제 2 장 테스트 분류와 테스팅 방법

2.1 개 요

테스트와 관련된 많은 개념이 존재한다. 예를 들어 컴포넌트 테스트, 통합 테스트, 시스템 테스트, 인수 테스트라는 용어도 있고, 기능 테스트와 비기능 테스트라는 개념도 있다. 또한, 정적 테스트, 동적 테스트라는 용어도 있다. 그뿐만 아니라 위험 기반 테스트, 리그레션 테스트, 모델 기반 테스트 등의 용어도 존재한다. 본 장에서는 이렇게 다양한 테스트 관련 개념을 테스트 분류와 테스팅 방법으로 구분하여 설명한다.

- 테스트 분류에서는 테스트 레벨(컴포넌트, 통합, 시스템, 인수), 테스트 유형(기능, 품질), 그리고 테스트 설계 기법을 기준으로 테스트 분류를 설명한다.
- 테스팅 방법에서는 적용하는 개발 생명 주기, 프로젝트 단계(개발 또는 유지 보수) 등 프로젝트의 상황을 고려하여 현실적으로 적용할 수 있는 테스트 방법을 설명한다.

2.2 테스트 분류

2.2.1 개요

소프트웨어 테스트는 테스트 레벨, 테스트 유형, 그리고 테스트 설계 기법에 따라서 다양하게 분류될 수 있다. 이들 3가지 기준은 각각 독립적인 기준이므로 그림 2.1과 같이 3차원 축으로 분류하여 나타내었다.

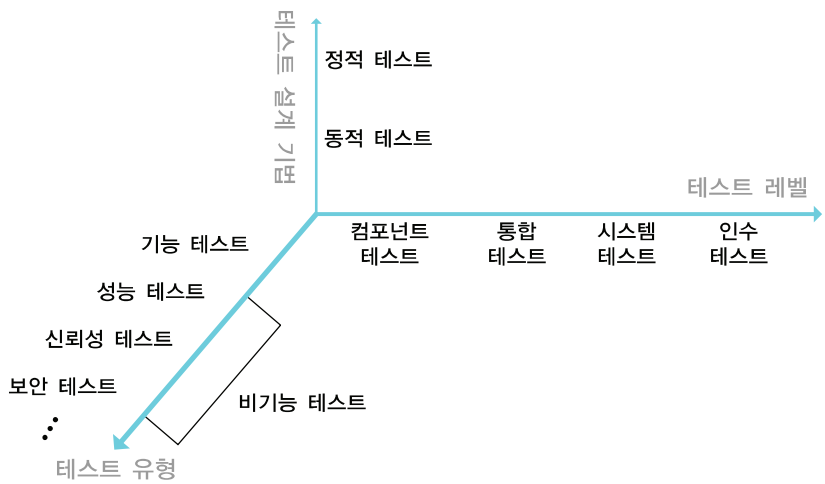

그림 2.1 테스트 분류

테스트 레벨은 컴포넌트 테스트, 통합 테스트, 시스템 테스트, 인수 테스트로 분류되고, 테스트 유형은 기능 테스트와 성능 테스트, 신뢰성 테스트, 보안 테스트 등으로 분류된다. 테스트 유형에서 기능 테스트를 제외한 성능 테스트, 신뢰성 테스트, 보안 테스트 등을 비기능 테스트라고 부른다. 그리고 테스트 설계 기법은 정적 테스트와 동적 테스트로 분류할 수 있다.

2.2.2 테스트 레벨에 의한 분류

테스트의 레벨(Level)에 따라서 컴포넌트(또는 단위) 테스트, 통합 테스트, 시스템 테스트 그리고 인수 테스트로 분류된다. 표 2.1은 테스트 레벨에 따라 분류한 각 테스트를 설명하고 있다.

표 2.1 테스트 레벨

테스트 기법	설명
컴포넌트(Component)/ 단위(unit) 테스트	시스템을 구성하는 단위 모듈을 테스트 대상으로 하여 개별 단위 모듈을 독립적으로 테스트한다.
통합(Integration) 테스트	시스템을 구성하는 단위 모듈들이 정확하게 통합되었는지에 초점을 둔다. 시스템 내부 구성 모듈과 이들 간의 관계를 정의한 구조 설계 명세서를 바탕으로 테스트가 진행된다.
시스템(System) 테스트	전체 시스템을 테스트 대상으로 하여 테스트가 진행된다. 요구사항 명세서에 명시된 방식대로 시스템이 동작하는지 확인하는 데 초점을 둔다.
인수(Acceptance) 테스트	시스템 테스트와 마찬가지로 전체 시스템을 하나의 단위로 보고 테스트가 진행된다. 그러나 시스템 테스트와 달리 인수 테스트는 고객/사용자의 관점에서 고객이 기대하는 방식으로 소프트웨어가 동작하는지 확인한다.

