Certified Tester
Foundation Level Syllabus

Version 2011

Korean

Korean Software Testing Qualifications Board

목차

제1장 테스팅의 기초 (K2) | 155분

학습목표

1.1 테스팅이 왜 필요한가? (K2)

LO-1.1.1 소프트웨어의 결함이 사람, 환경, 또는 기업에 어떤 악영향을 끼치는지 예를 들어 설명할 수 있다 (K2)

LO-1.1.2 결함의 원인과 그 결과를 구별할 수 있다 (K2)

LO-1.1.3 테스팅이 필요한 이유를 예를 들어서 설명할 수 있다 (K2)

LO-1.1.4 품질 보증(QA)에서 테스팅이 필요한 이유를 설명하고 더욱 높은 품질 확보에 테스팅이 어떻게 기여하는지 예를 들 수 있다 (K2)

LO-1.1.5 오류(Error), 결함(Defect), 결점(Fault), 장애(Failure) 등의 용어와 거기에 상응하는 실수(Mistake) 및 버그(Bug)라는 용어를 예를 들어 설명하고 비교할 수 있다 (K2)

1.2 테스팅이란 무엇인가? (K2)

LO-1.2.1 테스팅의 일반적인 목적을 상기할 수 있다 (K2)

LO-1.2.2 소프트웨어 수명주기에서의 각 단계별 테스팅 목적을 예를 들어 설명할 수 있다 (K2)

LO-1.2.3 테스팅과 디버깅을 구별할 수 있다 (K2)

1.3 테스팅의 7가지 기본 원리 (K2)

LO-1.3.1 테스팅의 7가지 기본 원리를 설명할 수 있다 (K2)

1.4 테스트 프로세스의 기초 (K1)

LO-1.4.1 테스트 계획부터 테스트 마감까지, 5가지 기본 테스트 활동과 각 활동의 주요 업무를 상기할 수 있다 (K1)

1.5 테스팅의 심리학 (K2)

LO-1.5.1 테스팅의 성공에 영향을 주는 심리적인 요인을 상기할 수 있다 (K1)

LO-1.5.2 테스터와 개발자의 심리 및 사고방식을 대조할 수 있다 (K2)

1.1 테스팅이 왜 필요한가? (K2) | 20분

용어

버그(bug), 결함(defect), 에러(error), 장애(failure), 결점(fault), 실수(mistake), 품질(quality), 리스크(risk)

1.1.1 소프트웨어 시스템 관점에서의 테스팅의 필요성 (K1)

소프트웨어 시스템은 비즈니스 응용 프로그램 (예: 은행업무)에서 소비자 제품(예: 자동차)까지 생활의 많은 부분에서 사용되고 있다. 대다수의 사람은 이러한 소프트웨어 시스템을 사용하면서 소프트웨어가 기대한 대로 동작하지 않는 경우를 많이 접해 보았을 것이다. 금전적, 시간적 손실, 비즈니스의 이미지 손상은 물론, 심하게는 부상이나 사망에 이르기까지 올바르게 동작하지 않는 소프트웨어는 다양하고 심각한 문제를 일으킬 수 있다.

1.1.2 소프트웨어 결함의 원인 (K2)

인간은 프로그램 코드 또는 문서를 작성하면서 결함(결점, 버그)을 만드는 오류(에러, 실수)를 범할 수 있다. 코드에 존재하는 결함이 실행되면 시스템은 의도된 것과는 다르게 행동하게 되는데, 이것을 장애라고 한다. 소프트웨어, 시스템, 문서의 결함은 장애의 원인이 되지만, 모든 결함이 장애를 일으키는 것은 아니다.

결함은 인간이 오류를 범하기 쉬우므로 발생하며, 시간적인 압박, 복잡한 코드, 기반 환경 (Infrastructure)의 복잡성, 기술이나 시스템의 변경, 그리고 수많은 시스템 상호 간의 연동 등의 이유로 발생한다.

장애는 이와 같은 결함에 의해서뿐만 아니라 환경적인 조건에 의해서도 발생한다. 즉, 방사선, 자기, 전자기장, 물리적 오염 또한 소프트웨어의 결함을 유발할 수 있으며, 이러한 환경적인 조건이 하드웨어 조건을 변경시켜 소프트웨어의 실행에 영향을 미칠 수 있다.

1.1.3 소프트웨어 개발, 유지보수, 운영에서의 테스팅의 역할 (K2)

발견하지 못했던 시스템 또는 문서의 결함들을 체계적인 테스팅을 통해 출시(Release) 전에 발견하고 수정한다면, 운영 환경 내에서 발생하는 결함들의 리스크(위험)를 줄이는데 기여할 수 있으며, 소프트웨어 시스템의 품질 향상에도 도움을 준다.

소프트웨어 테스팅은 계약상(법적) 요구조건들 또는 산업에 특화된 표준들을 만족시키기 위해서도 필요하다.

1.1.4 테스팅과 품질 (K2)

테스팅을 통해 발견한 결함(데이터)에 근거하여 대상 소프트웨어의 기능 또는 비기능적 요구사항과 품질 특성(기능성, 신뢰성, 사용성, 효율성, 유지보수성, 이식성 등) 관련 품질 측정이 가능하다(비기능 테스팅은 2장 참조). 소프트웨어의 품질 특성에 대한 더욱 상세한 정보는 국제표준인 '소프트웨어 공학 - 소프트웨어 제품 품질(Software Engineering - Software Product Quality)' (ISO/IEC 9126)에 기술되어 있다.

