Certified Tester

Basic Level Syllabus

Version 2016

한글

(사단법인)케이에스티큐비

Korean Software Testing Qualifications Board

목 차

제1장 테스팅 개요(K2) — 50분

학습 목표

LO-1.1.1 소프트웨어 결함이 사람, 환경 또는 기업에 어떤 악영향을 미치는지 기억할 수 있다 (K1)

LO-1.2.1 테스팅의 일반적인 목적을 기억할 수 있다 (K1)

LO-1.3.1 오류, 결함, 장애 용어를 이해하고 상호 관계를 설명할 수 있다 (K2)

LO-1.4.1 테스팅에 영향을 주는 심리적인 요인을 기억할 수 있다 (K1)

LO-1.4.2 독립적 테스팅의 중요성을 기억할 수 있다 (K1)

LO-1.5.1 테스터가 갖춰야 할 윤리적 태도 및 의식에 대해 설명할 수 있다 (K2)

주요 용어

소프트웨어, 리뷰, 정적 분석, 동적 테스팅, 정적 테스팅, 개발 프로세스, 테스팅 프로세스, 테스트 베이시스, 테스팅 목적, 개발 테스팅, 품질 평가, 이해관계자, 운영 테스팅, 결함, 에러, 장애, 실수, 품질, 디버깅, 요구사항, 테스팅, 테스트 대상, 테스팅 독립성, 윤리 의식

1.1 테스팅의 필요성 (K1)

소프트웨어는 비즈니스 응용 프로그램(예: 은행 업무)에서 소비재(예: 자동차)까지 생활의 많은 부분에서 사용되고 있다. 하지만 누구나 한번쯤 소프트웨어 시스템을 사용하면서 소프트웨어가 기대와 다르게 동작하는 경우를 경험했을 것이다. 정상적으로 동작하지 않는 소프트웨어는 금전 및 시간적 손실, 비즈니스 이미지 손상은 물론 나아가 부상이나 사망 등 심각하고 다양한 문제를 야기할 수 있다.

1.2 테스팅이란 무엇인가? (K1)

대체로 테스팅을 소프트웨어가 동작하면서 실행하는 동적 테스팅으로 한정하는 경우가 많다. 동적 테스팅은 테스팅의 일부일 뿐이며 테스팅은 더 광범위하고 전문적인 활동을 일컫는다.

테스팅은 테스트 계획과 제어, 테스트 케이스 설계와 실행, 결과 리뷰, 테스트 완료 조건 평가, 테스트 프로세스 및 테스팅

결과 보고, 테스트 마감 활동 등 다양한 활동을 포함한다. 테스팅은 문서(소스코드 포함) 리뷰와 정적 분석을 포함한다. 테스트 대상 시스템 실행(구동) 여부에 따라 동적 테스팅 (Dynamic testing)과 정적 테스팅 (Static testing)으로 구분한다.

또한 테스팅은 개발 프로세스와 테스팅 프로세스를 개선하기 위한 정보를 제공한다.

테스팅의 목적은 다음과 같다:

- 결함 발견
- 품질 수준에 대한 자신감 획득
- 의사결정에 필요한 정보 제공
- 결함 예방

개발 수명주기 초기에 진행하는 테스트 계획 수립 및 테스트 설계를 통한 테스트 베이시스 검증 활동은 결함 유입을 방지하는 데 도움이 된다. 요구사항 등 문서 리뷰와 이슈 식별 및 해결은 코드에 결함이 유입되는 것을 사전에 방지하도록 도와준다.

테스팅 관점이 다양한 이유는 테스팅의 목적이 다양하기 때문이다. 예를 들면 개발 테스팅(예: 단위, 통합, 시스템 테스팅)의 주요 목적은 소프트웨어의 결함을 발견하고 수정하기 위해 가능한 많은 장애 상황을 만들어내는 것이다. 인수 테스팅의 주요 목적은 요구사항 대로 시스템이 동작하는지 확인하고 인수 여부에 대한 확신을 얻는 데 있다.

결함 수정과 상관없이 단지 소프트웨어 품질을 평가하기 위한 테스팅은 시스템 출시 전에 리스크 정보를 프로젝트 이해관계자에게 전달하는 것이 목적일 수 있다. 유지보수 테스팅은 소프트웨어 수정 및 신규 개발 과정에서 유입되는 새로운 결함을 확인하는 리그레션 테스팅을 포함한다. 운영 테스팅은 신뢰성 또는 가용성 등 시스템 특성 평가가 목적이 될 수 있다.

테스팅은 디버깅과 다르다. 동적 테스팅은 결함으로 인한 장애를 발견하는 반면 디버깅은 그 장애를 분석하여 원인을 발견하고 제거하는 개발 활동이다. 테스터는 발견된 장애가 제대로 수정되었는지 재테스팅한다. 일반적으로 테스터는 테스팅을, 개발자는 디버그를 수행한다.

1.3 테스팅의 원리 (K2)

프로그램 코드 또는 문서를 작성하면서 사람은 실수 (Error, 에러)를 하고 그 실수가 결함 (Defect, 결점, 버그)을 만든다. 코드에 잠재된 결함이 실행되면 시스템은 의도와 다르게 동작하는데, 이를 장애 (Failure)라 한다. 소프트웨어, 시스템, 문서의 결함은 장애의 원인이 되지만, 모든 결함이 장애를 일으키진 않는다.

시간적 압박, 복잡한 코드, 환경의 복잡성, 기술 변화, 빈번한 요구사항 변경, 여러 다른 시스템 간 상호 연동 등 다양한 조건하에서 사람의 실수로 결함이 유입된다.

장애는 결함 외에도 다양한 환경적 조건에 의해서도 발생한다. 예를 들면, 방사선, 자기, 전자기장, 물리적 오염 또한 소프트웨어 결함을 유발할 수 있으며, 이러한 환경적 조건이 하드웨어 조건을 변경시켜 소프트웨어 실행에 영향을 미칠 수 있다.

1.4 테스트 심리와 독립성 (K1)

1.4.1 테스트 심리

테스팅과 리뷰는 소프트웨어 개발의 관점과 다르다. 개발자가 자신이 개발한 코드를 직접 테스트할 수 있지만 이 역할을 테스터에게 맡기는 것은 개발자가 개발 활동에 집중하도록 도와주기 위해서다. 또한 테스트 전문 조직이 독립적인 시각으로 테스팅을 한다는 추가적인 이점을 얻을 수 있다. 모든 테스트 단계(레벨)에서 독립적인 테스팅이 가능하다.

독립적인 테스터가 결함과 장애를 찾는 데 더 효과적이긴 하나 개발자는 자신이 작성한 코드에서 결함을 효율적으로 찾을 수 있다.

