항공용 소프트웨어 개발 모델 및 테스팅 설계 기법에 관한 연구

윤원근* · 이백준* · 진영권*

Study of Avionic Software Development Model & Testing Design Methods

Wonkeun Youn* · Baeckjun Yi* · Youngkwon Jin*

ABSTRACT

The paper presents the comprehensive studies of ISTQB(International Software Testing Qualification Board) for the reliable software development system in term of various aspects. It is critical to understand how the software testing is achieved is to efficiently manage the development process, to reduce the development cost, and to ultimately ensure the safety of the entire software system. This study may provide an enriched understanding about the rationale and true intent the behind software development model and testing design methods of ISTQB on software developer, test analysts, test engineers. In addition, this paper may serve as a useful supplementary material for the avionic engineers to establish the new regulations in avionic industries.

Key Words: Software Test, Software Quality Assurance, ISTQB, Avionic Software, DO-178B

1. 서 론

소프트웨어는 다양한 사업용 어플리케이션에서 국방, 항공에 이르기까지 폭넓게 생활의 많은 부분에서 사용되고 있으며 그 비중은 계속해서 증가하고 있다. 이러한 소프트웨어의 신뢰성이 보장되지 않는 경우 다양한 문제가 발생할 수 있다. 이로 인한 피해는 금전적인 손실, 시간 낭비, 그리고 인명 부상이나 사망에 이르기 까지 다양하고 심각하며, 소프트웨어 테스팅은 이러한 소프트웨어 시스템의 문제를 최소화하기 위해서 필수불가결한 요소이다.

소프트웨어 테스팅의 중요성에도 불구하고 소프트웨어 테스팅에 대한 각 표준마다 사용하는 용어도 상이하고 실무적인 내용도 직접적으로 활용하기가 어려운 현실이다. 특별히 항공 분야에서는 하드웨어와 소프트웨어가 통합적으로 개

†2013년 7월 2일 접수 ~ 2013년 9월 24일 심사완료
* 정회원, 한국항공우주연구원
연락저자, E-mail: wkyoun@kari.re.kr

발되지만 사실상 소프트웨어가 전체 시스템의 안전성을 결정하는 핵심 요소이다. 근래에 들어 항공분야에 사용되는 소프트웨어는 하드웨어 성능의 획기적 개선, 컴퓨터 구조의 변화, 메모리와 저장용량의 증가를 통해 현저하게 발전하였다. 따라서 항공 소프트웨어의 신뢰성을 확보하기 위해 다양한 군용 및 민간용 소프트웨어 규격들이 개발되었다

그 중에서 RTCA/DO-178B “항공기 탑재 시스템 및 장비 인증에 있어서의 소프트웨어 고려사항”는 민간항공 분야에서 탑재용 소프트웨어 승인을 위한 지침으로서 현재 전 세계의 항공업계에서 활발히 사용되고 있다. 하지만 최근 소프트웨어 개발 기술의 비약적인 발전으로 DO-178B가 다루지 못하는 인증 문제들이 늘어남에 따라 DO-178C로 2011년도 개정되었으며, 향후 계속적인 개정 논의가 있을 것으로 예상된다.

따라서 본 논문에서는 핵심적인 소프트웨어 테스팅 분야의 표준적인 지식체계 중 한가지인 ISTQB(International Software Testing Qualification Board)를 중심으로 소프트웨어 테스팅의 원리와 테스트 설계 기법을 살펴보고자 한다. 향후 민간 항공 소프트웨어 규격을 개정하고자 할 때 이 지식체계의 개념을 이용한다면 더욱 효율적으로 인증 문제를 해결 할 수 있을 것으로 예상된다.

2. 소프트웨어 수명주기와 테스팅

2.1 소프트웨어 개발 모델

소프트웨어 테스팅은 개발활동과 독립적으로 존재하지 않고 밀접하게 연계되어 있으므로, 개발 수명주기 모델에 따른 적용 방법은 다음과 같다.