테스팅으로 발견된 결함이 극소수이거나 없다면, 테스트 설계 및 실행이 정상적으로 진행되었다는 전제하에 소프트웨어의 품질에 대한 확신(Confidence)을 가질 수 있다. 올바르게 설계된 테스트가 성공적으로 완료되면 시스템의 전반적인 리스크 수준은 감소한다. 테스팅이 결함을 찾는다면, 발견된 결함이 수정될 때 소프트웨어 시스템의 품질은 향상된다.

품질을 높이기 위해서는 이전 프로젝트를 통해 많은 테스트 경험과 정보를 확보해야 한다. 다른 프로젝트에서 발견된 결함의 근본 원인에 대한 이해를 바탕으로 프로세스를 개선할 수 있으며, 그러한 결함의 재발을 방지함으로써, 결과적으로 차후 시스템의 품질을 개선할 수 있다. 이것은 품질 보증의 한 측면이다.

개발 표준이나 교육 훈련 그리고 결함 분석 등과 함께 테스팅은 품질 보증 활동의 하나로 통합되어야 한다.

1.1.5 테스팅, 얼마나 해야 충분한가? (K2)

적절한 테스팅의 정도를 파악하기 위해서는 리스크 수준을 고려해야 하며, 기술적인 내용, 안정성, 비즈니스 리스크, 시간, 비용과 같은 프로젝트의 제약 사항을 고려해야 한다. 리스크는 5장에서 다루어진다.

테스팅은 테스트된 소프트웨어나 시스템을 다음 개발 단계 및 고객에게 전달하는 데 있어 프로젝트 이해관계자들이 출시 결정을 내릴 수 있도록 충분한 정보를 제공해야 한다.

1.2 테스팅이란 무엇인가? (K2) | 30분

용어

디버깅(debugging), 요구사항(requirement), 리뷰(review), 테스트 케이스(test case), 테스팅(testing), 테스트 대상(test object)

배경

일반적으로 테스팅을 소프트웨어를 실행하면서 테스트를 수행하는 동적 테스팅으로만 인식하는 경우가 많다. 그것은 테스팅의 일부분일 뿐이며 테스트 활동 전부를 나타내는 것은 아니다.

테스트 활동은 테스트를 수행하기 전과 후에도 존재한다. 테스트 계획과 제어, 테스트 조건(Test conditions)의 선택, 테스트 케이스의 설계와 수행, 결과 점검, 테스트 완료 조건의 평가, 테스트 프로세스와 테스트 대상 시스템에 대한 보고, 테스트 단계 완료 후에 행하는 마무리(Finalizing) 또는 마감 활동의 종료와 같은 일련의 활동들이 여기에 포함된다. 테스팅은 문서(소스 코드 포함)의 리뷰와 정적 분석(Static analysis) 역시 포함한다.

동적 테스팅 (Dynamic testing)과 정적 테스팅(Static testing) 모두 유사한 목적을 달성하려는 방법으로 사용될 수 있으며, 테스트 대상 시스템의 품질을 향상시키고, 개발 프로세스와 테스팅 프로세스를 개선하기 위한 정보를 제공한다.

테스팅은 다음과 같은 목표를 가질 수 있다:

- 결함 발견
- 품질 수준에 대한 자신감 획득
- 의사결정을 위한 정보 제공
- 결함 예방

개발 수명주기 초기에 갖는 테스트 계획 관련 사고(思考) 과정과 활동(테스트 디자인을 통한 테스트 베이시스 검증)은 코드에 유입될 수 있는 결함을 방지하는 데 도움이 된다. 요구사항 등 문서의 리뷰와 이슈의 식별 및 해결 역시 코드에 결함이 나타나는 것을 사전에 방지하도록 도와준다.

테스팅에 대한 관점이 다양한 이유는 테스팅의 목적이 다양하기 때문이다. 예를 들면, 개발과정에서 테스팅(예: 컴포넌트, 통합 그리고 시스템 테스팅)의 주요한 목적은 소프트웨어의 결함을 찾아내고 수정하기 위해 가능한 많은 장애 상황을 만들어내는 것이다. 인수 테스팅에서의 주요한 목적은 예상된 대로 시스템이 동작하는지 확인하고, 요구사항에 맞는지 확신을 얻는 과정이다.

결함 수정의 의도 없이 단지 소프트웨어의 품질을 평가하기 위한 테스팅의 경우는 주어진 시간 내에서 시스템의 출시 관련 리스크 정보를 프로젝트 이해관계자에게 전달하는 것이 목적일 수 있다. 유지보수 테스팅은 종종 변경에 대한 개발 과정 동안 유입되는 새로운 결함을 확인하는 리그레션 테스팅 과정을 포함한다. 운영 테스팅의 주요 목적은 신뢰성 또는 가용성 등의 시스템 특성을 평가하는 것일 수 있다.