4개 레벨의 테스트는 일반적인 소프트웨어의 개발 단계 즉 요구 분석, 구조 설계(또는 아키텍처 설계), 상세 설계와 밀접한 연관이 있다. 그림 2.2는 소프트웨어 개발 각 단계와 그에 상응하는 테스트 레벨을 표현한 것이다. 좌측의 개발 단계와 우측의 테스트 수준이 알파벳 'V'형태를 이루므로 이를 V 모델이라고 부른다.

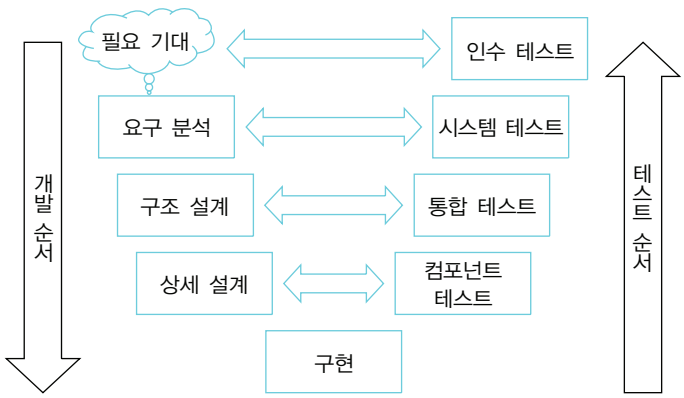

그림 2.2 V 모델

테스트는 컴포넌트 테스트, 통합 테스트, 시스템 테스트, 인수 테스트 순서로 진행된다. 그리고 각 테스트는 개발의 각 단계를 기준으로 진행된다. 예를 들어 컴포넌트 테스트는 단위 구성 요소(함수・클래스・컴포넌트 등)에 대한 상세 설계를 기준으로 구현된 단위를 테스트한다. 그리고 통합 테스트는 아키텍처 설계를 통하여 결정된 단위 간의 관계(호출 등)를 기준으로 통합된 단위를 테스트한다. 시스템 테스트와 인수 테스트는 각각 요구 분석과 고객・사용자의 필요・기대를 기준으로 테스트를 수행한다.

2.2.3 테스트 유형에 의한 분류

테스트를 통해 확인하고자 하는 소프트웨어의 동작 및 특성에 대한 기준은 요구사항 명세서에 정의된다. 그리고 요구사항 명세는 기능 요구사항과 품질 요구사항을 포함한다. 성능 효율성 테스트, 신뢰성 테스트 같이 개별 품질 특성에 초점을 두고 수행하는 테스트를 전통적으로 비기능(Non-functional) 테스트라고 부른다. 또한, ISO 29119 표준에서는 기능 테스트와 각각의 비기능 테스트를 유형 테스트(Type test)라고 부른다. 표 2.2는 테스트 유형을 기능 테스트와 비기능 테스트로 분류하여 설명한 것이다.

표 2.2 테스트 유형

테스트 유형	설명
기능(Functional) 테스트	• 기능 요구사항 측면의 결함 검출 및 충족 여부 확인을 목적으로 한다. • 모든 테스트 수준(컴포넌트 · 통합 · 시스템 · 인수 테스트)에서 진행된다.
비기능(Non-functional) 테스트	• 성능, 보안성, 신뢰성 등 품질 요구사항 측면의 결함 검출 및 충족 여부 확인을 목적으로 한다. • 일반적으로 시스템 테스트와 인수 테스트 수준에서 진행된다.

ISO 25010은 소프트웨어 품질 모델을 정의하고 있다. 소프트웨어 품질 모델은 소프트웨어가 갖춰야 하는 다양한 품질 특성에 관한 표준을 정의한 것이다. 표 2.3은 ISO 25010에서 정의한 품질 모델이다.

