서식

(그림 13)은 새로 작성한 테스트 케이스의 항목들을 간략히 보인 것이다. 새로 제안한 테스트 케이스는 표 작성의 용이성과 향후 테스트 케이스 재사용 자동화를 위해 DBMS와 연동이 가능한 스프레드시트 응용 프로그램을 사용하였다.

기존의 테스트 케이스의 일반적인 형식은 (그림 3)과 같이 5개 정도의 정보만을 기록할 수 있는 간단한 형태였다. 그러나 새로 제안한 테스트 케이스 서식은 26개의 정보를 기록할 수 있도록 해 테스트 케이스가 보유할 수 있는 정보량을 대폭 늘렸다.

테스트 케이스의 각 부분 중에 ①번에 해당하는 내용은 '시험 대상 시스템 정보'로써 수행하고 있는 프로젝트의 이름과 테스트 대상의 모델명, 테스트 케이스의 고유 식별번호, 수신 장치의 소프트웨어 버전, 수신 장치의 하드웨어 버전, 로드 소프트웨어의 버전, 수신 장치의 시스템 ID를 기술한다. 그리고 ②번 내용은 '시험자 정보'로써 시험일자, 테스트 케이스를 실행하면서 걸린 시간, 테스트 엔지니어가 소속된 팀의 이름, 테스트를 수행하는 테스트 엔지니어의 이름, 테스트를 위한 환경 및 전제 조건을 기술한다. ③번은 '시험 내용 정보'로써 테스트 케이스의 내용이 식별 가능한 시험명과 해당 테스트 케이스를 시험을 하기 위해 접근하는 절차를 기술한다. ④번은 '시험 조건 정보'로써 시험 환경과 전제조건 그리고 시험 데이터와 테스트 케이스의 중요도에 대해 기술한다. 중요도는 'Critical', 'Major', 'Minor'로 나누어 구별한다. ⑤번 내용은 '시험 내용'으로써 구체적인 시험 항목인 체크리스트와 그에 대한 기대 결과를 기술한다. ⑥번은 '시험 종류 및 결과 정보'로써 각 체크리스트가 시험하는 종류가 어떤 것인지 구별할 수 있도록 했으며 시험 결과와 중요도를 평가할 수 있도록 하였다.



(그림 13) 리엔지니어링 한 테스트 케이스 서식

체크리스트의 시험 종류를 사용자 인터페이스와 관련된 'UI' 테스트 그리고 기능과 관련된 'FN' 테스트로 구별하여 체크한다. 테스트 후 시험 결과에 대해 체크한다. 시험 결과로써 'Pass'는 성공적으로 기대결과를 얻은 경우이고 'Fail'은 기대결과와 시험 결과가 다른 경우이고 'N/A'는 알 수 없는 경우이다. 에러 중요도는 발생된 에러의 심각도에 대해 체크한다. 'critical'은 제품의 기본 기능 동작이 불가능하고 이후 아무런 동작을 하지 않거나 Stand by/Operation으로도 복구가 불가능하여 강제종료(Power off/on)를 시켜야 할 상황이 발생되는 경우이고, 'major'는 해당 기능이 정상적으로 수행되지 않거나 수행여부를 확인할 수 없는 경우 또는 수행 후에도 데이터가 불일치하는 경우이다. 'minor'는 제품이 제공하는 기능 수행에는 문제가 없으나 사용자에게 친밀하지 않거나 통상적인 개념에 어긋나는 경우이다.

⑦번은 '품질 정보'로써 ISO/IEC 9126에서 정의한 품질 요소와 관계있는 메트릭을 기술함으로써 시험이 종료된 이후 품질을 평가할 때 사용할 수 있는 정보를 기록하도록 하였다.

이 밖에도 시험 결과에 대한 특이 사항이나 참고사항을 기술하고 해당 테스트 케이스와 관련이 있는 다른 테스트 케이스를 연계하여 살펴볼 필요가 있을 경우 관련 테스트 케이스를 기술할 수 있도록 했다. 하나의 테스트 케이스에 해당하는 체크리스트의 실패 개수도 기록하여 전체 에러율을 정량적으로 나타낼 수 있도록 했다. 또한 비밀번호 인증이나 경고문을 알리는 팝업창과 같이 중복되어 사용되는 기능은 모두 음영처리를 통해 재사용되고 있는 테스트 케이스임을 표시하였다.