2.1.1 V-모델(Sequential Development model)

V-모델은 요구사항 정의 및 분석, 시스템 설계, 구현, 테스팅이라는 일련의 과정을 통해 소프트웨어 시스템을 개발하는 폭포수 개발 모델(Waterfall model)을 기반으로 하고 있다. 이 프로세스는 Fig.1과 같이 코딩단계에서 위쪽으로 꺾여서 알파벳 V자 모양으로 진행된다. V-모델은 개발 생명주기의 각 단계와 그에 상응하는 소프트웨어 시험 각 단계의 관계를 보여준다.

다양한 V-모델이 존재하지만 V-모델의 공통적인 부분은 다음의 4단계의 테스트 레벨을 제시하고 있다.

- 컴포넌트(단위) 테스팅(Component Testing)
- 통합 테스팅(Integration Testing)
- 시스템 테스팅(System Testing)
- 인수 테스팅(Acceptance Testing)

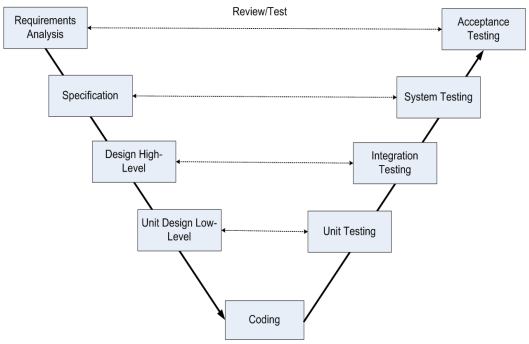

Fig. 1 V-model (Sequential Development Model)

2.1.2 반복적-점증적 개발모델(Iterative-incremental Development Models)

반복적-점증적인 개발방법은 요구사항 분석, 시스템 설계, 구현 및 테스팅 하는 개발 주기가 연속적으로 짧게 반복적인(Iterative) 활동으로 이루어진다. 이러한 방법은 초기 개발 반복단계에서 리스크가 높은 모듈이나 중요한 소프트웨어 아키텍처에 해당하는 시스템 일부를 우선적으로 개발하고 테스팅을 통해 결함을 조기에 발견하고 제거할 수 있는 장점을 가지고 있다.

2.1.3 개발 수명주기(Life Cycle) 모델에서의 테스팅

개발 수명주기 모델에서는 모든 개발활동은 대응되는 테스팅 활동을 수반하며 각 테스트 레벨은 그 레벨에 맞는 특정한 목적을 가지고 있다. 또한 주어진 테스트 레벨에 맞는 테스트의 분석과 설계는 대응되는 개발활동 동안에 시작

되어야 한다.

2.2 소프트웨어 테스트 레벨

2.2.1 컴포넌트 테스팅(Component Testing)

컴포넌트 테스팅은 테스트가 가능한 최소 단위로 나누어진 소프트웨어(모듈, 프로그램, 객체, 클래스 등) 내에서 프로그램 명세서에 정의된 바와 같이 기능의 오류 없이 수행되는지를 검증하는 테스트로 개발자 중심의 테스트이다. 일반적으로 컴포넌트 테스팅은 개발 수명주기의 상황과 시스템에 의존적이면서도 시스템의 다른 부분에서 분리하여 독립적으로 수행되어야 한다.

컴포넌트 테스팅은 화이트박스 테스트와 블랙박스 테스트 기법 모두 적용 가능하지만, 일반적으로 화이트박스 테스트 기법을 사용하며, 기능 테스트와 리소스 활용(ex. Memory leaks)과 같은 비 기능 테스트 케이스가 포함될 수 있다.

컴포넌트 테스팅 시에는 에러를 줄이기 위한 의도로 작성된 코드에 대한 분석을 진행한다. 또한 코드가 효율적으로 작성되었는지, 프로젝트 내에 합의된 코딩 표준을 준수하고 있는지도 검증한다. 모듈 설계 단계에서 준비된 테스트 케이스를 이용하며, 코드를 개발한 개발자가 직접 수행한다.