테스팅은 디버깅과 근본적으로 구분되는 개념이다. 동적 테스팅은 결함에 의해 발생한 장애를 보여준다. 반면,

디버깅은 그 장애의 원인을 발견, 분석하고 제거하는 개발 활동이며, 이후 테스터에 의한 재테스팅을 통해 발견된 장애가 제대로 수정되었는지 확인하게 된다. 일반적으로 테스터는 테스트를, 개발자는 디버그를 수행하는 것이 이러한 활동에 대한 책임 영역이다.

테스팅하는 절차(Process of testing)와 관련된 활동은 1.4장에 설명되어 있다.

1.3 테스팅의 7가지 기본 원리 (K2) | 35분

용어

완벽한 테스팅(exhaustive testing)

기본 원리

다수의 테스팅 원리가 지난 40여 년간 제안됐으며 테스팅 전반에 대한 일반적인 가이드라인 역할을 해오고 있다.

원리 1 – 테스팅은 결함이 존재함을 밝히는 활동이다

테스팅은 결함이 존재함을 밝힐 수 있지만, 결함이 없다는 것을 증명할 수는 없다. 테스팅은 소프트웨어에 잠재적으로 존재하는 결함을 줄일 수는 있지만, 결함이 전혀 발견되지 않은 경우라도 해당 소프트웨어에 결함이 없다고 증명할 수는 없다.

원리 2 – 완벽한 테스팅은 불가능하다

모든 가능성(입력과 사전 조건의 모든 조합)을 테스팅하는 것은 매우 간단한 소프트웨어를 제외하고는 가능하지 않다. 따라서 완벽한 테스팅보다는, 테스트 대상의 리스크 분석과 우선순위를 토대로 테스트 노력을 집중하는 것이 옳다.

원리 3 – 테스팅을 개발 초기에 시작한다

초기에 결함을 찾아내려면, 테스트 활동은 소프트웨어나 시스템 개발 수명주기에서 가능한 초기에 시작되어야 하며, 설정한 테스트 목표에 집중해야 한다.

원리 4 – 결함 집중

테스트 노력은 각 모듈별 추정되는 혹은 차후 관찰된 결함 밀도에 따라 분배되어야 한다. 출시 전에 발견되는 대다수의 결함은 소수의 특정 모듈에 집중되어 발생하는 경향을 보이며, 이러한 결함의 집중은 대부분 운영상의 장애를 초래한다.

원리 5 – 살충제 패러독스(Pesticide paradox)

동일한 테스트 케이스로 동일한 테스트를 반복적으로 수행한다면, 나중에는 더 이상 새로운 결함을 찾아내지 못한다. 이러한 "살충제 패러독스"를 극복하기 위해서는, 테스트 케이스를 정기적으로 리뷰, 개선할 필요가 있고, 잠재적인 결함을 더 찾아내기 위해서는 소프트웨어 또는 시스템의 다른 부분을 테스트하기 위한 새롭고 다른 테스트 케이스를 작성할 필요가 있다.

원리 6 – 테스팅은 정황(Context)에 의존적이다.

테스팅은 정황에 따라 다르게 진행된다. 예를 들어, 안전-최우선 소프트웨어를 테스트하는 경우, 전자상거래 사이트를 테스트할 때와는 다른 방식으로 진행해야 한다.

원리 7 – 오류-부재의 궤변

개발된 시스템이 사용자의 필요와 기대에 부응하지 못하고 사용성이 현저히 낮다면 결함을 찾고 수정하는 과정은 아무 소용없다.

1.4 테스트 프로세스의 기초 (K1) | 35분

용어

확인 테스팅(confirmation testing), 재테스팅(re-testing), 완료 조건(exit criteria), 인시던트(incident), 리그레션 테스팅(regression testing), 테스트 베이시스(test basis), 테스트 조건(test condition), 테스트 커버리지(test coverage), 테스트 데이터(test data), 테스트 실행(test execution), 테스트 로그(test log), 테스트 계획(test plan), 테스트 프로시저(test procedure), 테스트 정책(test policy), 테스트 스위트(test suite), 테스트 요약 보고서(test summary report), 테스트웨어(testware)

배경

전체 테스트 과정 중 가장 시각적으로 드러나는 부분은 테스트 실행 단계다. 그러나 테스팅을 효율적이고 효과적으로 실행하기 위해서는 테스트를 계획하고, 테스트 케이스를 설계하며, 테스트 수행을 준비하고, 테스트 진행 상태를 확인하고 평가하는 활동을 선행해야 한다.

기본적인 테스트 프로세스의 주요 활동은 다음과 같이 구성되어 있다.

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

테스트 프로세스의 주요 활동은 논리적으로는 순차적이지만, 프로세스 내의 활동들은 중복되거나 동시에 발생할 수 있다. 따라서 시스템과 프로젝트의 정황에 맞추어 이런 주요 활동을 조정하는 것이 필요하다.

1.4.1 테스트 계획과 제어 (K1)

테스트 계획 수립은 프로젝트의 목표와 임무를 달성하기 위해 테스팅 목표와 각 테스트 활동의 명세 사항을 정의하는 활동이다.

테스트 제어는 계획 대비 실제 진행 상황을 비교하는 지속적인 활동으로 계획과의 차이 정도를 포함한 진행 상태를 보고하는 것을 포함한다. 테스트 프로젝트의 목표 및 임무를 달성하기 위해 필요한 조치를 취하는 것도 여기에 해당된다. 테스팅을 제어하기 위해서는 프로젝트 내내 테스팅 활동을 모니터링해야 한다. 테스트 계획은 테스팅의 제어와 모니터링 활동으로부터 받는 피드백을 반영한다.