이와 같이 시험 대상 시스템에 대한 정보와 시험자 정보 그리고 시험 내용에 관한 정보와 시험 조건에 관한 정보, 마지막으로 시험 결과에 대한 정보를 구체적으로 기술함으로써 시행된 테스트 케이스에 대한 다양한 정보와 구체적인 테스트 결과를 보존할 수 있도록 설계하였다.

7. 리엔지니어링 효과 분석

기존의 테스트 케이스를 리엔지니어링하는 목적은 단기간

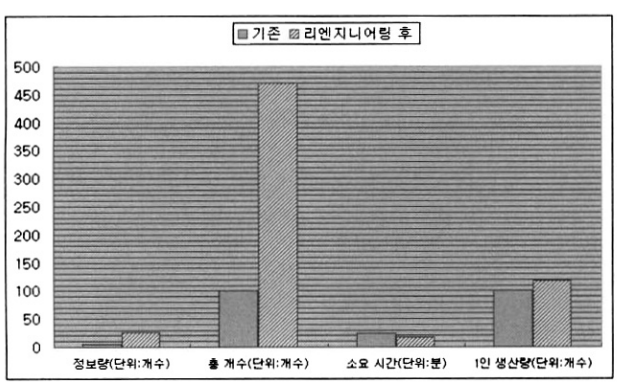
에 적은 인원으로 기존의 테스트 케이스보다 더 많은 정보량을 보유하고 있는 테스트 케이스를 작성하고 이러한 테스트 케이스를 재사용함으로써 테스트 케이스 개발에 대한 생산성을 높이는 것이다[13]. 이 뿐만 아니라 재개발된 테스트 케이스는 실제 테스트를 수행하는 테스트 엔지니어가 이해하기 쉽게 작성되어야 하고 동시에 테스트 시간을 줄일 수 있어야 한다. 이러한 이유로 본 절에서는 테스트 케이스의 리엔지니어링 결과 얻어진 생산성과 테스트 수행 시간에 관한 효율성에 대해 분석했다.

7.1 테스트 케이스 리엔지니어링 생산성

제안된 테스트 케이스는 기존의 테스트 케이스에 비해 더 많은 정보의 기록이 가능해졌다. (그림 13)은 기존의 테스트 케이스를 리엔지니어링한 결과에 대한 내용을 그래프로 정리하였다.

테스트 케이스에 기록할 수 있는 필드의 종류만 비교했을 경우 기존에 사용하던 테스트 케이스의 정보량은 (그림 3)에서 보는 바와 같이 테스트 케이스 번호, 이름, 테스트 내용, 결과 그리고 테스트 데이터로 모두 5개이다. 하지만 본 연구에서 제안한 정보 필드는 모두 26개로 기존 테스트 케이스에 대비하여 5배 정도 증가했다.

한편 기존의 테스트 케이스의 개수는 모두 100개이다. 하지만 개선한 테스트 케이스는 471개로 확장되어 기능의 초기 상태와 접근 절차의 시험이 가능해졌고 테스트 종류를 구분할 수 있도록 하였다. 기존의 테스트 케이스의 개발 기간 및 생산량은 40시간(1名*8시간*5일) 동안 100개의 테스트 케이스를 생산했다. 따라서 테스트 케이스 1개를 작성하기 위해 0.4시간(24분)이 소요되었다. 본 연구에서 진행한 테스트 케이스는 160시간(4名*8시간*5일) 동안 작업을 수행하여 471개를 생산했다. 따라서 테스트 케이스 1개를 작성하기 위해 소요된 시간은 약 0.3시간(18분) 정도가 소요되어 6분 정도가 절약되었다. 그리고 1인당 테스트 케이스를 개발한 생산량은 평균 118개 정도로 기존의 테스트 케이스 개발 생산량보다 약 18개 정도 더 많이 개발했다. 471개의 테스트 케이스를 만들기 위해 기존 테스트 스 개발 소요 시간을 적용한다면 약 188.4시간(24분*741개)이 필요하다. 하지만 160시간 안에 테스트 케이스의 작성을 끝낼 수 있었다.