사람이나 프로젝트는 목표가 있으며 그 목표를 지향한다. 테스터도 자신의 계획과 활동을 테스트 관리자나 프로젝트 이해관계자가 설정한 테스트 목표에 부합되도록 한다(예: 결함 발견을 목표로 할지 아니면 소프트웨어가 목적에 맞게 동작하는지 확인하는 것을 목표로 할지 등). 따라서 테스팅 목표를 명확하게 정하는 것은 매우 중요하다.

테스팅을 통해 장애를 발견하는 것이 테스트 대상 제품이나 개발자에 대한 비판으로 오해 받을 수 있다. 테스팅이 제품 리스크 관리 측면에서 매우 건설적이고 긍정적인 활동임에도 불구하고 종종 파괴적이고 부정적인 활동으로 인식되기도

한다. 시스템이나 소프트웨어에서 효율적으로 장애를 찾기 위해서는 호기심, 전문성, 비판적 시선, 세심한 주의력, 개발자 및 개발팀 동료와의 원활한 의사소통 그리고 결함 유추에 필요한 다양한 경험 등이 필요하다.

오류나 결함, 장애에 대해 긍정적인 방법으로 공유한다면 테스터와 개발자, 분석가, 설계자 간에 발생할 수 있는 감정 악화를 피할 수 있다. 이것은 테스팅뿐만 아니라 리뷰에서도 마찬가지다.

테스터가 결함을 지적하고 보고하는 등의 불쾌한 소식만을 전하는 사람으로만 인식되면 긍정적인 의사소통 형성이 어려워진다. 다음은 테스터와 이해관계자 간의 의사소통 및 관계 개선을 위한 몇 가지 방법이다.

- 다툼보다는 협력으로 시작하라 – 더 나은 품질의 시스템 구현이라는 공통 목표를 공유하는 공동체이다
- 개발자에 대한 비판 대신 중립적이고 객관적인 태도로 결함만을 전달한다. 객관적이고 사실에 근거한 인시던트 리포트를 작성한다
- 상대방이 어떻게 느끼는지 또는, 왜 그렇게 반응하는지 이해하도록 노력한다
- 전달한 내용을 상대방이 정확하게 이해했는지 확인한다

1.4.2 테스트 독립성

독립적인 테스터를 활용하면 테스팅과 리뷰로 결함을 발견하는 효과성을 높일 수 있다. 독립성 수준은 다음과 같이 분류할 수 있다.

- 독립성이 없음. 즉 개발자가 자신의 코드를 직접 테스트
- 개발팀 내 독립적 테스터(개발자와 다른 인원)
- 프로젝트 관리자 혹은 상위 관리자에게 직접 보고하는 조직 내 독립적 테스팅팀 또는 그룹
- 비즈니스 조직 또는 사용자 커뮤니티에서 채용한 독립적 테스터
- 사용성, 보안성, 인증 테스터(소프트웨어 또는 시스템이 표준 및 규정을 준수하는지 입증하는 전문가)처럼 특정 테스트 분야의 독립적 테스트 전문가
- 아웃소싱(Out-sourcing) 또는 조직 외부의 독립적 테스터

일부 또는 모든 테스트 단계(레벨)를 독립적인 테스터가 수행한다. 개발자는 하위 레벨 테스팅(단위, 통합 테스팅)에 참여하는 경우가 많다.

테스트 독립성(테스팅 조직을 독립적으로 운영하는 것)의 장점은 다음과 같다.

- 독립적 테스터는 한쪽에 치우치지 않으며 개발자와는 다른 시각에서 다른 종류의 결함을 발견한다
- 독립적 테스터는 시스템 설계 및 구현 단계에서 개발 담당자가 정의한 모델을 검증할 수 있다

테스팅 독립성의 단점은 다음과 같다.

- 개발팀 및 개발/제품 관련 정보로부터 고립될 수 있다
- 개발자가 품질에 대한 책임감을 잃을 수 있다
- 독립적인 테스터는 병목 현상 또는 출시 지연에 대해 비난을 받거나 책임에 자유로울 수 없다

테스팅은 테스터뿐만 아니라 프로젝트 관리자, 품질 관리자, 비즈니스와 해당 분야 전문가, 인프라 또는 IT 운영자 등 다양한 이해관계자가 수행할 수 있다.

1.5 테스팅과 윤리의식 (K2)

소프트웨어 테스팅에 참여하는 테스터는 기밀 정보 및 특정 정보들을 접하는 경우가 있다. 윤리 강령은 이러한 정보를 부적절하게 사용하지 않도록 보장하기 위해 필요하다. 엔지니어를 위한 ACM과 IEEE의 윤리 강령에 따라 ISTQB 및 KSTQB는 다음과 같은 윤리 강령을 제시한다.

- 공공성 – 소프트웨어 테스터는 공공의 이익에 부합하도록 행동한다
- 고객과 고용주 – 소프트웨어 테스터는 공공의 이익을 근거로 고객과 고용주의 이익에 부합하도록 행동한다
- 결과물 – 소프트웨어 테스터는 전문성을 갖춘 결과물을 작성하고 제시한다
- 판정 – 소프트웨어 테스터는 진실되고 독립성을 유지한 상태에서 결과를 판정한다
- 관리 – 소프트웨어 테스트 관리자 및 리더는 윤리적으로 테스팅하도록 지원하고 장려한다
- 직업 의식 – 소프트웨어 테스터는 공공의 이익에 부합하는 전문 직업인의 위상과 평판(명성)을 유지하고 선도한다
- 동료 의식 – 소프트웨어 테스터는 동료에게 공정하며 협조적이고 소프트웨어 개발자와 협력을 도모한다
- 자기 계발 – 소프트웨어 테스터는 끊임없이 학습하여 전문성을 유지하며 윤리적 접근 방식을 유지한다

제2장 소프트웨어 테스팅 기초 (K2) — 60분

학습 목표

LO-2.1.1 테스트 계획부터 테스트 마감 활동 등 5 가지 기본 테스트 활동과 각 테스트 활동을 기억할 수 있다 (K1)

LO-2.2.1 품질 리스크의 개념을 이해하고, 품질 리스크를 테스트에 적용하는 이유를 설명할 수 있다 (K2)

LO-2.3.1 테스트 단계(레벨)의 특징을 기억할 수 있다(목적, 대상, 목표, 베이시스, 주체 등) (K1)

LO-2.3.2 확인 테스팅과 리그레션 테스팅을 구분하고 목적을 기억할 수 있다 (K1)

LO-2.4.1 테스트 리더와 테스터의 업무를 기억할 수 있다 (K1)

주요 용어

테스트 계획, 테스트 제어, 테스트 분석, 테스트 설계, 테스트 구현, 테스트 실행, 완료 조건 평가 및 보고, 테스트 마감, 테스트 컨디션, 테스트 케이스, 테스트 베이시스, 테스트 아이템, 테스트 케이스 설계, 우선순위, 테스트 데이터, 테스트 환경, 테스트 도구, 추적성, 테스트 프로시저, 테스트 스크립트, 테스트 스위트, 인시던트, 테스트 로그, 테스트웨어, 테스팅 교훈, 테스팅 성숙도 개선, 품질 리스크, 프로젝트 리스크, 영향, 테스팅 전략, 리스크 기반 테스팅 전략, 리스크 수준, 테스트 단계(레벨), 단위 테스팅, 통합 테스팅, 시스템 테스팅, 인수 테스팅, 사용자 인수 테스팅, 운영자 인수 테스팅, 계약 인수 테스팅, 알파 테스팅, 베타 테스팅, 재테스팅, 리그레션 테스팅, 테스트 자동화, 테스트 리더, 테스터, 테스트 용이성

2.1 테스트 프로세스 (K1)

전체 테스팅 활동 중 시각적으로 가장 드러나는 부분은 테스트 실행 활동이다. 그러나 테스팅을 효율적이고 효과적으로 실행하기 위해서는 테스팅을 계획하고, 테스트 케이스를 설계하고, 테스팅 실행을 준비하고, 테스팅 진행 상태를 모니터링하고 평가하는 활동 등이 필요하다.