테스트 계획과 제어는 5장에서 상세하게 다뤄진다.

1.4.2 테스트 분석과 설계 (K1)

테스트 분석과 설계는 일반적인 테스팅 목적을 실제적이고 구체적인 테스트 조건과 테스트 케이스로 변환하는 활동이다.

테스트 분석과 설계는 다음의 주요한 작업을 포함한다.

- 요구사항, 소프트웨어 무결성 수준[1] (리스크 레벨), 리스크 분석 리포트, 아키텍처(Architecture), 설계, 인터페이스(Interface) 명세 등 테스트 베이시스 리뷰
- 테스트 베이시스와 테스트 대상의 테스트 용이성 평가
- 테스트 아이템, 요구명세, 동작 및 소프트웨어 구조 등의 분석을 기반으로 테스트 조건을 식별하고 우선순위를 산정
- 개요 수준의 테스트 케이스 설계와 우선순위 산정
- 테스트 조건과 테스트 케이스에 필요한 테스트 데이터 식별
- 테스트 환경 설계 및 필요한 기반 환경 및 도구 식별
- 테스트 베이시스와 테스트 케이스 간의 양방향 추적성 생성

1.4.3 테스트 구현과 실행 (K1)

테스트 구현과 실행은 테스트 실행을 위해 테스트 케이스를 특정한 순서에 따라 결합하여 테스트 프로시저 또는 스크립트를 명세화하고 테스트 실행에 필요한 다른 정보를 취합하고, 테스트 환경을 구축하고, 테스트를 실행하는 활동이다.

테스트 구현과 실행은 다음과 같은 주요 작업을 포함한다.

- 테스트 케이스의 설계 마감, 구현 및 우선순위 선정 (테스트 데이터 식별 포함)
- 테스트 프로시저의 개발과 우선순위 선정, 테스트 데이터의 생성, 부가적으로 테스트 하네스 (Test harnesses, 테스트 드라이버)의 준비 및 자동화 테스트 스크립트 작성
- 효율적인 테스트 실행을 위해 테스트 프로시저에서 테스트 스위트(테스트 케이스 묶음) 생성
- 테스트 환경이 올바르게 구축되었는지 확인
- 테스트 베이시스와 테스트 케이스 간의 양방향 추적성 검증 및 업데이트
- 계획된 순서를 기준으로, 수동 또는 테스트 실행 도구를 사용하여 테스트 프로시저 수행
- 테스트 수행 결과를 기록하고, 테스트 중인 소프트웨어, 테스트 도구, 테스트웨어의 버전과 정보 기록
- 예상 결과와 실제 결과의 비교
- 예상 결과와 실제 결과 사이의 불일치를 인시던트로 보고하고, 그 원인을 분석(예: 코드 결함, 특정 테스트 데이터로 인한 결함, 테스트 문서에 의한 결함, 또는 테스트 실행 중 발생한 결함 등)
- 각각의 불일치에 대한 수정 조치에 따라 필요한 테스트 활동을 반복. 예를 들어, 수정 확인을 위해 이전에 실패한 테스트를 재실행하거나(확인 테스팅), 수정으로 인해 소프트웨어의 변경되지 않은 영역에 결함이 유입되지 않았는지, 또는 수정이 다른 결함을 발생시키지 않았는지 확인하기 위해 테스트를 반복 실행(리그레션 테스팅)

1.4.4 완료 조건의 평가와 보고 (K1)

완료 조건 평가는 초기에 정의된 테스트 목표에 비해 실제 테스트가 어느 정도 실행되었는지를 평가하는 활동이다. 이 활동은 각 테스트 레벨마다 수행되어야 한다. (2.2장 참조)

[1] 소프트웨어가 프로젝트 이해관계자에 의해 중요하다고 생각되어 정의된 소프트웨어 또는 소프트웨어 기반 시스템의 특징(예: 소프트웨어의 복잡성, 리스크 평가, 안전 수준, 안전 수단, 바람직한 성능, 신뢰성, 비용 등)과 얼마나 부합하는지의 정도/수준

완료 조건의 평가는 다음과 같은 주요 작업을 포함한다.

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

1.4.5 테스트 마감 활동 (K1)

테스트 마감 활동은 완료된 테스트 활동에서 데이터를 수집하여, 테스트에서 발견된 사실 및 수치적 데이터와 함께 테스팅 경험과 테스트웨어를 종합하고 축적하는 활동이다. 테스트 마감 활동은 소프트웨어 시스템이 출시되었을 때, 테스트 프로젝트가 완료(또는 취소) 되었을 때, 특정 마일스톤(Milestone)이 달성 되었을 때, 또는 유지보수 활동에서 추가 개발되거나 업데이트된 부분이 출시 완료 되었을 때와 같은 프로젝트 마일스톤 때 일어난다.

테스트 마감 활동은 다음의 주요 작업을 포함한다.