2.2.2 통합 테스팅(Integration Testing)

통합 테스팅은, 개별 모듈들을 통합하여, 통합된 컴포넌트 간의 인터페이스와 상호작용 상의 오류를 발견하는 작업을 수행한다. 통합 테스팅은 하나 이상의 테스트 레벨이 있을 수 있으며, 다양한 크기의 테스트 대상에 대해 수행될 수 있다. 통합 테스팅은 개발된 프로그램과 다른 시스템과의 상호작용 테스트하며, 테스트된 모듈들과 다음에 테스트될 모듈을 결합시킨 후 테스트하는 과정을 거친다. 통합 테스팅을 통해서는 오류의 원인을 찾기 어려우며, 디버깅에 많은 시간을 소비되는 경향이 있다.

일반적으로 이 단계에서는 테스트는 블랙박스 테스트이며, 따라서 코드를 직접 확인하는 형태는 아니다. 아키텍처 설계 단계에서 준비된 테스트 케이스를 사용하여 테스트가 진행되며, 일반적으로 개발자가 진행한다. 또한 통합 테스팅은 기능적 특성과 비 기능적 특성을 모두 포함해야 한다.

2.2.3 시스템 테스팅(System Testing)

시스템 테스팅은 프로그램이 설정한 성능 목표를 만족할 수 있는지를 확인하기 위한 것으로 실제 구현된 시스템과 계획된 사양(Specifications)을 서로 비교하는 작업으로서 개발 프로젝트 차원(범위)에서 정의된 전체 시스템 또는 제품의 동작에 대해 테스트 하는 것이다. 테스팅에서 발견하지 못하여 발생할 수 있는 “환경특성 장애(Environment-specific failure)”리스크를 최소화하기 위해서 시스템 테스팅은 가능한 범위에서 실제 최종 사용 환경과 유사한 환경에서 수행해야 한다.

시스템 테스팅은 기능 및 비 기능 요구 사항을 모두 검증해야 한다. 기능적 요구 사항의 시스템 테스팅은 테스트 대상의 특성을 가장 상세하게 명세한 문서를 기반으로 테스트를 설계하는 명세기반(블랙박스) 기법을 수행한다. 비 기능적 요구사항의 시스템 테스팅은 소프트웨어의 기능적 품질 특성 외의 나머지 부분에 대한 요구사항 검증을 수행한다.

2.2.4 인수 테스팅(Acceptance Testing)

인수 테스팅은 소프트웨어를 이용할 최종 고객이 주로 수행하는 시험으로서 시스템이나 시스템의 일부 또는 특정한 비 기능적인 특성에 대해 “확신(Confidence)”를 얻는 것으로서 시스템을 배포하거나 실제 사용할 만한 준비(Readiness)가 되었는지에 대해 평가한다.

인수테스팅은 비즈니스 사용자가 시스템 사용의 적절성을 평가하는 사용자 인수 테스팅과 백업/복원 테스팅, 재난 복구, 사용자 관리, 유지보수 작업, 보안 취약성에 대한 정기적인 점검과 같은 시스템 관리자에 의한 운영상의 테스팅을 포함하며, 소프트웨어 계약상의 인수 통과

조건을 준수하는 지 확인하는 계약 인수 테스팅, 정부 지침, 법률 또는 안전 규정 등을 준수하는지 확인하는 규정 인수 테스팅이 있다.

또한 소프트웨어가 상업적으로 배포되기 전에 시장의 기존 고객이나 잠재 고객으로부터 피드백을 받기 위해 개발 조직 내의 고객에 의해 수행되는 알파 테스팅(Alpha Testing)과 실제 환경에서 사용자 또는 잠재 고객에 의해 수행되는 베타 테스팅(Beta Testing) 또는 필드 테스팅(Field Testing)이 있다.

3. 정적 기법(Static Technique)

정적 기법은 소프트웨어를 실행하지 않고 테스팅 하는 기법으로 리뷰와 같은 수동적(Manual) 기법과 정적 분석 자동화 도구를 활용한 정적 분석이 있다. 리뷰는 코드를 포함하여 소프트웨어 개발 및 테스트 산출물(프로그램 코드, 요구사항 명세서, 설계 명세서, 테스트 계획서, 테스트 설계서, 테스트 케이스, 테스트 스크립트, 사용자 지침서 등)을 검토하고 테스팅 하는 방법이며, 이는 일반적으로 동적 테스팅을 실행하기 전에 수행된다.