주요 테스트 활동은 다음과 같다.

- 테스트 계획과 제어
- 테스트 분석과 설계
- 테스트 구현과 실행
- 완료 조건의 평가와 보고

- 테스트 마감 활동

테스트 활동들을 순차적으로 진행해야 하지만 반복하거나 여러 활동을 동시에 수행할 수도 있다. 따라서 시스템과 프로젝트의 정황에 따라 테스팅 활동을 조정하는 것이 필요하다.

2.1.1 테스트 계획과 제어

테스트 계획 수립은 프로젝트의 목표와 임무 달성에 필요한 테스팅 목표와 테스트 활동을 명시하고 정의하는 활동이다.

테스트 제어는 계획 대비 실제 진행 상황을 비교하는 지속적인 활동으로 계획과의 차이와 진행 상태 보고를 포함한다. 테스트 프로젝트의 목표 및 임무를 달성하기 위해 필요한 조치를 취하는 것도 테스트 제어에 해당한다. 테스팅을 제어하기 위해서는 테스팅 활동을 모니터링해야 한다. 테스팅의 제어와 모니터링 활동을 통해 수집한 정보와 피드백을 반영해 테스트 계획을 수정한다.

2.1.2 테스트 분석과 설계

테스트 분석과 설계는 테스팅 목적을 테스트 컨디션 (Test Condition)과 테스트 케이스 (Test Case)로 명확하고 구체적으로 변환하는 활동이다.

테스트 분석과 설계는 다음과 같은 주요 활동을 포함한다.

- 테스트 베이시스 리뷰(요구사항, 개발 설계 명세, 인터페이스 (Interface) 명세 등)
- 테스트 베이시스와 테스트 대상의 테스트 용이성 평가
- 테스트 아이템, 요구 명세, 동작 및 소프트웨어 구조 등을 분석해 테스트 컨디션을 식별하고 우선순위 결정
- 개요 수준의 테스트 케이스 설계와 우선순위 결정
- 테스트 컨디션과 테스트 케이스에 필요한 테스트 데이터 식별
- 테스트 환경 설계 및 필요한 기반 환경 및 도구 식별
- 테스트 베이시스와 테스트 케이스 간 양방향 추적 정보 작성

2.1.3 테스트 구현과 실행

테스트 케이스를 특정 순서에 따라 결합해 테스트 프로시저 (Procedure) 또는 테스트 스크립트 (Script)를 작성하고, 테스트

실행에 필요한 다양한 정보를 수집하여 테스트 환경을 구축해 테스트를 실행하는 활동이 테스트 구현과 실행이다.

테스트 구현과 실행은 다음과 같은 주요 활동을 포함한다.

- 테스트 케이스 설계 마감, 테스트 케이스 구현 및 우선순위 선정(테스트 데이터 식별 포함)
- 테스트 프로시저 개발과 우선순위 선정, 테스트 데이터의 생성, 필요 시 자동화 테스트 스크립트 작성
- 효율적인 테스트 실행을 위해 테스트 프로시저와 테스트 스위트 (Test Suite, 테스트 케이스 묶음) 생성
- 테스트 환경이 올바르게 구축되었는지 확인
- 테스트 베이시스와 테스트 케이스 간 양방향 추적성 확인 및 업데이트
- 계획된 순서에 따라 수동 또는 테스트 실행 도구를 사용해 테스트 프로시저 실행
- 테스트 실행 결과 기록, 테스트 대상 소프트웨어, 테스트 도구, 테스트웨어의 버전과 정보 기록
- 예상 결과와 실제 결과 비교
- 예상 결과와 실제 결과 사이의 불일치를 인시던트로 보고, 그 원인을 분석하도록 지원(예: 코드 결함, 특정 테스트 데이터로 인한 결함, 테스트 문서에 의한 결함 또는 테스트 실행 중 발생한 결함 등)
- 인시던트 별 수정 조치에 따라 필요한 테스트 활동 반복. 예를 들어, 결함 수정 확인(확인 테스팅), 수정으로 인해 소프트웨어에 다른 결함이 유입되지 않았는지 확인(리그레션 테스팅)

2.1.4 완료 조건의 평가와 보고

완료 조건 평가는 프로젝트 초기에 정의한 테스트 목표를 달성했는지 확인(평가)하는 활동이며, 모든 테스트 단계(레벨)에서 수행한다.

완료 조건의 평가는 다음과 같은 주요 활동을 포함한다.

- 테스트 실행 로그 (Test logs)가 테스트 계획에 명시된 완료 조건을 만족하는지 확인
- 추가적인 테스트가 필요한지, 아니면 정의한 테스트 완료 조건을 변경해야 하는지 평가
- 프로젝트 이해관계자에게 배포할 테스트 요약 보고서 작성

2.1.5 테스트 마감 활동

테스트 마감 활동은 테스트 활동으로 생성된 데이터 즉, 테스트 결과와 수치적 데이터, 테스팅 경험과 테스트웨어 (Testware)를 수집하고 축적하는 활동이다. 테스트 마감 활동은 소프트웨어 시스템이 출시되거나 테스트 단계 또는 프로젝트가 완료될 때 등 테스트 활동이 일단락될 때마다 수행한다.

테스트 마감 활동은 다음과 같은 주요 활동을 포함한다.

- (계획된) 결과물 (상태) 확인
- 인시던트 리포트 마감 또는 미해결 항목의 변경 요청 리포트 작성
- 시스템 인수 문서화
- 향후 사용을 위해 테스트웨어, 테스트 환경, 테스트 기반 설비 보관
- 테스트웨어를 유지보수 조직에 인계
- 향후 출시나 프로젝트에 필요한 테스팅 교훈 분석
- 테스팅 성숙도 개선을 위해 수집한 정보 활용

2.2 리스크와 테스팅 (K2)

소프트웨어나 시스템에서 의도하지 않은 이벤트(동작)나 위험이 존재하는 잠재적인 장애 영역 (Potential failure areas)을 제품 리스크 (Product risks)라고 하며, 품질 리스크 (Quality Risk)라고도 한다. 시스템의 잠재적 장애 영역은 다음과 같다.