- 계획되었던 결과물의 산출 상태 확인
- 인시던트 리포트 마감, 또는 미해결 항목의 변경 요청 리포트 제기
- 시스템 인수의 문서화
- 차후 사용을 위해 테스트웨어, 테스트 환경, 테스트 기반 설비를 마무리하고 보관
- 테스트웨어를 유지보수 조직에 인계
- 향후 릴리스나 프로젝트에 필요한 변경 내용의 확인을 위해 테스트를 통해 얻어진 교훈 분석
- 테스트 성숙도 개선을 위해 수집된 정보 활용

1.5 테스팅의 심리학 (K2) | 25분

용어

오류 추정(error guessing), 독립성(independence)

배경

테스팅이나 리뷰 과정에서 필요한 사고방식은 소프트웨어 개발에서의 사고방식과는 다르다. 개발자의 사고방식에 따라 자신이 개발한 코드를 직접 테스트 할 수도 있지만, 이 역할을 테스터에게 분리시키는 것은 개발자가 개발 활동에 집중하도록 도와주기 위해서다. 또한, 훈련된 전문적인 테스팅 리소스에 의한 독립적인 시각의 제공 등 추가적인 이점을 얻기 위해서이기도 하다. 독립적인 테스팅은 테스팅의 어느 레벨에서나 수행할 수 있다.

일정 수준의 독립성 확보는(작성자가 갖는 저작물에 대한 저자의 편향성을 줄이면서) 테스터가 결함과 장애를 찾아내는 데 더 효과적일 수 있게 해준다. 그러나 독립성이 친숙함을 대체할 수 없으며, 개발자가 자신이 작성한 코드에서 많은 결함을 효율적으로 찾아낼 수 있는 것이 사실이다.

테스팅의 독립성 정도를 낮음에서 높음 순으로 아래와 같이 정의할 수 있다.

- 테스트 중인 소프트웨어를 작성한 저자가 설계한 테스트 (독립성 낮음)
- 개발 조직 내의 다른 사람이 설계한 테스트
- 독립적인 테스트 팀과 같은 다른 조직의 일원 또는 테스트 전문가(예: 사용성 또는 성능 테스트 전문가 등)가 설계한 테스트
- 다른 조직이나 회사(예: 외부 조직에 의한 자격 인증, 아웃소싱(Outsourcing)) 소속의 인원이 설계한 테스트

사람이나 프로젝트는 목표에 따라 움직인다. 마찬가지로 테스터들도 자신의 계획을 테스트 관리자나 프로젝트 이해관계자가 설정한 목표와 일치시키고자 한다 (예: 결함의 발견을 목표로 할지, 아니면 소프트웨어가 목적에 맞게 동작하는지를 확인하는 등). 따라서 테스팅의 목표를 명확하게 정하는 것이 매우 중요하다.

테스트 과정 중에 장애를 확인하는 것은 테스트 대상 제품이나 작성자에 대한 비판으로 오해될 소지가 있다. 그런 이유로, 테스팅이 제품 리스크 관리 측면에서 매우 건설적인 활동임에도 불구하고, 종종 파괴적인 활동으로 간주된다. 시스템에서 장애를 찾아내기 위해선 호기심, 전문적이면서 비관적으로 사물을 볼 수 있는 능력, 비판적 시선, 세심한 주의력, 개발자 및 개발팀 동료와의 원활한 의사소통, 그리고 결함을 유추해 내는데 근거가 되는 경험 등을 필요로 한다.

오류나 결함, 장애가 긍정적인 방법으로 의사소통된다면, 테스터와 개발자, 분석가, 설계자 간에 발생할 수 있는 감정 악화를 피할 수 있다. 이것은 테스팅 뿐만 아니라 리뷰 과정에서도 그대로 적용된다.

테스터나 테스트 리더는 결함이나 테스트 진행 상황, 리스크 요소에 대한 사실적인 정보를 긍정적인 방법으로 현명하게 전달할 수 있는 좋은 대인 관계가 필요하다. 결함 정보는 소프트웨어나 문서의 작성자가 해당 산출물 관련 기술을 개선하는 데 도움을 줄 수 있다. 테스팅 기간 동안 결함을 발견하고 수정하는 것은 이후의 시간과 비용을 절감할 수 있으며, 리스크 요소를 줄일 수 있다.

테스터가 결함을 지적하는 불쾌한 소식만을 전하는 것으로 비춰지면 긍정적인 의사소통 형성 문제가 발생할 것이다. 다음은 테스터와 다른 관련자 간의 의사소통과 관계 개선을 위한 몇 가지 방법이다.

- 다툼보다는 협력으로 시작하라 – 더 나은 품질의 시스템 제작이라는 공통의 목표를 모두에게 주지시킨다
- 소프트웨어를 개발한 사람에 대한 비평을 배제하고, 중립적이고 사실에 근거한 제품의 결함만을 전달하려고 노력한다. 예를 들어, 객관적이고 사실적인 인시던트 리포트를 작성하고 발견한 사항을 리뷰한다
- 다른 인원이 어떻게 느끼는지, 왜 그렇게 반응하는지 이해하도록 노력한다
- 상호간에 의사소통했던 것을 상대방이 정확하게 이해했는지 확인한다

1.6 윤리 강령	10분

소프트웨어 테스팅에 참여함으로써 개인은 기밀 정보 및 특권적 정보들을 접하게 된다. 윤리 강령은 다른 이유 외에도 이러한 정보가 부적절하게 사용되지 않도록 보장하기 위해 필요하다. 엔지니어를 위한 ACM과 IEEE의 윤리 강령에 준하여 ISTQB에서는 다음과 같은 윤리 강령을 갖추고 있다.