리뷰는 주로 수동으로 진행되지만 도구의 도움을 받을 수도 있으며, 수동으로 진행하는 리뷰 활동은 개발 산출물을 검토하고 의견을 제시하는 것이다. 요구사항 명세, 설계 명세, 코드, 테스트 계획, 테스트 명세, 테스트 케이스, 모든 소프트웨어 개발 및 테스트 산출물은 리뷰의 대상이 될 수 있다. 리뷰를 통해서 조기 결함을 발견하고 수정할 수 있고, 이에 따라 개발 생산성이 향상되고 개발기간이 단축 될 수 있다. 또한 테스팅 비용 감소 및 시간 단축과 개발 수명주기 전체에 걸친 비용 감소의 효과를 얻을 수 있다.

리뷰와 정적분석, 동적 테스팅은 모두 결함 발견이라는 동일한 목적을 가지고 있지만, 이들은 상호 보완적이어서 각각의 기법은 서로 다른 종류의 결함들을 효과적이고 효율적으로 발견할 수 있다. 정적 기법은 동적 테스팅과는 달리 장애(Failures) 자체보다는 장애의 원인(결함)을 발견한다. 동적 테스팅보다 리뷰를 통해서 발견하기 쉬운 결함의 종류는 다음과 같다.

- 표준 위반
- 요구사항 결함
- 개발 설계(Design) 결함
- 불충분한 유지 보수성
- 부정확한 인터페이스 명세

4. 소프트웨어 테스트 설계 기법

테스트 기법은 전형적으로 블랙박스와 화이트박스로 구분할 수 있다. 블랙박스 기법(명세 기반 기법과 경험 기반 기법을 포함함)은 테스트 대상의 내부구조(코드)를 참조하지 않고 테스트 베이시스(Test Basis), 그리고 개발자와 테스터, 사용자들의 경험을 분석하여 기능적 혹은 비 기능적인 테스트 조건으로 테스트 케이스를 도출하는 방법이다. 반면 화이트박스 기법(구조 기반 기법)은 컴포넌트 혹은 소프트웨어(시스템)의 구조(코드)에 대한 사전 지식을 기반으로 테스트 케이스를 도출하는 방법이다[1].

소프트웨어(시스템) 내부구조(코드)의 참조 여부에 따라 블랙박스 기법과 화이트박스 기법으로 양분하는 방법 이외에 테스트 설계의 근원을 기준으로 명세 기반 기법, 구조 기반 기법, 경험 기반 기법으로 분류할 수 있다(Fig.2).

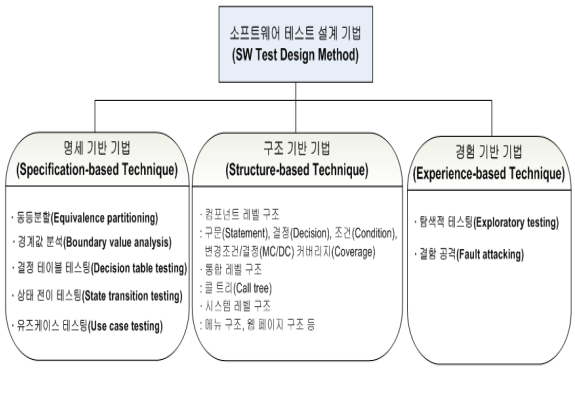

Fig. 2 Software Test Design Methods

4.1 명세기반 기법(Specification-based technique)

명세 기반 기법은 주어진 명세(일반적으로 모델의 형태)를 바탕으로 테스트 케이스를 도출하는 것을 의미하며, 프로그램이 어떻게 구현되어 있는지 관여하지 않고 의도된 대로 동작하는지 확인하는 것이다. 명세 기반 기법은 해결한 문제를 명세하기 위해 공식적 또는 비공식적 모델을 사용하며, 이러한 모델에서 테스트 케이스를 시스템적으로 도출하는 것이 가능하다. 명세 기반 기법은 블랙박스 기법으로 간주되며 커버리지를 측정할 수 있으나 그 의미가 구조 기반 기법의 커버리지에 비해 제한적이다. 명세 기반 기법에는 다음과 같은 종류가 있다.