- 장애 가능성이 높은 (Failure-prone) 소프트웨어
- 개인이나 회사에 손실을 끼칠 가능성이 높은 소프트웨어와 하드웨어
- 품질이 낮은 소프트웨어의 특성 (Characteristics – 예: 기능성, 신뢰성, 사용성, 성능)
- 낮은 데이터 무결성 및 품질(예: 데이터 마이그레이션 이슈, 데이터 변환 이슈, 데이터 전송 이슈, 데이터 표준 위반)
- 정의된 대로 기능을 수행하지 못하는 소프트웨어

제품 리스크는 프로젝트 리스크 (Project Risk)와 구분한다. 리스크는 테스팅 시작 시점과 테스팅 강도를 결정하며, 테스팅은 의도하지 않은 결과를 야기시키는 리스크를 줄이거나, 의도하지 않은 결과로 인한 영향 (Impact)과 손실을 줄인다.

테스트 계획 단계에서 제품 리스크를 식별하여 심각한 제품 리스크를 줄일 수 있도록 테스팅을 계획한다면 테스팅을 효과적으로 수행할 수 있다. 일반적으로 시간과 인원이 부족한 상황에서 완벽하게 테스팅하길 기대하지만 한계가 있다. 이에 제품 리스크가 높은 영역이나 기능에 더 많은 시간과 인원을 할당하고, 리스크가 낮은 영역이나 기능에는 적은 시간과 리스크를 할당한다면 더 유의미한 테스팅을 할 수 있다. 이것이 리스크 기반 테스팅 전략이다.

리스크 기반 테스팅은 프로젝트 초기부터 리스크에 대처하므로 제품의 리스크 수준을 줄여준다. 리스크 기반 테스팅은 제품 리스크 식별, 리스크 분석 및 결과 등을 이용해 테스트 계획 및 제어, 테스트 설계, 테스트 구현 및 실행을 가이드 한다. 리스크 기반 테스팅 전략에서 리스크 수준에 따라 차별적으로 적용할 수 있는 항목은 다음과 같다.

- 테스트 기법
- 테스팅 추가 여부
- 테스트 실행 우선순위
- 기타 테스팅 관련 활동

2.3 테스트 단계(레벨) (K1)

테스트 단계(레벨)는 다음과 같은 특징을 가진다: 테스트 목표(목적), 테스트 케이스 도출에 필요한 개발 산출물(테스트 베이시스), 테스트 대상, 결함 및 장애 유형, 테스트 환경 및 도구, 테스트 전략, 수행 주체

2.3.1 단위 테스팅 (Component/Unit Testing)

테스트 베이시스:

- 단위 요구사항
- 상세 설계 명세
- 코드

일반적인 테스트 대상:

- 단위(컴포넌트)
- 프로그램

단위(모듈, 또는 프로그램) 테스팅은 테스트 가능한 최소 단위로 구분한 소프트웨어 모듈, 프로그램, 객체, 클래스 등에서 기능을 검증하고 결함을 찾는다.

단위 명세, 소프트웨어 상세 설계 또는 데이터 모델 명세 등 개발 산출물을 테스트 베이시스 (Test Basis)로 활용한다. 일반적으로 단위 테스팅은 개발 환경에서 코드를 중심으로 수행하며 단위 테스트 프레임워크 또는 디버깅 도구 같은 개발 환경의 지원이 필요하다. 일반적으로 코드를 개발한 개발자가 단위 테스팅을 수행하는 경우가 많으며 단위 테스팅 단계에서 발견한 결함은 즉시 수정하며 결함을 기록하고 관리하진 않는다.

2.3.2 통합 테스팅 (Integration Testing)

테스트 베이시스:

- 소프트웨어와 시스템 설계 명세
- 시스템 구조 명세
- 워크플로우 (Workflows)
- 유스케이스 (Use Case)

일반적인 테스트 대상:

- 서브 시스템(단위 프로그램들이 결합된 시스템)
- 인터페이스

통합 테스팅은 컴포넌트 간 인터페이스를 테스트하며 OS(운영체제), 파일 시스템, 하드웨어 또는 시스템 간 인터페이스와 같은 시스템의 각기 다른 부분과 상호 연동하는 동작을 테스트한다.

주로 개발 환경에서 아키텍처를 이해하는 개발자가 수행한다.

2.3.3 시스템 테스팅 (System Testing)

테스트 베이시스:

- 시스템 및 소프트웨어 요구사항 명세
- 유스케이스 (Use Case)
- 기능 명세

일반적인 테스트 대상:

- 시스템, 사용자 및 운영 매뉴얼
- 시스템 구성 및 구성 데이터

시스템 테스팅은 전체 시스템 또는 제품의 동작을 테스트한다. 시스템 테스팅은 실사용 환경 또는 이와 유사한 환경에서 수행하며 이를 통해, '환경 특성 장애 (Environment-specific failure)' 리스크를 최소화한다.

시스템 테스팅은 리스크, 요구사항 명세, 비즈니스 프로세스, 유스케이스 등 시스템의 기능을 설명한 개요 수준의 명세나 모델, OS 및 시스템 리소스 간 상호작용 등을 테스트한다.
시스템 테스팅은 기능 및 비기능 요구사항, 데이터 품질 특성 등을 모두 검증해야 한다.

일반적으로 시스템 테스팅은 독립적인 테스트팀이 수행한다.

2.3.4 인수 테스팅 (Acceptance Testing)

테스트 베이시스:

- 사용자 요구사항
- 시스템 요구사항
- 유스케이스 (Use Case)
- 비즈니스 프로세스

일반적 테스트 대상:

- 비즈니스 프로세스
- 운영 및 유지보수 프로세스
- 사용 절차
- 출력 양식(형식), 보고서

일반적으로 인수 테스팅은 시스템을 사용하는 고객이나 사용자가 수행하며 다른 관련자도 참여할 수 있다.

인수 테스팅은 시스템 또는 특정한 비기능적 특성에 대해 '확신'을 얻는 것이다. 결함을 찾는 것은 인수 테스팅의 주된 목적이 아니며, 시스템 배포 및 사용 가능 여부를 확인(평가)한다. 그러나 인수 테스팅이 반드시 최종 단계의 테스팅이 아닐 수 있다. 예를 들어, 개별적인 시스템 인수 테스팅 후 대규모 시스템 통합 테스팅을 실행하는 경우도 있다.

인수 테스팅은 모든 개발 단계에서 수행할 수 있다. 예를 들면

- 상용(COTS) 소프트웨어 제품을 설치하거나 통합시킨 후 인수 테스팅을 수행할 수 있다
- 신규 및 개선 기능에 대한 인수 테스팅을 시스템 테스팅 이전에 수행할 수 있다

인수 테스팅의 일반적인 형태는 다음과 같다.