표 2.3 ISO 25010 품질 모델

주특성	부특성
기능 적합성	완전성, 정확성, 타당성
사용성	타당성 식별력, 학습성, 운영 용이성, 사용자 오류 보호, 사용자 인터페이스 미학, 접근성
성능 효율성	시간 행동(Time-behaviour), 자원 활용성, 수용성(Capacity)
호환성	공존성, 상호운영성
신뢰성	성숙성, 가용성, 장애 허용성, 회복 가능성
보안성	기밀성, 무결성, 부인방지, 책임성, 진본성
유지보수성	모듈성, 재사용성, 분석성, 변경 용이성, 테스트 용이성
이식성	적응성, 설치 용이성, 대치 용이성

ISO 25010 품질 모델은 기능 적합성, 사용성, 성능 효율성, 호환성, 신뢰성, 보안성, 유지보수성, 이식성을 주특성으로 정의한다. 그리고 이 8개의 주특성별로 세부적인 품질 특성을 부특성으로 정의하고 있다. 예를 들어, 호환성은 공존성과 상호운영성으로 세분화되고, 유지보수성은 모듈성, 재사용성, 분석성, 변경 용이성, 테스트 용이성으로 세분화된다.

이러한 품질 특성은 소프트웨어가 충족해야 하는 요소이며, 기능 요구사항과 더불어 소프트웨어 요구사항의 한 유형이다. 그러므로 소프트웨어가 이러한 각 품질 특성을 충족하는지 테스트를 수행해야 한다. 예를 들어, 소프트웨어가 주어진 성능 요구사항을 충족하는지 확인하기 위하여 성능 효율성 테스트를 수행할 수 있고, 신뢰성 요구사항에 대한 충족 여부를 판단하기 위하여 신뢰성 테스트를 수행할 수 있다.

2.2.4 테스트 설계 기법에 따른 분류

2.2.4.1 개요

테스트는 테스트 설계 기법에 따라서 정적 테스트와 동적 테스트로 분류된다. 정적 테스트는 리뷰와 정적 분석으로 분류되며 동적 테스트는 명세 기반 테스트, 구조 기반 테스트, 그리고 경험 기반 테스트로 분류된다.

2.2.4.2 정적 테스트

정적 테스트는 테스트 대상을 실행하지 않는 방식으로 테스트를 수행한다. 대표적인 방법으로는 리뷰(Review)와 정적 분석(Static analysis)이 있다.

2.2.4.2.1 리뷰

리뷰는 소프트웨어의 다양한 산출물에 존재하는 결함을 검출하거나 프로젝트의 진행 상황을 점검하기 위한 활동으로, 전문가 그룹이 수행한다. 리뷰를 효과적으로 수행하기 위하여 정해진 절차를 따르는데, 순서는 1) 경영진 준비, 2) 리뷰 계획, 3) 리뷰 절차 개요 설명, 4) 작업물 개요 설명, 5) 개별 준비, 6) 그룹 검토, 7) 재작업, 8) 후속 작업이다.

리뷰는 목적 및 구체적인 수행 방법에 따라 관리 리뷰, 기술 리뷰, 인스펙션(Inspection), 워크쓰루(Walk-through), 그리고 감사(Audit)로 구분된다. 표 2.4는 리뷰 유형에 대한 설명이다.

표 2.4 정적 테스트 – 리뷰 유형

리뷰 유형	설명
관리 리뷰	프로젝트 진행 상황에 대한 검토를 바탕으로 일정, 인력, 범위 등에 대한 통제 및 의사결정을 지원한다.
기술 리뷰	정의된 계획 및 명세를 준수하고 있는지에 대한 검토를 수행한다.
인스펙션	문제(Anomaly)를 식별하고 문제에 대한 올바른 해결(Resolution)을 검증한다.
워크쓰루	문제를 식별하고 더 나아가서 대안 조사, 개선 활동, 학습 기회 제공을 수행한다.
감사	객관적인 표준과 규제에 대한 준수를 독립적으로 평가한다.

2.2.4.2.2 정적 분석

정적 분석은 산출물(주로 소스 코드)의 구조적 속성을 이용하여 자동화된 방식으로 도구에 의해서 수행된다. 표 2.5는 대표적인 정적 분석 방법에 대한 설명이다.