공공성 – 인증 소프트웨어 테스터는 공공의 이익에 부합되도록 행동한다

고객과 고용주 – 인증 소프트웨어 테스터는 공공의 이익과 부합하면서, 고객과 고용주의 이익을 최우선으로 행동한다

제품 – 인증 소프트웨어 테스터는 제공하는(테스트 대상 제품과 시스템의) 결과물이 최상의 전문가적 표준을 충족하도록 한다

판단 – 인증 소프트웨어 테스터는 전문적인 판단을 함에 있어서 위상과 독립성을 유지한다

관리 – 인증 소프트웨어 테스트 관리자 및 리더는 소프트웨어 테스팅 관리에 대해 윤리적 접근을 취하도록 지지하고 장려한다

전문성 – 인증 소프트웨어 테스터는 공공의 이익에 부합하는 전문 직업인으로의 위상과 평판(명성)을 선도한다

동료 – 인증 소프트웨어 테스터는 동료에게 공정하고 협조적이며, 소프트웨어 개발자와 협력을 도모한다

자기 자신 – 인증 소프트웨어 테스터는 테스터의 전문 직업 관련 평생 학습에 참여하며 직업 관행에 있어서 윤리적 접근 방식을 취해야 한다

제2장 소프트웨어 수명주기와 테스팅 (K2) | 115분

학습목표

2.1 소프트웨어 개발 모델 (K2)

LO-2.1.1 개발 수명주기에서 개발 활동, 테스트 활동과 개발 산출물의 관계를 프로젝트와 제품 유형을 예로 들어 설명할 수 있다 (K2)

LO-2.1.2 소프트웨어 개발 모델은 프로젝트와 제품 특성 등의 정황에 맞도록 선택되어야 함을 이해할 수 있다 (K1)

LO-2.1.3 모든 개발 수명주기 모델에 적용 가능한 좋은 테스팅의 특징을 상기할 수 있다 (K1)

2.2 테스트 레벨 (K2)

LO-2.2.1 서로 다른 테스팅 레벨을 다음과 같은 비교 항목으로 설명할 수 있다. 주요 목표, 일반적인 테스팅 대상, 일반적인 테스팅 목표(예: 기능적 또는 구조적), 관련 개발 산출물, 테스트 시행 주체, 식별하고자 하는 결함과 장애의 유형 (K2)

2.3 테스트 유형 (K2)

LO-2.3.1 4가지 소프트웨어 테스트 유형(기능, 비기능, 구조, 변경 사항 관련)을 예를 들어 비교할 수 있다 (K2)

LO-2.3.2 기능 테스팅 또는 구조 테스팅이 모든 테스트 레벨에서 수행 가능함을 인지할 수 있다 (K1)

LO-2.3.3 비기능적인 요구사항에 근거한 비기능 테스트 유형을 식별하고 기술할 수 있다 (K2)

LO-2.3.4 소프트웨어 시스템의 구조 또는 아키텍처 분석에 근거한 테스트 유형을 식별하고 기술할 수 있다 (K2)

LO-2.3.5 확인 테스팅과 리그레션 테스팅의 목적을 구분하여 기술할 수 있다 (K2)

2.4 유지보수 테스팅 (K2)

LO-2.4.1 테스트 유형, 테스트 진입 조건, 테스트 대상을 기준으로 유지보수 테스팅(기존 시스템에 대한 테스팅)과 새로운 응용 프로그램의 테스팅을 비교할 수 있다 (K2)

LO-2.4.2 유지보수 테스팅의 수행 조건(변경, 마이그레이션, 단종 등)을 설명할 수 있다 (K1)

LO-2.4.3 유지보수에서의 리그레션 테스팅과 영향도 분석(Impact analysis)의 역할을 기술할 수 있다 (K2)

2.1 소프트웨어 개발 모델 (K2) | 20분

용어

상용 소프트웨어(COTS), 순차-점진적 개발 모델(iterative-incremental development model), 검증(벨리데이션, validation), 검증(베리피케이션, verification), V-모델

배경

테스팅은 소프트웨어 개발 활동과 독립적으로 존재하지 않고 밀접하게 연계되어 있으므로, 개발 수명주기 모델(Life cycle model)에 따라 테스트 접근법을 다르게 적용해야 한다.

2.1.1 V-모델 (순차적 개발 모델) (K2)

여러 가지 변형된 형태의 V-모델이 존재하지만, 일반적인 유형의 V-모델은 4단계의 테스트 레벨로 구성되어 있고 4단계의 개발 레벨과 대응된다.

이 실라버스에서 다루는 4단계의 테스트 레벨은 다음과 같다:

- 컴포넌트(단위) 테스팅
- 통합 테스팅
- 시스템 테스팅
- 인수 테스팅

실제 업무에서는 프로젝트나 소프트웨어 제품의 특성에 따라 V-모델에서의 개발 단계 및 테스트 레벨을 더 많이 구성하거나 더 적게 구성할 수도 있다. 예를 들어, 컴포넌트 테스팅 이후에 컴포넌트 통합 테스팅이 있을 수 있고, 시스템 테스팅 이후에 시스템 통합 테스팅이 존재할 수 있다.