사용자 인수 테스팅

일반적으로 비즈니스 사용자가 시스템 사용의 적절성을 확인한다

운영상의(인수) 테스팅

시스템 관리자에 의한 테스트 활동으로 일반적인 테스트 항목은 다음과 같다:

- 백업 / 복원 테스팅
- 재난 복구
- 유지보수 작업
- 보안 취약성에 대한 정기적인 점검

계약 인수 테스팅과 규정 인수 테스팅

맞춤식-개발 (Custom-developed) 소프트웨어가 계약상의 인수 조건을 만족하는지 확인하는 테스팅이다. 인수 조건은 계약서 작성 시 명시한다. 규정 인수 테스팅은 정부 지침, 법률 또는 안전 규정 등 준수해야 할 규정에 적합하게 개발되었는지 확인하는 테스팅이다.

알파 테스팅과 베타(또는 필드) 테스팅

마켓에서 판매되는 소프트웨어 또는 상용 (COTS) 소프트웨어 개발자는 종종 소프트웨어가 출시되기 전에 기존 고객이나 잠재 고객으로부터 피드백을 받을 필요가 있다. 알파 테스팅은 개발사가 수행하며 개발팀이 직접 수행하진 않는다. 반면 베타 테스팅 또는 필드 테스팅은 실제 환경에서 사용자 혹은 잠재 고객이 수행한다.

개발 완료 후 고객사로 이동하기 전에 테스팅하는 '공장 인수 테스팅 (Factory acceptance testing)'은 알파 테스팅과 같은 의미이며 고객 사이트에서 테스팅하는 '사이트 인수 테스팅 (Site acceptance testing)'은 베타 테스팅과 같은 의미다.

2.3.5 재테스팅/리그레션 테스팅 (Re-testing and Regression Testing)

결함 수정 후, 결함이 성공적으로 제거되었는지 확인하기 위해 다시 테스트하며 이것을 재테스팅 또는 확인 테스팅이라고 한다. 여기서 결함을 수정하는 디버깅(결함의 확인 및 수정)은 개발 활동이며 테스팅 활동이 아니다.

리그레션 테스팅은 이미 테스트된 프로그램의 테스팅을 반복하는 것으로 결함 수정과 변경으로 인해 새로운 결함이 유입되거나 이전에 발견하지 못한 결함을 발견하는 것이다. 이러한 결함은 테스트 중인 소프트웨어에 존재할 수도 있고 관련이 있거나 전혀 관련이 없는 다른 소프트웨어에 있을 수 있다. 리그레션 테스팅은 소프트웨어 또는 그 환경이 변경되었을 때 수행한다. 소프트웨어의 잠재 결함이 야기할 수 있는 리스크에 따라 리그레션 테스팅의 범위와 정도가 달라진다.

리그레션 테스팅은 모든 테스트 레벨에서 수행할 수 있으며, 리그레션 테스트 스위트는 여러 번 반복 수행하므로 리그레션 테스팅은 가장 유력한 자동화 대상이다.

2.4 테스팅의 역할 (K1)

테스팅의 역할은 크게 테스트 리더와 테스터로 나뉜다. 프로젝트, 제품 정황 (Context), 역할 수행 인원의 역량, 조직 특성 등에 따라 테스팅 역할의 활동과 임무는 달라질 수 있다.

테스터 리더는 테스트 관리자 또는 테스트 조정자 (Coordinator)라고도 한다. 프로젝트 관리자, 개발 관리자, 품질 보증 관리자 또는 테스트 그룹 관리자가 테스트 리더 역할을 수행한다. 테스트 리더는 테스팅 활동과 업무를 계획하고 모니터링하고 제어하는 역할을 수행한다. 규모가 큰 조직에서는 테스트 리더와 테스트 관리자를 구분하기도 한다.

테스트 리더의 전형적인 역할과 임무는 다음과 같다.

- 테스트 전략과 계획을 프로젝트 관리자 및 이해관계자와 합의 및 조정한다
- 프로젝트의 테스트 전략과 조직의 테스트 정책을 작성하고 검토한다
- 프로젝트 정황, 테스트 목적, 리스크를 이해하면서 테스트 계획을 수립한다
- 테스트 계획 단계에서 테스트 전략(접근법)을 선택하고, 테스팅 시간과 노력, 비용 등을 추정하고 테스팅 필요 자원(일정, 인원. 비용 등)을 획득하고 테스트 레벨 및 테스트 절차와 목표를 정의하고 인시던트 관리를 계획한다
- 테스트 설계, 준비, 구현, 실행, 테스트 결과 모니터링 및 완료 조건 평가를 주도한다
- 테스트 결과와 진척을 기반으로 계획 활동을 조정하고 문제점을 보완하기 위해 필요한 행동을 취한다
- 테스트 진척 및 테스팅 품질과 테스트 대상 제품의 품질을 평가하는 적절한 메트릭(지표)을 선정한다
- 자동화 대상, 범위, 방법 등을 결정한다
- 테스트 지원 도구를 선정하고 사용자 교육 및 훈련을 계획한다
- 테스트 환경 구축 사항을 결정한다
- 테스트 과정에서 수집한 정보를 근거로 테스트 결과 보고서를 작성한다

전형적인 테스터의 역할과 임무는 다음과 같다.

- 테스트 계획 및 테스트 전략을 리뷰한다
- 테스트 용이성 (Testability)을 평가하기 위해 요구사항, 명세, 모델을 리뷰한다
- 테스트 케이스, 테스트 프로시저를 작성한다
- 테스트 환경을 구축한다(시스템 및 네트워크 관리자와 협업)
- 테스트 데이터를 준비하거나 작성(획득) 한다
- 테스트 구현, 실행, 로그 기록 및 테스트 실행 결과 평가, 인시던트 보고를 수행한다
- 테스트 운영 및 관리 도구, 테스트 모니터링 도구를 사용한다

- 테스트 자동화를 구현한다(자동화 스크립트 작성 등)
- 작성한 테스트 케이스, 테스트 프로시저 등을 리뷰한다

테스트 레벨, 테스트 대상 제품, 프로젝트와 관련된 리스크에 따라 다양한 분야의 담당자가 일정 수준의 독립성을 유지하면서 테스터의 역할을 담당할 수 있다. 일반적으로 단위 테스트와 통합 테스트 레벨에서는 개발자가 테스터인 경우가 많으며, 시스템 레벨에서는 독립적인 테스트 조직이 테스터인 경우가 많고 인수 테스트 레벨에서는 비즈니스 전문가 또는 사용자가 테스터인 경우가 많다.