표 2.5 정적 테스트 – 정적 분석 유형

리뷰 유형	설명
코딩 표준	MISRA-C, MISRA-C++ 등의 코딩 표준에 대한 준수 여부 검사
복잡도 측정	사이클로매틱 복잡도 등, 프로그램의 복잡도 측정
자료 흐름 분석	프로그램 자료 흐름에 이상(Anomaly) 존재 여부 분석

2.2.4.3 동적 테스트

동적 테스트는 테스트 대상을 실행하여 결함을 검출하는 방법으로, 적절한 입력값, 즉, 테스트 케이스를 결정해야 한다. 이해를 돕기 위해 자동차안전기준에 정의된 차량 정면충돌 시험을 예시로 살펴보자.

정면충돌 시험	자동차안전기준 제 102조
승용자동차의 경우, 시속 48.3km의 속도로 고정벽에 정면충돌시킬 때 운전 좌석 및 전방 탑승 좌석에 착석시킨 인체모형의 머리, 흉부, 대퇴부 등이 받는 충격이 아래 값을 초과하지 아니할 것 • 머리 상해 기준값(HIC): 1,000 • 흉부 가속도: 60g • 대퇴부 압축 하중: 1,020kg	

정면충돌 시험을 하는 경우, 탑승자에 미치는 충격은 충돌 시의 차량 속도에 따라 달라질 수 있다. 그러므로 충돌 시점의 차량 속도를 적절히 결정하고 시험을 진행해야 한다. 이때, 충돌 시점의 차량 속도가 테스트 케이스로 구성된다. 실제로 자동차안전연구원은 요구된 48.3km/h보다 빠른 56km/h 속도로 충돌 시험을 실시한다.

이렇게 테스트 케이스를 결정할 때, 소프트웨어의 어느 부분에 결함이 존재하는지 알 수 없으므로 가능한 한 많은 경우의 수를 조사하여 테스트 케이스를 결정해야 할 것 같지만, 소프트웨어는 매우 복잡해서 너무나 많은 상황이 있을 수 있으므로 모든 상황을 테스트하기 위한 테스트 케이스를 준비하기란 현실적으로 불가능하다. 그러므로 존재할 수 있는 결함을 누락하지 않으면서도 테스트 비용을 절감하려면 가능한 한 적은 수의 테스트 케이스를

사용하도록 테스트 케이스를 설계하는 것이 중요하다.

동적 테스트는 테스트를 통하여 확인하고자 하는 상황을 어떤 방법으로 결정하느냐에 따라 명세 기반 테스트, 구조 기반 테스트, 경험 기반 테스트로 분류된다.

2.2.4.3.1 명세 기반 테스트와 구조 기반 테스트

이 절에서는 동적 테스트 방법 중 명세 기반 테스트와 구조 기반 테스트에 대하여 설명한다. 그림 2.3은 명세 기반 테스트와 구조 기반 테스트의 개념을 보여준다.

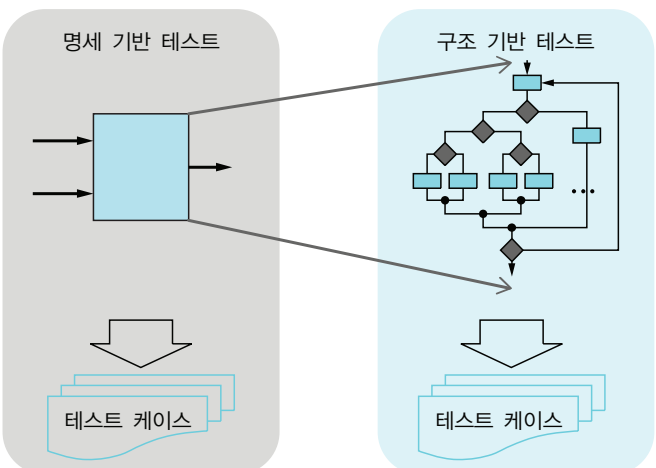

그림 2.3 명세 기반 테스트와 구조 기반 테스트 개념

명세 기반 테스트는 소스 코드를 참고하지 않고 테스트 케이스를 결정한다. 예를 들어 임의의 입력값을 생성하여 테스트하는 임의(Random) 테스트 방법도 소스 코드를 이용하지 않으므로 명세 기반 테스트라고 볼 수 있다.