개발 기간 중에 생산되는 소프트웨어 개발 산출물(비즈니스 시나리오, 유즈케이스, 요구사항 명세, 설계 문서, 코드 등)은 하나 또는 그 이상의 테스트 레벨에서 테스팅을 수행하는 데 있어 베이시스가 된다. 일반적인 개발 산출물은 CMMI(Capability Maturity Model Integration)나 '소프트웨어 라이프사이클 프로세스(Software life cycle process), (IEEE/IEC 12207)를 참고하도록 한다. 베리피케이션/벨리데이션(Verification and Validation) 및 초기 단계에서의 테스트 설계는 소프트웨어 개발 산출물의 개발 기간 동안 수행될 수 있다.

2.1.2 반복적-점증적(Iterative-incremental) 개발 모델 (K2)

반복적-점진적인 개발은 요구사항 확립, 설계, 시스템 구현 및 테스트를 짧게 연속적으로 반복 진행하는 방식이다. 대표적인 유형으로는 프로토타이핑(Prototyping), RAD(Rapid Application Development), RUP(Rational Unified Process), 애자일(Agile) 개발 모델 등이 있다. 이러한 개발 모델에 따라 생산되는 시스템은 각 반복주기 동안 여러 개의 테스트 레벨에 걸쳐 테스트 될 수 있다. 이전에 개발한 결과물은 현재의 반복주기에서 추가 개발한 증분(증가된 부분)에 의해 규모가 점차 커져 부분 시스템(Partial system)을 형성하게 된다. 이런 부분 시스템은 테스트되어야 한다. 리그레션 테스팅의 중요도는 첫 번째 반복주기 이후 모든 반복주기에서 점차 증가하게 된다. 베리피케이션과 벨리데이션은 각각의 증분 산출물을 대상으로 수행될 수 있다.

2.1.3 개발 수명주기(Lifecycle) 모델에서의 테스팅 (K2)

성공적인 테스팅을 위해서는, 그 개발 수명주기 모델에 관계없이 다음과 같은 요건들이 필요하다.

- 모든 개발 활동은 그에 상응하는 테스팅 활동을 동반한다
- 각 테스트 레벨은 그 레벨에 맞는 특정한 목적을 가지고 있다
- 주어진 테스트 레벨에 맞는 테스트의 분석과 설계는 대응하는 개발 활동 동안에 시작되어야 한다
- 개발 수명주기 동안에 문서의 초안이 작성되는 즉시 테스터는 이러한 문서를 리뷰하는 활동에 참가해야 한다

테스트 레벨은 프로젝트나 시스템 아키텍처의 성격에 따라 재조정되거나 합쳐질 수 있다. 예를 들어, 상용 소프트웨어(COTS) 제품을 시스템에 통합하는 경우, 구매자는 시스템 레벨의 통합 테스팅(예: 기반 환경이나 다른 시스템에 통합 또는 배포된 시스템에 통합)과 인수 테스팅(기능적 테스팅, 비기능적 테스팅, 사용자 테스팅, 운영 테스팅 등)을 수행할 수 있다.

2.2 테스트 레벨 (K2) | 40분

용어

알파 테스팅(alpha testing), 베타 테스팅(beta testing), 컴포넌트 테스팅(component testing), 드라이버(driver), 필드 테스트(field testing), 기능 요구사항(functional requirement), 통합(integration), 통합 테스팅(integration testing), 비기능 요구사항(non-functional requirement), 강건성 테스팅(robustness testing), 스텁(stub), 시스템 테스팅(system testing), 테스트 환경(test environment), 테스트 레벨(test level), 테스트 주도 개발(TDD, test-driven development), 사용자 인수 테스트(user acceptance testing)

배경

각각의 테스트 레벨에 대해 다음의 내용이 식별될 수 있다: 일반적인 목표(목적), 테스트 케이스를 도출해 내는데 필요한 개발 산출물(테스트 베이시스), 테스트 대상(테스트되는 그 무엇), 발견될 전형적인 결함과 장애, 테스트 하네스(테스트 드라이버) 요구사항과 도구 지원, 상세한 테스트 접근법과 수행 주체

시스템의 구성 데이터 테스트는 테스트 계획을 하는 동안 고려해야 한다.

2.2.1 컴포넌트 테스팅(Component Testing) (K2)

테스트 베이시스:

- 컴포넌트 요구사항
- 상세 설계
- 코드

일반적인 테스트 대상:

- 컴포넌트
- 프로그램
- 데이터 변환 / 마이그레이션 프로그램
- 데이터베이스 모듈

컴포넌트 테스팅(단위, 모듈, 또는 프로그램 테스팅)은 테스트 가능한(최소) 단위로 분리된 소프트웨어 모듈, 프로그램, 객체, 클래스 등 내에서 결함을 찾고 그 기능을 검증하는 것이다. 컴포넌트 테스팅은 개발 수명주기와 시스템에 따라 시스템의 다른 부분에서 격리하여 독립적으로 수행할 수 있다. 이때, 스텁(Stub)과 드라이버, 시뮬레이터가 사용될 수 있다.