4.1.1 동등 분할(Equivalence partitioning)

소프트웨어나 시스템이 특정 범위의 입력값에 의해 결과 값이 동일할 경우 이러한 입력값의 범위를 하나의 동등 그룹(클래스)으로 간주할 수 있다. 그리고 동일 범위 내의 입력값은 내부적으로 같은 방식으로 처리된다고 가정하였을 때, 입력 값/출력 값 영역(Input/Output Space)을 유한개의 상호 독립적인 집합으로 나누어 수학적인 등가집합을 만든 후 각 등가 집합의 원소 중 대푯값 하나를 선택하여 테스트 케이스를 작성하는 것이 동등분할이다.

동등 분할 클래스의 하나의 입력 테스트는 나머지를 모두 테스트한 것과 같은 의미를 지니며, 입력값 이외에 동등 분할을 적용할 수 있는 분야는 다음과 같다.

- 출력값(Outputs)
- 내부값(Internal Values)
- 시간관련 값(Time-related values, 이벤트 이전과 이후)
- 통합 테스팅에서 다루는 모듈 간 인터페이스 파라미터(Interface parameters)

4.1.2 경계값 분석(Boundary value analysis)

경계값 분석은 많은 결함이 입력값에 근접한 값에 집중되는 경향이 있다는데 기반하여 동등 분할의 경계부분에 해당되는 경계값까지 포함하여 테스트 하는 기법이다. 해당 분할영역의 최대값과 최소값은 그 영역의 경계값이 된다. 경계값 분석은 결함 발견율이 높고, 적용하기 쉬운 장점이 있어 가장 많이 사용되는 테스트 기법 중 하나이다. 경계값 분석은 종종 동등 분할의 확장으로 여겨지며, 동등 분할과 동일한 방식으로 커버리지를 보장한다.

4.1.3 결정 테이블 테스팅(Decision table testing)

결정 테이블(Decision table)은 논리적인 조건이나 상황(Conditions)을 구현하는 시스템에서 요구사항을 도출하거나 내부 시스템을 문서화하는데 매우 유용한 도구이다. 이것은 시스템이 구현해야 할 비즈니스 규칙(Business rules)을 문서화 하는데 사용된다. 명세를 분석하고, 시스템의 조건과 동작(Actions)을 식별한다. 결정 테이블에서 입력조건과 기대결과는 참(True)과 거짓(False)으로 표현된다. 결정 테이블은 동작을 유발시키는 조건 또는 상황(Triggering conditions – 주로 모든 입력 조건에 대한 참과 거짓의 조합) 그리고 각 해당 조합의 예상 결과까지 포함한다.

결정 테이블 기법은 소프트웨어 동작이 여러 가지 논리적 조건에 의존적인 모든 경우에 적용이 가능하며, 요구사항 등 테스트 베이시스의 문제점을 드러나게 하는데 효과적인 테스트 케이스를 생성할 수 있다. 하지만 작성에 많은 노력과 시간이 소요될 수 있으며, 복잡한 시스템은 표현하기 어렵고 작성 시 논리적 실수를 할 가능성이 높다.

4.1.4 상태 전이 테스팅(State transition testing)

시스템은 현재 상황(Conditions)과 이전의 이력(History)을 반영하는 상태(States) 및 그 변화(Transition)에 따라 다르게 동작할 수 있다. 시스템의 이러한 측면을 상태 전이 다이어그램(State transition diagram)으로 표현할 수 있다. 상태 전이 다이어그램을 통해 소프트웨어 또는 시스템을 상태 사이의 관계 즉, 상태 간의 전이, 상태를 변화시키는 이벤트와 입력값, 상태의

변화로 유발되는 동작 등으로 파악하고 이를 통해 다음과 같은 방식으로 테스트를 설계하는 것이 가능하다.