(그림 14) 테스트 케이스 리엔지니어링 생산성

〈표 2〉 재사용 항목

항 목	개 수
일반-비밀번호	40
알림-비밀번호	25
Group	62
예약	7
텍스트 입력 창	12
세부 정보	2
총 계	**148**

왜냐하면 새로 작성된 테스트 케이스는 재사용이 가능하기 때문에 총 471개 중 반복되는 비밀번호 검증이나 예약과 같은 기능에 대한 테스트 케이스는 기본 모델의 테스트 케이스 내에서도 재사용이 되었기 때문이다.

<표 2>는 리엔지니어링 된 테스트 케이스 중 재사용 된 항목을 나타내고 있다. 재사용된 테스트 케이스는 '일반-비밀번호' 인증이 40개, '알림-비밀번호' 인증이 25개, Group 기능이 62개, 예약기능이 7, 텍스트 입력 창이 12개, 세부 정보가 2개로써 모두 148개이다. 그 결과 하나의 모델에 대해 테스트 케이스를 작성하는 과정에서도 약 32%(148/471*100%) 정도가 재사용되었음을 알 수 있다.

7.2 리엔지니어링된 테스트 케이스의 시간 효율성

<표 3>은 기존의 테스트 케이스와 리엔지니어링 된 테스트 케이스를 이용하여 동일한 모델에 대해 테스트를 수행하는 동안 소요된 시간을 보이고 있다.

테스트에 걸린 시간을 조사하기 위해 테스트를 수행하면서 모든 체크리스트에 대해 초 단위로 시간을 기록했다. <표 3>에서 보는 바와 같이 기존의 테스트 케이스는 6시간 54분 59초로 약 415분이 소요되었고 리엔지니어링 된 테스트 케이스에 대한 테스트 수행 시간은 12시간 32분 56초로 약 753분 정도가 소요되었다. 따라서 기존의 테스트 케이스 하나를 테스트하는데 소요된 시간은 평균 4.15분(249초)이 걸린다. 하지만 리엔지니어링된 테스트 케이스의 테스트 수행 시간은 약 1.6분(96초)이 걸렸다. 리엔지니어링 된 테스트 케이스가 기존의 테스트 케이스에 비해 테스트를 수행시간이 테스트 케이스 당 평균 2.5배 정도의 효율성이 있음을 알 수 있다. 이러한 이유로는 상세하게 작성된 테스트 케이스가 그렇지 않은 테스트 케이스에 비해 테스트 명세서를 이해하기 쉽기 때문이다. 또한 명확하게 구별된 체크리스트로 인해 테스트 엔지니어가 중복해서 테스트할 확률이 줄어

〈표 3〉 테스트 수행 시간 효율성

기존 테스트 케이스			리엔지니어링 된 테스트 케이스		
항 목	개수	소요시간	항 목	개수	소요시간
채널탐색	35	1시간 45분 27초	채널탐색	138	3시간 20분 24초
일정/예약	26	1시간 7분 53초	일정/예약	43	1시간 41분 38초
속성설정	12	34분	속성설정	101	3시간 6분 37초
채널편집	25	15분 22초	채널편집	45	30분 40초
초기화	9	3시간 7분 46초	초기화	141	3시간 51분 15초
장치정보	3	4분 31초	장치정보	3	2분 22초
총 계	**100**	**6시간 54분 59초**	총 계	**471**	**12시간 32분 56초**

들기 때문이다. 시스템에 대한 사전 지식이나 테스트 경험이 부족한 초보 테스트 엔지니어가 테스트를 수행했을 경우에는 기존 테스트 케이스와 리엔지니어링된 테스트 케이스 간의 효율성 차이가 더 클 것으로 예상된다. 왜냐하면 기존의 테스트 케이스는 기능에 대한 접근 절차나 작동 방법과 같은 정보가 결여되어 있어서 테스트를 수행하기 전에 시스템에 대한 사전 지식이 필요하다. 반면 테스트 항목에 대한 접근 절차와 작동 방법 그리고 시험을 위해 입력해야 하는 값들을 자세하게 기술한 리엔지니어링된 테스트 케이스의 경우는 사전 지식이 없더라도 상세한 테스트가 가능하기 때문이다.