반면에, 구조 기반 테스트는 구현된 소스 코드를 참고해서 테스트 케이스를 결정한다. 만약, 소스 코드의 특정 문장 또는 특정 경로(Path)를 실행하기 위한 입력값을 결정한다면 이 테스트는 구조 기반 테스트라고 볼 수 있다. 표 2.6은 명세 기반 테스트와 구조 기반 테스트를 수행할 때 사용되는 구체적인 테스트 설계 기법을 나열한 것이다.

표 2.6 명세 기반 테스트와 구조 기반 테스트 유형

명세 기반 테스트	구조 기반 테스트
동등 분할	문장 테스트
분류 트리 기법	결정 테스트
경곗값 분석	조건 테스트
신택스 테스트	결정/조건 테스트
조합 테스트	다중 조건 테스트
상태 전이 테스트	변형 조건/결정 테스트(MCDC)
인과 그래핑	기본 경로 테스트
결정표 테스트	
시나리오 테스트	

2.2.4.3.2 경험 기반 테스트

경험 기반 테스트는 기존의 테스트 경험, 테스트 대상이 되는 시스템 및 해당 도메인에 대한 경험 등을 바탕으로 수행하는 테스트 방법을 일컫는다.

2.2.4.3.2.1 오류 추정

오류 추정(Error guessing)은 개발자가 범할 수 있는 실수를 추정하고 이에 따른 결함이 검출되도록 테스트 케이스를 설계하는 방법이다. 즉, 특정 테스트 대상이 주어지면 테스터의 경험과 직관을 바탕으로 개발자가 범할 수 있는 실수들을 나열하고, 해당 실수에 따른 결함을 노출하는 테스트를 수행하는 것이다. 예를 들어, tan(x) 함수의 경우 ∞가 되는 90° 에 대한 값을 특별히 처리하지 않는 실수가 예상되므로 이 함수에 대한 테스트 입력에 90° 도를 포함시킬 수 있다.

오류 추정은 매우 직관적이고 상황에 따라 적합한 방식으로 수행되므로 일반화된 기법과 절차를 정의하기는 어렵다. 하지만 기본 아이디어는 발생할 수 있는 오류 상황을 나열하는 것에서 시작된다. 예를 들면, 일반적으로 특별하게 처리해야 하는 0 값을 미처리한 오류 상황이 발생할 수 있다. 그리고 가변적인 크기의 입력에서 특정 값을 검색하는 함수가 있을 때, '입력의 크기가 0인 경우'와 '일치하는 값을 발견하지 못한 경우'에 대한 오류 상황을 추정하고 이를 테스트할 수 있다.

오류 추정은 동등 분할이나 경곗값 분석 같은 명세 기반 테스트 방법과 함께 사용될 수 있다. 예를 들면, 동등 분할 방법에서 유효한 분할(Valid partition)은 테스트 대상에 대한 명세를 통하여 식별되지만 유효하지 않은 값의 영역은 명세의 정의가 명확하지 않으므로 테

스터의 경험과 직관을 활용하여 유효하지 않은 분할(Invalid partition)을 결정할 수 있다.

오류 추정의 가장 효과적인 방법은 오류가 검출된 대표적인 사례를 이용하는 것이다. 다음 예시는 정렬 프로그램을 테스트할 때 오류 추정을 통하여 식별된 테스트 상황이다.

- 입력 목록의 크기가 0일 때
- 입력 목록이 오직 하나의 데이터만 가지고 있는 경우
- 입력 목록이 모두 동일한 데이터를 가지고 있는 경우
- 입력 목록이 모두 정렬이 되어 있는 경우

특히, 오류 추정은 일반적으로 예상되지 않는 상황이 사용자 입력값으로 적절히 처리되고 있는지 확인할 때 유용하게 사용될 수 있다. 표 2.7은 오류 추정 기법에서 고려할 수 있는 대표적인 상황을 보여 준다.

표 2.7 **오류 추정 기법 예**

상황	설명
필수 입력	필수 입력 항목인 경우 값이 입력되지 않는 상황을 테스트한다.
입력 항목의 길이	입력 항목의 길이에 제약이 있는 경우, 더 작거나 더 긴 항목이 입력되는 상황을 테스트한다. 예 주민번호 앞자리는 6자리가 되어야 함
입력 항목의 형식	입력 항목에 대한 형식을 위반하는 상황을 테스트한다. 예 생년월일 형식: YY/MM/DD
입력값의 명시적 제약사항	입력 항목의 값에 대하여 명시적으로 정해진 범위를 위반하는 상황을 테스트한다. 예 YY: 00 - 99, MM: 01 - 12, DD: 01 - 31
입력값의 묵시적 제약사항	입력 항목의 값에 대하여 명시되지 않았지만, 상식적으로 가정되는 범위를 위반하는 상황을 테스트한다. 예 YY: 19 이하의 값; 금년이 2019년이므로