제3장 소프트웨어 테스팅 실무 (K3) 130분

학습 목표

LO-3.1.1 테스트 컨디션, 테스트 케이스, 테스트 스위트, 테스트 프로시저 등 용어를 비교할 수 있다 (K2)

LO-3.2.1 명세 기반, 구조 기반, 경험 기반 테스트 설계 기법의 특성과 공통점, 차이점을 설명할 수 있다 (K2)

LO-3.3.1 동등 분할, 경계값 분석, 시나리오를 이용해 테스트 케이스를 작성할 수 있다 (K3)

LO-3.4.1 직관, 경험을 근거로 테스트 케이스를 작성하는 이유를 기억할 수 있다 (K1)

LO-3.4.2 오류 추정, 탐색적 테스팅 등 경험 기반 기법의 특징과 장점을 설명할 수 있다 (K2)

LO-3.5.1 발견한 장애 현상을 인시던트 보고서로 작성할 수 있다 (K3)

주요 용어

테스트 베이시스, 테스트 컨디션, 테스트 설계, 명세서, 요구사항, 추적성, 커버리지, 테스트 강도, 테스트 설계 기법, 블랙박스 테스팅, 경험 기반 테스팅, 화이트박스 테스팅, 동등 분할, 경계값 분석, 시나리오 테스팅, 탐색적 테스팅, 오류 추정, 인시던트 관리, 인시던트 보고서, 인시던트 상태, 인시던트 관리 도구

3.1 테스트 케이스 구성 (K2)

테스트 분석 단계에서 테스트 베이시스 (Test basis)를 분석해 무엇을 테스트할지 결정하는데, 이는 테스트 컨디션(Condition)을 식별하는 과정이다. 테스트 컨디션은 테스트 할 항목 또는 이벤트, 기능, 처리 내용, 품질 특성(비기능 포함), 구조적 요소 등 테스트할 다양한 아이디어를 의미한다.

테스트 설계 단계는 식별한 테스트 컨디션에 적절한 테스트 설계 기법을 적용해 테스트 케이스와 테스트 데이터를 설계하고 명세(작성)화한다. 테스트 설계 시, 테스터가 아는 만큼 다양한 테스트 설계 기법을 적용할 수 있으며 완성도 높은 테스트 케이스를 작성할 수 있다.

테스트 케이스는 테스트 목적 또는 테스트 컨디션을 위해 작성한 입력값(들), 실행 사전 조건, 기대 결과, 실행 사후 조건 등으로 구성된다. 소프트웨어 테스트 표준 (Software Testing, ISO/IEC/IEEE 29119-3)에서 테스트 케이스 명세(양식)와 관련 정보를 정의하고 있다.

테스트 케이스의 기본 항목은 다음과 같다.

- 테스트 케이스 식별자 (Unique Identifier)
- 목적 또는 제목 (Objective)
- 우선순위 (Priority)
- 추적성 (Traceability)
- 실행 사전 조건 (Preconditions)
- 입력값 (Inputs)
- 기대 결과 (Expected Results)
- 실행(실제) 결과 (Actual Results and Test Result)

만일 테스트 케이스 기본 항목 중 기대 결과 항목을 정의하지 않았다면, 틀린 결과가 정상 결과로 판정될 수 있다. 이와 같이 테스트 실행 결과가 정상인지 판단할 수 없는 어려운 상황을 초래하지 않기 위해 테스트 실행 전에 기대 결과를 정의하는 것이 원칙이다.

테스트 케이스 기본 항목 중 추적성 항목은 소홀하기 쉬운 항목이지만 매우 중요하다. 명세서 또는 요구사항(정의서)에서 테스트 컨디션을 도출하여 테스트 케이스를 작성한다면 요구사항, 테스트 컨디션, 테스트 케이스 간의 관계를 추적할 수 있는 정보가 필요하다. 요구사항이 변경되면 어떤 테스트 케이스를 수정해야 할지 알 수 있으며, 또한 실행한 테스트 케이스가 어느 요구사항과 관련됐는지 파악할 수 있기 때문에 요구사항 커버리지 (Requirement coverage)를 측정할 수 있다.

테스트 분석 및 설계 시, 리스크에 따라 테스트 설계 기법을 선택할 수 있다. 리스크가 높은 요구사항은 강도가 높은 테스트 설계 기법을 적용하며, 반면 리스크가 낮은 요구사항은 강도가 낮은 테스트 설계 기법을 적용한다. (리스크 분석은 2.2장 참고, 테스트 설계 기법은 3.2장 참고)

테스트 분석과 설계 후, 테스트 케이스를 작성하는 것을 테스트 구현이라 한다. 테스트 실행에 필요한 모든 준비를 마무리하는 것이 테스트 구현 단계의 최종 목표다. 따라서 테스트 구현 단계에서는 테스트 케이스, 테스트 스위트, 테스트 프로시저 등을 작성한다. 그리고 환경 구축과 자동화 스크립트 작성 등 테스트 실행에 필요한 추가적인 활동도 진행한다.

대체로 테스트 케이스는 두 개 이상 복수로 작성하므로 테스트 케이스를 그룹으로 구성하여 관리한다. 테스트 케이스의 그룹 또는 묶음을 테스트 스위트 (Test Suite)라 하며, 기능, 화면, 대상 등 다양한 목적에 따라 테스트 스위트를 구성한다.

그리고 테스트 스위트 내 테스트 케이스들의 실행 순서를 정의한 것을 테스트 프로시저라 한다. 실행 순서가 정해진 테스트는 테스트 케이스들의 실행 순서가 결과에 영향을 주기 때문에 실행 순서를 반드시 명시해야 한다. 만약 테스트

실행 자동화 도구로 테스트 한다면 테스트 순서를 테스트 스크립트(자동화된 테스트 프로시저)로 작성한다. 소프트웨어 테스트 표준 (Software Testing, ISO/IEC/IEEE 29119-3)에서 테스트 프로시저 명세(양식)와 관련된 정보를 정의하고 있다.

3.2 테스트 설계 기법 (K2)

테스트 설계는 테스트 컨디션, 테스트 케이스, 테스트 데이터를 식별하고 명세화하는 것이다.

일반적으로 테스트 기법을 블랙박스와 화이트박스로 구분한다. 블랙박스 기법(명세 기반 기법과 경험 기반 기법 포함)은 테스트 대상의 내부 구조(코드) 대신 테스트 베이시스 그리고 개발자와 테스터, 사용자들의 경험을 바탕으로 기능 혹은 비기능 테스트 케이스를 도출한다. 반면 화이트박스 기법(구조 기반 기법)은 단위(컴포넌트) 또는 소프트웨어(시스템)의 구조(코드)를 중심으로 테스트 케이스를 도출한다. 블랙박스 테스트 기법과 화이트박스 테스트 기법에 개발자, 테스터, 사용자들의 경험을 반영한다면 테스트의 완성도를 더 높일 수 있다.

명세 기반 테스트 설계 기법의 일반적인 특징은 다음과 같다.

- 테스트 대상을 명세화한 공식 또는 비공식 모델 사용
- 정해진 절차와 원리를 이용해 체계적으로 테스트 케이스 도출 가능

구조 기반 테스트 설계 기법의 일반적인 특징은 다음과 같다.