7.3 리엔지니어링된 테스트 케이스의 커버리지

테스트 커버리지는 소프트웨어 시험에 대한 품질을 평가하는 하나의 방법이라 할 수 있다. 다시 말하면 테스트 케이스를 이용하여 제품에 대해 얼마나 잘 테스트가 이루어졌는지 알 수 있는 지표를 제공한다. 테스트 케이스의 커버리지를 측정하는 기준은 테스트 케이스의 기술 형식이나 방법 또는 시험 레벨에 따라 달라질 수 있다. 화이트 박스 테스트의 경우 커버리지는 문장 기준 또는 분기나 경로를 기준으로 한 테스트 케이스 커버리지가 있을 수 있다[2]. 따라서 동일한 범주의 화이트 박스 테스트 경우라도 각각의 커버리지를 비교하기 위해 획일적인 기준 방법을 설정하기에는 어려움이 있다. 또한 단위 테스트의 경우와 같이 소스코드를 이용한 테스트 케이스의 커버리지와 사용자의 요구사항을 만족하고 있는지 확인하기 위해 서술형식으로 작성한 시스템 테스트 레벨의 테스트 케이스는 서로 다른 관점의 커버리지 비교 기준이 필요하다. 본 연구의 시험 수준과 테스트 케이스의 형식은 시스템 레벨에서 기능을 시험하기 위해 서술형식으로 작성하였다. 따라서 본 연구의 테스트 커버리지는 작성한 테스트 케이스가 모든 기능을 대상으로 누락 없이 접근하고 있는지에 대한 여부와 기존 테스트 케이스 대비 새로 개발된 테스트 케이스의 양적 수치를 이용하여 간접적인 비교를 하도록 한다.

5.1절에서 언급했듯이 기존이 테스트 케이스는 시스템 기능을 충분히 접근하지 못하고 있다. 너무 추상적이거나 요약되어 있고 또는 누락된 기능들도 있다. 특정 기능에 대해 추상적이거나 요약되어 있는 정도를 객관적으로 측정하여 정량화하기는 쉽지 않다. 따라서 시스템이 제공해야 하는 기능에 대한 테스트 케이스의 접근 정도와 양적 수치로 간접 비교를 수행했다. (그림 4)의 기능 구조도의 최하위 기능 수준을 기준으로 시스템이 제공하는 기능의 개수는 모두 41개이다. 그러나 기존의 테스트 케이스는 35개의 기능에 대해서만 작성되어 있다. 사용자가 채널 스케쥴 표에서 채널들을 탐색하는 부분과 같은 기능은 누락되어 있다. 하지만 새로 작성된 테스트 케이스는 41개의 기능을 모두 포함하고 있고 각 기능에 대한 접근 절차까지도 시험 대상으로 정의했다. 따라서 새로 제안한 테스트 케이스는 기존 테스트 케이스에 비해 더 많은 부분을 확인 할 수 있게 되었다. 이는

〈표 4〉 테스트 케이스 수행 결과 비교

	테스트 케이스(T/C) 결과 비교		
비교 항목	기존 T/C	제안 T/C	증 가
기능 커버리지	약 85%	100%	약 15% 증가
T/C 갯수	100 개	471 개	4.7배 증가
오류추출율	12 개	16 개	약 1.3배 증가
1개당 T/C 수행 시간	4.15 분	1.6 분	약 2.5배 감소

<표 1>에서 제시한 좋은 테스트 케이스가 되기 위한 조건 중 하나인 테스트 커버리지에 대한 속성을 따르고 있다. 또한 전체 테스트 케이스의 양은 (그림 13)과 같이 기존 100개에서 471개로 약 4.7배 정도 증가했음을 알 수 있다. 이는 동일한 기능을 기존 테스트 케이스에 비해 더욱 세분화 한 결과이다. 따라서 새로 작성된 테스트 케이스는 기존 테스트 케이스에 비해 더욱 구체적이고 자세한 테스트가 가능하도록 작성되었다고 할 수 있다. <표 4>는 기존 테스트 케이스와 새로 제안한 테스트 케이스를 이용하여 시험한 결과와 차이점을 보이고 있다.