2.2.4.3.2.2 탐색적 테스트

탐색적 테스트는 사전에 구체적으로 테스트 케이스를 설계하여 기록하고 이를 바탕으로 테스트를 수행하는 방식이 아니라, 테스트 대상에 대한 이해, 테스트 케이스 설계, 테스트 실행을 병행하는 방식이다. 즉, 테스트 대상에 대한 이해를 바탕으로 즉석에서 테스트 케이스를 결정한 후, 문서화 없이 해당 테스트를 바로 수행한다. 그리고 이 테스트의 결과를 바탕으로 다음 테스트를 결정한다. 테스터는 자신이 원하는 방식으로 테스트를 수행하면서 테스트 대상이 어떻게 동작하는지, 어떤 기능이 동작하는지 파악해 나간다.

따라서 탐색적 테스트의 효과는 테스터의 지식에 의존한다. 예를 들어, 테스트를 수행하는 과정에서 파악하게 되는 테스트 대상에 대한 이해의 정도, 테스트 대상에 대한 사전 경험, 발생할 가능성이 있는 결함과 테스트 대상에 대한 위험 요소에 대한 이해 등이 중요한 영향을 미치게 된다.

앞서 얘기한 바와 같이 탐색적 테스트는 테스트를 위한 문서를 작성하지 않는다. 탐색적 테스트는 체계적인 이론과 방법을 바탕으로 테스트 설계를 수행하여 사전에 테스트 명세서(테스트 설계 명세서, 테스트 케이스 명세서, 테스트 절차 명세서)를 작성하는 대신, 테스터의 경험과 직관을 바탕으로 즉석에서 테스트 케이스를 결정하고 바로 수행한다. 심지어 테스트 계획서를 작성하지 않기도 한다. 그러나 테스트에 사용하는 자동화 도구를 통하여 테스트 로그 및 테스트 결과 등이 자동으로 생성될 수 있다.

테스트해야 하는 세부 피처를 명확하게 식별하고 의사소통하기 위한 테스트 차터(Charter)를 간략히 작성할 수 있다. 차터에는 테스트하고자 하는 세부 피처를 비롯하여 검출된 결함과 관련하여 추후 해결이 필요한 이슈 등을 기록한다.

탐색적 테스트는 애자일 방법을 사용하는 웹 응용 시스템의 테스트에 적합한 방법이다. 이러한 유형의 시스템은 개발 주기가 매우 짧기 때문에 세부적인 테스트 명세서를 개발하고 관리할 시간이 충분하지 않다.

탐색적 테스트를 수행할 때 명확한 가이드가 없으면 다음과 같은 단점이 있다. 첫째, 초기에 시스템에 대한 정보 및 이해가 부족하므로 결함을 검출하기 위하여 명확한 목표 없이 시스템의 이곳저곳을 탐색하는 데 과도한 시간을 사용할 위험이 있다. 둘째, 여러 명 또는 여러 팀이 탐색적 테스트에 참여할 경우에 동일한 기능을 중복해서 반복적으로 테스트할 위험이 있다. 또한, 테스트 범위를 명확하게 문서화하지 않고 테스트를 실시하므로 테스트의 적합성 즉, 커버리지(Coverage)에 대한 판단을 할 수 없다.

2.3 테스팅 방법

2.3.1 개요

지금까지 테스트에 대한 분류로서 테스트 레벨에 따른 분류, 테스트 유형에 따른 분류, 그리고 테스트 설계 기법에 따른 분류를 설명하였다. 본 절에서는 프로젝트의 다양한 현실적인 상황을 고려하여 실제 적용할 수 있는 대표적인 테스트 수행 방법을 소개한다.

2.3.2 리그레션 테스트

유지보수 단계에서는 다양한 이유로 소프트웨어 변경이 발생한다. 예를 들어, 사용자가 소프트웨어를 사용하는 과정에서 발견한 결함이 보고되는 경우, 이를 제거하기 위해 소프트웨어를 수정하게 된다. 또한, 기능을 추가하거나 성능을 개선하기 위하여 소프트웨어를 수정할 수도 있고, 새로운 운영 환경에 적응하기 위하여 수정하기도 한다. 또한, 더 높은 수준의 유지보수성을 확보하기 위하여 소프트웨어를 수정할 때도 있다.