- 소프트웨어 구현 및 구조 정보를 기반으로 테스트 케이스 도출(예: 코드, 개발 설계 정보)
- 작성한 테스트 케이스로 테스트 커버리지 측정 가능, 커버리지를 높이기 위해 체계적으로 테스트 케이스 추가 도출 가능

경험 기반 테스트 설계 기법의 일반적인 특징은 다음과 같다.

- 테스트 대상에 대한 테스터의 지식이나 경험을 기반으로 테스트 케이스 도출
- 테스트 대상에 대한 테스터, 개발자, 사용자 및 기타 이해관계자의 지식 활용, 소프트웨어 사용법 및 환경 관련 정보 포함
- 테스트 대상 소프트웨어 또는 시스템의 결함 분포 및 발생 빈도가 높은 결함 정보 활용

3.3 명세 기반 설계 기법 (K3)

3.3.1 동등 분할 (Equivalence Partitioning)

소프트웨어나 시스템의 입력값들 중 결과(값)가 동일한 입력값들을 하나의 그룹으로 간주한다. 이 그룹을 동등 분할 (Equivalence Partition) 클래스 (Class)라 하고, 동등 분할 클래스를 이용하는 테스트 기법이 동등 분할 테스팅이다. 동등 분할 테스팅은 동일한 그룹 내의 모든 입력값이 동일한 방식으로 처리된다고 가정하므로 동일한 그룹 내 하나의 입력값을 대표값으로 선택해 테스트한다. 동등 분할 클래스는 유효(정상) 입력과 비유효(비정상) 입력 범위에서 식별할 수 있다. 그리고 입력값 (Input) 외에도 출력값 (Outputs), 내부 처리값 (Internal values), 시간 관련값 (Time-related values, 이벤트 전/후), 인터페이스 파라미터 (Interface parameters)에 대해 동등 분할 클래스를 정의할 수 있다. 식별한 모든 유효/비유효 동등 분할 클래스를 포함하도록 테스트를 설계한다.

동등 분할 테스팅은 입력값 및 출력값의 동등 분할 클래스 커버리지를 측정할 수 있다. 또한 동등 분할 테스팅은 사용자에 의한 입력, 인터페이스를 통한 입력, 통합 테스팅 단계에서 인터페이스 파라미터 등에 적용할 수 있다. 동등 분할 테스팅은 명세 기반 기법 중 가장 많이 사용하며, 모든 테스트 단계에 적용할 수 있다.

3.3.2 경계값 분석(Boundary Value Analysis).

분할된 영역 즉, 동등 분할의 경계에 주변에서 결함이 발생할 확률이 높기 때문에 경계값을 중심으로 테스트 케이스를 설계하는 기법이 경계값 분석이다. 일반적으로 동등 분할 내에서 임의로 선택한 대표값보다는 경계값이 결함 발견에 효과적이다. 한 동등 분할 영역의 최대값과 최소값이 그 영역의 경계값이며 하나의 동등 분할 영역에는 두 개의 경계값이 있다.

하나의 동등 분할 영역에서 하나의 대표값을 선택하는 동등 분할 테스팅보다 두 경계값을 선택해 테스트하는 경계값 분석이 테스트 케이스 수가 많아 테스팅 시간이 더 필요하긴 하지만 대신 테스트 강도를 높일 수 있다.

경계값 분석은 동등 분할 테스팅의 테스팅 강도를 높이기 위해 사용하며 모든 테스트 단계에 적용할 수 있다. 명세가 명확할수록 경계값 구분이 쉬우며 경계값 분석은 경계값의 테스트 커버리지를 측정할 수 있다. 그리고 시간, 크기, 용량, 거리 등 연속적인 수나 범위 등을 테스트하기 적합하다.

3.3.3 시나리오 테스팅 (Scenario Testing)

사용자 시나리오 또는 업무 흐름(도)을 이용해 테스트 케이스를 작성할 수 있다. 사용자 시나리오 또는 업무 흐름(도)의 목적과 절차를 고려해 가능한 모든 흐름을 누락 없이 테스트 케이스로 작성한다.

사용자 시나리오 또는 업무 흐름(도)가 사용 방법을 대표하므로 시나리오 테스팅은 시스템 사용 절차와 흐름에서 결함을 발견하는 데 유용하다. 그리고 시나리오 테스트는 고객 또는 사용자 그룹이 참여하는 인수 테스팅에 적합하며, 단위 테스트에서 발견하기 어려운 단위 간 상호작용과 간섭으로 인한 복합 결함을 찾는 데 도움이 된다. 시나리오 테스트는 다른 명세 기반 테스팅 기법과 혼용해 사용할 수 있다.

3.4 경험 기반 기법 (K2)

3.4.1 경험 기반 테스트 개요

테스터가 경험한 유사 프로그램이나 기술, 경험, 직관 및 테스터의 기술 능력을 기반으로 테스트 케이스를 도출하는 것이 경험 기반 테스팅이다. 체계적인 기법 특히 명세 기반 기법과 구조 기반 기법 등 공식 기법 적용 후, 추가적으로 경험 기반 기법을 적용해 공식 기법으로 발견하기 어려운 특이한 결함을 찾는 데 유용하다. 그러나 이 경험 기반 기법은 테스터의 경험에 따라 효율성 및 효과성이 매우 달라지므로 테스트 관리자의 역량이 필요하다.

경험 기반 테스팅은 참고할 명세가 거의 없거나 불명확하고 부족한 경우, 시간이 부족한 경우, 또는 공식적인 테스팅을 강화/보완할 때 유용하다.

3.4.2 오류 추정과 탐색적 테스팅

가장 많이 사용하는 경험 기반 기법은 오류 추정 (Error guessing)이다. 테스터가 경험을 근거로 발생 가능성이 높은 결함을 중심으로 테스트한다. 오류 추정을 체계적으로 수행하려면 발생 가능한 모든 결함 (Defects)을 나열하고 모든 결함 또는 오류를 발견할 수 있도록 테스트를 설계한다. 이러한 체계적인 접근법을 결함 공격 (Fault attack)이라 한다. 경험이나 기존 장애 및 결함 데이터 또는 일반화된 소프트웨어 결함 지식을 활용해 결함이나 장애 목록을 작성한다.

경험 기반 테스팅 중 가장 체계적인 탐색적 테스팅 (Exploratory testing)은 테스트 설계와 실행, 기록 그리고 학습을 동시에

진행하는 테스트 접근법이다. 탐색적 테스팅은 테스트 목적 및 범위 등을 명시한 테스트 차터 (Test charter)를 기반으로 정해진 시간 (Session) 내에 테스트를 실행하며 실행 후 개인의 테스트 수행 경험과 주요 내용을 공유하는 회고 (Debriefing)를 한다. 탐색적 테스팅 실행 중 테스터는 테스트 실행 과정과 결과를 기록해 테스트 결과 리포트 및 회고에 사용한다. 일반적으로 탐색적 테스팅은 여러 세션을 반복해 테스트를 실행하며, 매 세션마다 차터 분배, 테스트 실행, 노트 작성, 회고 등을 수행한다.
테스트 실행 시간인 세션은 보통 60~90분 사이에서 테스트 대상 및 환경, 상황 등을 고려해 정한다. 세션 단위로 테스트를 실행함으로써 테스터의 집중도를 높일 수 있어 효과적인 테스트가 가능하다.