7.4 리엔지니어링된 테스트 케이스의 오류추출성

오류추출성은 2.2절에서 언급했듯이 테스트 목적에 맞는 오류를 잘 발견할 수 있어야 한다. 디지털 방송 수신 장치는 온도와 위도 그리고 고도와 습도, 지역별 방송 신호 체계 및 전송 프로토콜, TV의 특성 등에 따라 많은 영향을 받는다. 즉 개발환경에서의 시험뿐만 아니라 실제 사용 환경에서 예측할 수 없는 다양한 변수에 대한 고려도 필요하다. 하지만 본 연구는 필드 테스트가 아닌 시스템 수준의 기능에 대한 테스트 케이스를 리엔지니어링했다. 그리고 기존 테스트 케이스와 새로 개발한 테스트 케이스의 동일한 비교 기준을 위해 동일한 시험 환경을 이용했다. 실험에 참여한 테스트 엔지니어는 테스트 경험이 모두 3개월 미만의 견습사원으로 최대한 타겟 시스템에 대한 사전 지식 및 시험 경험이 없는 인원으로 구성되었다. 이는 개인적인 능력이나 타겟 시스템에 대한 사전지식보다는 테스트 케이스에 기술되어 있는 정보만을 이용하여 객관적인 비교 결과를 도출하기 위함이다.

<표 4>와 같이 비교 결과 기존 테스트 케이스를 이용한 실험에서는 모두 12개의 오류를 발견했다. 그리고 새로 제안한 테스트 케이스를 이용한 시험은 기존 테스트 케이스로 발견할 수 있었던 오류를 포함해 모두 16개의 오류를 찾아낼 수 있었다. 그리고 발견된 오류는 심각도 정도를 매우 심각, 중요, 미약과 같이 세 등급으로 구분했는데 대부분 중요 등급과 미약 등급에 분포되었다. 오류 추출률은 발견된 오류의 개수만을 비교했을 경우 기존 대비 약 1.3배 증가했다. 테스트 커버리지의 증가 정도를 측정하기 위한 간접적인 방법으로 테스트 케이스의 양만을 이용한 경우 4.7배 정도가 증가했지만 실제 오류 추출율은 1.3배 정도 증가했다. 이는 타겟 시스템에 대한 검증이 오랜 기간에 걸쳐 이미 끝난 상태이기 때문에 새로 개발된 태스트 케이스의 커버리지 증가에 비례한 오류 추출 성능을 기대하기 어렵기 때문이다. 하지만 이미 시험이 끝난 후 시장에 공개된 제품에 대

해 또 다른 기능적 오류를 발견했다는 것에는 잘 설계된 테스트 케이스의 필요성을 잘 보여주고 있다.

8. 결 론

본 논문에서는 시스템 테스트 레벨에서 작성된 블랙박스 테스트 케이스를 분석하고 이에 대한 재사용과 테스트 효율성의 관점에서 리엔지니어링한 절차와 방법을 소개했다. 테스트 케이스의 리엔지니어링은 일단 기본 테스트 케이스와 테스트 대상 시스템을 분석하는 것으로 시작한다. 그리고 테스트 케이스 서식을 설계하고 재사용 가능한 컴포넌트를 추출하고 테스트 케이스를 재작업했다. 또한 기능과 모델을 고려하여 재사용하기 위한 단위를 살펴봤다. 재사용 단위로는 기본 모델을 구성하고 있는 기능 중 필수적인 기능을 위주로 선택했다. 그리고 실제 생산되는 모델에 대한 테스트 케이스를 분석하고 리엔지니어링하여 재사용이 가능한 새로운 테스트 케이스를 작성했고 이에 대한 재사용성을 기존 테스트 케이스와 비교했다. 리엔지니어링 된 테스트 케이스의 정보량은 약 5배 정도로 기존의 테스트 케이스에 비해 더 많은 정보의 기록이 가능해졌다. 테스트 케이스이 개수는 3.5배 정도 증가했다. 더 많은 정보와 누락된 기능들에 대한 시험 항목이 모두 포함했다. 또한 리엔지니어링 된 테스트 케이스의 테스트 수행 시간도 비교하여 기존 테스트 케이스와의 생산성 및 효율성에 대해 차이점을 보였다. 리엔지니어링 기법을 사용한 테스트 케이스의 작성 시간은 평균 테스트 케이스 1개당 6분 정도가 절약되었다. 이는 기능과 모델의 특징을 적절히 이용하여 재사용이 가능하도록 작성한 새로운 테스트 케이스의 재사용성으로 인해 테스트 케이스의 개발 시간이 단축됨을 보인 것이다. 또한 중복됨이 없이 상세하게 기술된 테스트 케이스가 그렇지 않은 테스트 케이스에 비해 테스트 수행 시간이 테스트 케이스 당 평균 2.5배 정도의 효율성이 높음을 보였다. 그 결과 기존의 테스트 케이스를 리엔지니어링함으로써 테스트 케이스를 개발할 때나 또는 개발된 테스트 케이스를 이용하여 테스트를 수행할 때 많은 시간을 절약할 수 있음을 알 수 있다. 또한 잘 설계된 테스트 케이스는 누락된 기능 없이 자세히 시험 항목들을 기술함으로써 테스트 커버리지를 높임으로써 더 많은 오류를 발견할 수 있음을 보였다.