이러한 이유로 소프트웨어가 변경되었다면 리그레션 테스트(Regression Test)를 수행해야 한다. 리그레션 테스트는 변경 후에 수행되는 테스트로, 소프트웨어에 가해진 변경이 의도하지 않게 결함을 만들지는 않았는지, 시스템이 기존의 요구사항을 충족하는지 검증하기 위하여 수행된다.

리그레션 테스트에 다양한 테스트 전략이 적용될 수 있는데, 대표적인 방법으로는 Retest-all 전략, 선택적 리그레션 테스트 전략, 테스트 최소화 전략, 그리고 테스트 우선 순위화 전략이 있다.

그리고 소프트웨어가 변경되면 각 레벨 테스트 순서대로 즉, 컴포넌트 테스트, 통합 테스트, 시스템 테스트 각 레벨마다 리그레션 테스트를 수행한다.

2.3.3 소프트웨어 생명 주기 모델과 테스트

소프트웨어 생명 주기는 소프트웨어 개발 체계의 추상적 표현이며 순차적 또는 병렬적인 일련의 단계로 구성된다. 예를 들어, 요구사항을 수집하고 문제를 이해・분석하는 단계부터 시작하여 설계 단계 및 시스템을 구성하는 모듈을 구현하는 단계를 거친다.

테스트는 이러한 소프트웨어 생명 주기 모델의 특성을 고려하여 수행되어야 한다. 예를 들어, 순차적 생명 주기 모델의 경우, 테스트는 구현이 완료된 시점에 1회만 수행되지만 진화형 모델과 애자일 모델과 같이 반복적이고 점진적인 모델에서는 테스트도 반복적・점진적으로 수행된다.

대표적인 순차적 생명 주기 모델인 폭포수 모델에서는 구현이 완료된 후에 테스트가 시작된다. 테스트 시작 시점이 구현이 완료된 후이므로 테스트를 수행할 때 요구사항 명세부터 구조 설계 및 상세 설계 산출물을 비롯하여 소스 코드에 이르기까지 필요한 모든 자료가 존재한다는 이점이 있지만, 이때 검출된 결함을 해결하는 데 많은 비용과 시간이 소요될 가능

성이 크다. 예를 들어, 결함의 원인이 근본적인 구조 설계상의 문제인 경우라면 개발 종료 시점에서는 이를 해결할 수 있는 시간이 충분하지 않고, 많은 비용이 소요될 것이다.

또 다른 순차적 개발 모델인 V-모델은 폭포수 모델과 달리 개발 시작과 함께 테스트도 시작된다. 그리고 각 개발 단계에서 발생하는 결함을 검출할 수 있는 테스트 레벨이 존재한다. 요구분석 단계에서는 시스템 테스트를 위한 테스트 계획 설계 및 테스트 케이스와 절차를 도출하고, 구조 설계 단계에서는 통합테스트 테스트 계획 수립 및 테스트 케이스를 도출한다. 그리고 V-모델에서는 동적 테스트와 더불어 개발 산출물에 대한 정적 테스트도 수행한다. 즉, 요구사항 명세서, 구조 설계 명세서, 상세 설계 명세서, 그리고 소스 코드 등에 대한 리뷰·정적 분석을 수행하여 결함을 검출한다.

진화적 개발 모델은 이터레이션(Iteration)과 점진적인Incremental) 방식으로 개발을 진행한다. 즉, 이 개발 모델은 시스템의 구성요소 중 핵심 부분을 개발한 후, 각 구성 요소와 추가 요구사항을 여러 이터레이션을 통해 개선하고 발전시켜 최종 완성품을 개발한다. 진화적 개발 모델은 수행하는 각 이터레이션마다 테스트 수행 계획을 작성하며 이에 따라 테스트를 수행한다. 각 이터레이션은 순차적 모델처럼 요구사항 분석, 설계, 구현, 테스트와 같은 단계로 구성되며 각 개발 단계에서 테스트 관련 프로세스가 수행된다.