- 전형적인 상태의 순서를 커버하는 방식
- 모든 상태를 커버하는 방식
- 모든 상태 전이를 실행하는 방식
- 특정한 상태 전이 순서를 실행하는 방식
- 불가능한 상태 전이를 테스트 하는 방식

4.1.5 유즈케이스 테스팅(Use case testing)

유즈케이스 테스팅은 시스템과 사용자 사이의 상호작용을 표현하고, 해당 상호작용은 시스템 유저에게 결과 값을 제공하는 유즈케이스 또는 비즈니스 시나리오를 기반으로 테스트를 명세화 한다. 유즈케이스는 대개 시나리오 또는 기본 흐름으로 구성되어 있고 개별 유즈케이스는 자세하게 표현하기 위해 유즈케이스 상세(Use case description)를 갖는다. 유즈케이스는 시스템이 실제 사용되는 방식에 기반을 두어 “프로세스 흐름” 을 기술한다. 따라서 유즈케이스에 기반을 두어 생성된 테스트 케이스는 시스템이 실제 사용되는 프로세스 흐름에서 결함을 발견하는데 상당히 유용하다.

4.2 구조 기반 기법(Structure-based Technique)

구조 기반(화이트 박스) 테스팅은 소프트웨어나 시스템의 구조를 중심으로 테스팅 하는 것이다[2]. 시스템 또는 소프트웨어의 구조가 테스트 스위트에 의해 테스트된 정도를 커버리지(Coverage)라고 하며, 특정 구조의 종류에 대해 커버된 백분율로 표시한다. 구조 기반 테스팅의 커버리지의 종류는 다음과 같다(Fig.3). (전체 조건식은 If나 While에 있는 조건문 전체와 같이 코드에서 분기하는 결정 포인트 전체를 의미한다. 개별조건식은 전체조건식에 연산자(And, Or 등)로 구분되어 있는 개개의 조건식을 의미한다.)

- 구문 커버리지(Statement Coverage): 테스트 의해 실행된 구문이 몇 퍼센트인지 측정하는 것으로 다른 커버리지에 비해 가장 약하다.
- 결정 커버리지(Decision Coverage): 결정 포인트 내의 전체조건식이 최소한 참(True)이 한번 그리고 거짓(False)이 한 번씩 선택되었는지 측정하여 퍼센트로 표현하는 것이다.
- 조건 커버리지(Condition Coverage): 전체 조건식의 결과와 관계없이 각 개별조건식이 참 한번, 거짓 한번을 모두 갖도록 개별조건식을 조합하는 것으로 결정커버리지보다 강력한 형태의 커버리지이다.
- 변경조건/결정 커버리지(Condition/decision Coverage): 각 개별조건식이 다른 개별조건식에 무관하게 전체조건식의 결과에 독립적으로 영향을 주는 커버리지이다.
- 다중조건 커버리지(Multiple condition Coverage): 결정 포인트 내에 있는 모든 개별조건식의 모든 가능한 논리적인 조합을 고려한 것으로 모든 경우에 100% 커버리지 보장하는 것이다.

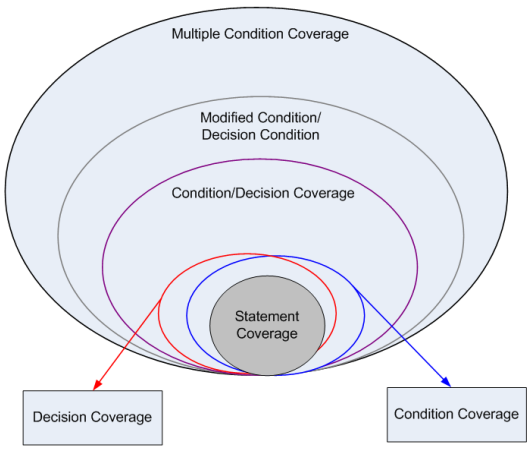

Fig. 3 Relationship between various coverages

4.3 경험 기반 기법(Experience-based Technique)