명세 기반 및 구조 기반 테스팅 등 체계적인 테스팅으로 심각한 결함을 발견했다는 것을 탐색적 테스팅으로 확신을 얻을 수 있다.

3.5 인시던트 관리 (K3)

테스팅의 주요 목적 중 한 가지는 결함 발견이므로 실제 결과와 기대 결과의 불일치를 식별하고 이를 기록해야 한다. 실제 결과와 기대 결과의 불일치를 인시던트 (Incident)라 하며 인시던트는 추가적 조치나 조사가 필요한 모든 상황을 의미한다. 따라서 인시던트는 결함은 물론 개선 요구사항을 포함한다.

인시던트는 해결하는 것이 목적이므로 인시던트 관리 절차와 원칙이 필요하다. 테스터가 인시던트를 발견하고 이를 개발자에게 전달해 인시던트를 수정하도록 유도하고, 인시던트가 수정(해결)되었는지 확인하여 인시던트 처리를 마감하는 모든 활동을 추적해야 한다. 따라서 인시던트를 효과적으로 관리하기 위해 인시던트 관리 절차와 인시던트 분류 규칙을 정의해야 한다.

인시던트는 프로그램 개발, 리뷰, 테스팅 또는 소프트웨어 사용 과정에서 발생 또는 발견할 수 있으며, 프로그램 코드, 운영 중인 시스템, 요구사항 문서, 개발 문서, 테스트 문서, 도움말이나 설치 가이드 등 모든 개발 및 테스트 결과물에서 발견된다.

인시던트 보고서는 다음과 같은 목적을 갖는다.

- 개발자와 프로젝트 관련자에게 문제 상황을 공유하고, 결함 수정에 필요한 정보 제공
- 테스트 리더가 시스템의 품질 또는 테스트 진척을 확인할 수 있는 정보 제공
- 테스트 프로세스 개선 (Test process improvement)에 대한 아이디어 제공

인시던트 보고서 작성 시 입력 및 관리하는 정보는 다음과 같다.

- 인시던트 보고서 식별자 (Incident Report Identifier)
- 요약 (Summary)
- 상세 설명 (Incident Description)
  - 입력 정보 (Input)
  - 기대 결과 (Expected Results)
  - 실제 결과 (Actual Results)
  - 장애 및 이상 현상 (Anomalies)
  - 발생 일자 (Date and Time)
  - 재현 절차 (Procedure Step)
  - 테스트 환경 (Environment)
  - 테스터 (Testers)
  - 참관자 (Observers)
- 인시던트 영향도 (Impact)
- 추적성 (Traceability)
- 인시던트 발견 시점 (Which process in Software Development Lifecycle)
- 인시던트 심각도 (Severity)
- 인시던트 수정 우선순위 (Priority)

인시던트 처리 시 추가적으로 관리하는 항목은 다음과 같다.

- 인시던트의 상태: 발견, 수정 대기, 수정 완료, 종료
- 인시던트에 대한 의견 및 처리 결과(결론, 의견, 승인 여부)
- 변경 이력: 인시던트 발견, 원인 분석, 수정, 수정 여부 확인 등 처리 활동에 대한 기록

인시던트 처리 과정을 추적할 수 있는 인시던트 상태는 일반적으로 다음과 같이 정의하지만 조직 또는 프로젝트 상황에 따라 인시던트 상태를 추가하거나 축소해 사용한다.

- 발견: 발견한 인시던트를 시스템에 등록한 최초 상태, 테스터 또는 인시던트 발견자가 입력
- 수정 중: 등록한 인시던트를 개발자가 수정(조치)하는 상태, 개발자 또는 인시던트 조치자가 입력
- 수정 완료: 인시던트 수정(조치)을 완료한 상태, 개발자 또는 인시던트 조치자가 입력
- 종료: 수정(조치) 완료된 인시던트의 수정(조치) 여부 확인이 완료된 상태. 결함 처리 완료. 테스터 또는 인시던트 발견자가 입력

일반적으로 인시던트를 체계적으로 관리하기 위해 인시던트 관리 도구 또는 이슈 관리 도구를 사용한다. 인시던트는 테스트 조직과 개발 조직이 체계적이고 투명하게 관리해야 하며, 이를 위해 자동화 도구를 활용하는 것이다. 인시던트 관리 도구는 인시던트 상태를 변경하고 추적하는 기본 기능 외에도 인시던트 상황을 실시간으로 조회 할 수 있는 대시보드 기능을 제공하기도 한다. 그리고 인시던트 관련 통계 및 보고서를 제공함으로써 이해관계자의 의사 결정에 도움을 준다.

Appendix A – 참고 문서

- ISTQB Certified Tester Foundation Level Syllabus 2011
- ISTQB Glossary of Term used in Software Testing Version 2.1
- ISO/IEC/IEEE 29119-3(Software Testing Standard, Test Documentation)

Appendix B – 학습목표 수준

학습 목표의 지식 수준 정의

- [레벨 K1] 기억, 상기, 기초적인 수준의 인식 (Remember): 용어, 개념 등을 알며 기억한다

  (예제) 결함 수정 중 신규 결함이 유입됐는지를 확인하는 테스팅은?

  - 리그레션 테스팅 (Regression Testing)

- [레벨 K2] 이해, 요약, 구분, 설명 (Understand): 관련 배경이나 이유를 이해하고 설명할 수 있다

  (예제) 장애 (Failure) 발생 원인은?

  - 사람의 실수 (Error)로 결함 (Defect)이 유입되고, 시스템 또는 소프트웨어가 구동하면서 결함을 실행시키기 때문
  - 시스템 또는 소프트웨어의 잠재 결함 외에도 환경적인 요인에 의해 발생

  (예제) 명세 기반 테스팅의 일반적인 특징은?

  - 테스트 대상을 명세화한 공식 또는 비공식 모델 사용
  - 정해진 절차와 원리를 이용해 체계적으로 테스트 케이스 도출

- [레벨 K3] 구현, 작성, 사용, 적용(Apply): 주어진 조건과 기법을 이용해 문제를 해결할 수 있다

  (예제) 다음 요구사항 명세에서 동등 분할 테스트 케이스는?

  (예제) 다음 시나리오에서 도출 가능한 테스트 케이스 수는?

Appendix C – 변경 이력

일자	버전	주요 내용	기타
2016.09.07	v1.0	KSTQB Certified Tester Basic Level 실라버스	

Appendix D – 찾아보기

ㅊ

ㅌ

ㅍ

ㅎ

- End of Documentation -