애자일 개발 방법론은 진화적 개발 모델처럼 반복적이면서 점진적인 개발 접근 방식을 따른다. 애자일 방법론은 일반적으로 테스트 주도 개발(Test-Driven Development, TDD)을 적용한다. TDD는 프로그램에 대한 테스트 케이스를 먼저 작성하고, 이 테스트 케이스를 통과하는 실제 프로그램의 코드를 나중에 작성하는 방식이다. 그리고 TDD와 더불어 애자일 개발에서 중요한 실천 규칙인 지속적 통합(CI, Continuous Integration)이 있다. 말 그대로 지속적 통합은 통합이 어느 한 시점에 이루어지는 것이 아니라 계속해서 통합을 수행하는 것을 말한다.

2.3.4 위험 기반 테스트

테스트는 소프트웨어 프로젝트와 마찬가지로 주어진 비용과 일정 내에서 수행되어야 한다. 따라서 주어진 자원이라는 제약 내에서 테스트 목적을 달성하기 위한 노력이 필요하다.

테스트 접근 방법뿐만 아니라 테스트 범위를 결정할 때에도 주어진 비용과 일정을 고려해야 한다. 즉, 테스트 비용의 범위 내에서 단위 테스트, 통합 테스트 및 시스템 테스트의 수행 수준이 결정된다. 또한, 단위 테스트가 수행될 대상이 되는 모듈을 결정하고, 시스템 테스트를 수행할 때 테스트 될 기능 및 비기능적 특성 즉, 피처도 한정할 수 있다.

테스트 대상을 결정하고 선택된 테스트 대상에 대한 테스트 범위를 전체 시스템의 일부로 한정시키는 것은 테스트 비용을 줄일 수는 있지만 테스트 되지 않은 부분에서 결함이 발생할 가능성이 커지고 이는 결국 소프트웨어 품질에 악영향을 미칠 수 있다.

따라서 테스트 대상과 범위를 결정할 때는 테스트 미수행에 따른 위험을 고려해야 한다. 즉, 테스트가 수행되지 않았다고 하더라도 그에 따른 위험 수준이 낮은 것들은 테스트 대상에서 제외할 수 있지만, 테스트 제외에 따른 위험 수준이 높은 것은 테스트 대상에 반드시 포함해야 한다. 이와 같이, 피처에 대한 위험 분석을 바탕으로 테스트에 대한 계획과 설계 그리고 실행 등의 활동을 수행하는 것을 위험 기반 테스트라고 한다.

2.3.5 모델 기반 테스트

모든 명세 기반 테스트는 테스트 대상에 대한 기대 동작을 표현하는 일종의 모델을 이용한다. 이러한 모델은 자연어로 표현된 형식이거나 상태 전이도 또는 UML 다이어그램과 같은 시각적인 표현방식이 될 수 있으며 의사결정표와 같은 표 형태도 가능하다.

기존의 테스트는 모델을 이용하더라도 보통은 수작업으로 테스트 입력 및 출력을 결정하는 방식만을 제공하였다. 반면에, 모델 기반 테스트는 테스트 절차를 수행할 수 있는 정보가 자동으로 추출될 수 있을 정도로 정형화되고 상세한 모델을 바탕으로 한다. 즉, 모델을 바탕으로 테스트 계획을 수립하고, 테스트 케이스, 테스트 절차, 테스트 입력 및 예상 결과 등을 결정한다.

모델 기반 테스트 수행은 테스트 계획에서 테스트 종료까지 대부분의 활동을 자동화할 수 있다는 이점이 있다. 게다가 모델 자체에 존재할 수 있는 다양한 유형의 문제를 이른 시점에 식별하는 방법으로 개발 단계 산출물의 결함을 검출할 수 있다는 이점도 있다. 이러한 이유로, 모델 기반 테스트는 장애가 발생했을 때 많은 비용이 유발되는 자동차, 의료 등 안전 필수(Safety critical) 소프트웨어를 대상으로 수행되고 있다.

모델 기반 테스트는 일단 모델이 구축된 후에는 자동화를 통해서 효율적인 테스트를 수행할 수 있다는 이점이 있지만, 모델 기반 테스트의 근간이 되는 모델, 즉, 정형적이고 상세한 테스트 모델을 구축하는 비용이 추가되는 단점이 있다.

테스트 대상의 동작에 대한 상세한 모델링에는 의사결정표, UML 상태 다이어그램, UML 액티비티 다이어그램을 비롯한 정형적 표현법을 사용할 수 있다.