경험 기반 테스팅은 이전에 테스터가 다루었던 유사 어플리케이션이나 기술에서의 경험, 직관, 테스터의 기술 능력으로부터 테스트 케이스를 추출해 내는 방식으로 이는 엔지니어의 지식에 따라 효과성이 결정된다. 체계적인 기법을

보강하기 위해 추가적으로 경험 기반 기법을 적용하는 것은 공식적인 기법으로 다루기 어려운 특별한 테스트 케이스를 찾아내고 실행하는데 유용하다[3].

공식적인 테스트 기법과 경험 기반 테스트 기법으로 구현된 테스트 케이스는 찾아낼 수 있는 결함의 종류가 제각기 다르기 때문에 경험기반 테스트 기법이 의미를 갖는다. 경험기반 기법 중에 대표적인 기법으로는 탐색적 테스팅으로서 이는 테스트 설계, 테스트 수행, 테스트 계획, 테스트 기록 및 학습을 동시에 직행하는 발견적인(Heuristic) 테스팅 접근법이 있다.

5. 결 론

최근 컴퓨터의 하드웨어 기술이 급진적으로 발전하고, 사용자의 소프트웨어에 대한 요구사항도 점차로 복잡해지고 있는 상황에서 요구사항을 충족시키지 못하는 기술의 격차와 관리 방법론의 부적합성, 전문 인력의 부족 현상 등은 이러한 위기를 점점 심화시키고 있다. 또한 오늘날 소프트웨어를 개발하고 보수, 유지 할 수 있는 능력의 필요성은 충분히 인식되어 많은 연구개발이 추진되고 있으나 소프트웨어 기술과 요구사항간의 격차는 아직까지도 존재하고 있다.

특별히 소프트웨어는 항공 시스템 전체의 신뢰성을 결정짓는 매우 중요한 요소이다[4]. 실제로 최근 항공기에서 소프트웨어가 담당하는 기능은 통신, 항법, 자세제어, 자동비행, 엔진제어 등 그 범위가 매우 광범위하고 이에 따른 산업규모도 빠르게 성장하고 있다. 또한 항공 디지털 시스템의 급속한 발전으로 소프트웨어가 복잡화, 고도화되면서 안전성에 치명적인 항공기 기능의 소프트웨어의 사용이 증가함에 따라 새로운 안전성과 인증 논의가 대두되고 있다[5].

본 논문에서 소개하는 소프트웨어 테스팅 분야의 표준적인 지식체계인 ISTQB 소프트웨어 테스팅은 소프트웨어의 신뢰성을 보장하는 데 있어 효과적이라는 것이 증명되어 왔다. 따라서 본 논문의 소프트웨어 테스트 체계지식에 따라 민간 항공용 소프트웨어 인증 규격을 새로 규정하고 이를 통해 테스팅을 진행할 경우 비용과 시간 측면에서 더욱 효율적으로 인증 지침을 충족시키는 항공용 소프트웨어를 개발할 수 있을 것으로 기대된다. 또한 소프트웨어 테스팅에는 정답이 아닌 모범답안이 존재하기 때문에 다양한 실무 경험이 곁들인다면 테스팅 개념 및 실무이론 상호간의 연관성을 이해하는데 큰 도움이 될 것이다.

참 고 문 헌

[1] Tomas Muller, et al,"Certified Tester, Foundation Level Syllabus". ISTQB(International Software Testing Qualification Board), 2005.

[2] 김도균, “소프트웨어 테스팅”, 2006.

[3] 권원일 외 4명, “개발자도 알아야할 소프트웨어 테스팅 실무”, 2011.

[4] 박무혁, 항공용 S/W 개발 및 인증 기술 동향”, 항공우주산업기술동향, 제 5권, 제 1호, 2007, pp. 15-24.

[5] 이백준, 김성겸, “비항공용 소프트웨어의 설계 · 인증 고려사항”, 항공우주기술, 제 3권, 제 1호, 2004, pp. 178-182.