참 고 문 헌

[1] Tassey, G., "The Economic Impacts of Inadequate Infrastructure for Software Testing: Final Report," National Institute of Standards and Technology, 2002.

[2] Glenford J. Myers, 'The Art of Software Testing', Second Edition, John Wiley & Sons. 2004.

[3] B. Broekman and E. Notenboom, 'Testing Embedded Software', Addison-Wesley, 2003.

[4] H. Gomma, 'Designing Software Product Lines with UML,' Addison Wesley, 2004.

[5] Ross, K. 'Practical Software System Testing, Lecturing Note', K. J. Ross & Associates Pty. Ltd. 1998.

[6] 서광익, 최은만, "다양한 블랙박스 테스트 기법들의 테스트 성능 비교," 정보과학회논문지: 소프트웨어 및 응용, 2005.

[7] 김진철, 정태욱, "모바일 소프트웨어 테스트 자동화," STEN Journal, Vol.3, pp.61-66, 2005.

[8] IEEE Std 610. 12-1990, "IEEE Standard Glossary of Software Engineering Terminology," IEEE, 1990.

[9] Allen, L., "Taking a Peek Inside the Black Box," Astek Engineering, 2001.

[10] W. M. Rlrich, 'Re-engineering: Defining an Integrated Migration Framework,' CASETrends, May, 1991.

[11] E. Yourdon, 'Re-3, Part1: Re-Engineering, Restructuring, and ReverseEngineering,' AmericanProgrammer,, Vol.2, No.4, pp.33-10, April, 1898.

[12] H. Mili, A. Mili, S, Yacoub, E. Addy, 'Reuse-Base Software Engineering', Jone Wiley & Sons, pp.7-9, 2002.

[13] Ted J. Biggerstaff and Alan J. Perils, 'Software Reusability,' Vol.1: Concepts and Models, Frontier Series, ACM Press 1989.

[14] A. von Mayrhauser, R. Mraz, J. Walls, P. Ocken, "Domain Based Testing : Increasing Test Case Reuse," Proceedings of Computer Design: VLSI in Computer and Processors, ICCD 94, IEEE, OCT. 10-12, 1994.

[15] Lee White and Edward Cohen. 'A Domain Strategy for Computer Program Testing,' IEEE TSE, SE-6(3), pp.247-257, May, 1980.

[16] R S. Arnold, 'Software Reengineering', IEEE Computer Society Press Tutorial, 1993.

[17] R. S. Arnold, 'Software Restructuring,' IEEE Proc. Vol.77, No.4, pp.607-617, April, 1989.

서 광 익

e-mail : emchoi@dgu.ac.kr
2002년 동국대학교 컴퓨터공학과(학사)
2004년 동국대학교 컴퓨터공학과(공학석사)
현 재 동국대학교 컴퓨터공학과(박사)
관심분야 : 소프트웨어 테스트, 소프트웨어 품질, 임베디드 소프트웨어 테스트, 테스트 프로세스

최 은 만

e-mail : emchoi@dgu.ac.kr
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
관심분야 : 객체지향 설계, 소프트웨어 테스트, 프로세스와 메트릭, Program Comprehension