컴포넌트 테스팅은 구조적인 테스팅(분기 커버리지 등)은 물론 기능성 테스트와 리소스 관련(메모리 유출 검색 등) 테스팅 또는 강건성 테스팅과 같은 특정 비기능 테스트를 포함한다. 테스트 케이스는 컴포넌트 명세, 소프트웨어 상세 설계 또는 데이터 모델 명세와 같은 개발 산출물에서 도출된다.

일반적으로 컴포넌트 테스팅은 코드를 중심으로 수행하며, 단위 테스트 프레임웍 또는 디버깅 도구 같은 개발 환경의 지원이 필요하다. 실무에서는 코드를 작성한 프로그래머가 직접 컴포넌트 테스팅에 참여한다. 일반적으로 결함을 발견할 때마다 바로 수정하고, 결함에 대한 기록 과정은 생략하는 것이 일반적이다.

컴포넌트 테스팅의 접근법 중에는 코딩 전에 테스트 케이스를 준비하고 자동화하는 방법이 있다. 이것이 테스트 중심의 개발 방법론(Test-first approach 또는 Test-driven development)이다. 이는 반복적인 성향이 매우 강한 접근법으로, 테스트 케이스를 개발한 후 작은 규모의 코드를 작성하여 통합하고 테스트가 통과할 때까지 반복 수행한다.

2.2.2 통합 테스팅(Integration Testing) (K2)

테스트 베이시스:

- 소프트웨어와 시스템 설계
- 아키텍처
- 워크플로우(Workflows)
- 유즈케이스

일반적인 테스트 대상:

- 서브시스템
- 데이터베이스 구축
- 기반 환경
- 인터페이스
- 시스템 구성과 구성 데이터

통합 테스팅은 컴포넌트 간의 인터페이스를 테스트하는 것은 물론, OS, 파일 시스템, 하드웨어 또는 시스템간 인터페이스와 같은 시스템의 각기 다른 부분과 상호 연동하는 동작을 테스트한다.

통합 테스팅은 하나 이상의 테스트 레벨이 있을 수 있으며, 다양한 크기의 테스트 대상에 대해 수행할 수 있다. 예를 들면,

1. 컴포넌트 통합 테스팅은 소프트웨어 컴포넌트 사이의 상호 작용을 테스트하며 컴포넌트 테스팅 이후에 수행된다.
2. 시스템 통합 테스팅은 서로 다른 시스템간에 또는 하드웨어와 소프트웨어 사이의 상호 작용을 테스트하며 시스템 테스팅 이후에 수행된다. 이 경우에 개발 조직은 한쪽 시스템에 국한되어 제어 권한을 갖게 된다. 이것은 리스크로 간주될 수 있다. 업무 흐름으로 구현된 비즈니스 프로세스가 여러 개의 시스템을 포함하고 있을 수 있다. 시스템 간의 교차점에서 중대한 문제가 야기될 수 있다.

통합하는 범위가 크면 클수록 장애가 특정 컴포넌트나 시스템에서 기인하므로 격리하는 것이 더 어려워지고, 이는 개발의 리스크와 문제 해결(Troubleshooting)을 위한 시간을 증가시킬 수 있다.

시스템 통합 전략은 시스템 아키텍처(예 상향식, 하향식), 기능적 테스트, 트랜잭션 처리 순서, 시스템 또는 컴포넌트의 다른 측면 등에 기반을 둘 수 있다. 결함을 쉽게 격리하고 결함을 조기에 발견하기 위해, 한 번에 통합하는 빅뱅(Big-bang) 전략보다는 순차적이고 체계적인 통합 전략(Incremental approach)을 사용해야 한다.

특정 비기능적 특성(예: 성능) 테스트는 기능 테스트뿐만 아니라, 통합 테스트에 포함될 수도 있다.

통합의 각 단계에서 테스터는 통합 그 자체에만 집중한다. 예를 들어, 모듈 A와 B를 통합한다면, 개별 모듈의 기능에 관심을 두는 컴포넌트 테스팅과 달리, 통합 테스트는 두 개 모듈 사이의 커뮤니케이션에 관심을 두게 된다. 기능적 접근법과 구조적인 접근법 모두 사용될 수 있다.

이상적으로는 테스터가 아키텍처에 대한 이해를 바탕으로 통합 테스팅 계획에 관여해야 한다. 컴포넌트 또는 시스템이 만들어지기 전에 통합 테스트를 계획한다면, 테스팅을 가장 효율적으로 수행할 수 있도록 컴포넌트의 개발 순서를 계획할 수도 있다.

2.2.3 시스템 테스팅(System Testing) (K2)

테스트 베이시스:

- 시스템 및 소프트웨어 요구사항 명세
- 유즈케이스
- 기능 명세
- 리스크 분석 리포트

일반적인 테스트 대상:

- 시스템, 사용자 및 운영 매뉴얼
- 시스템 구성 및 구성 데이터

시스템 테스팅은 개발 프로젝트 차원(범위)에서 정의한 전체 시스템 또는 제품의 동작에 대해 테스트하는 것이다. 테스팅 범위는 명확하게 해당 테스트 레벨의 마스터/레벨 테스트 계획에 정의되어 있어야 한다.

테스팅에서 발견하지 못해 발생할 수 있는 "환경특성 장애(Environment-specific failure)" 리스크를 최소화하기 위해서 시스템 테스팅은 가능한 범위에서 실제 최종 사용 환경 또는 이와 유사한 환경에서 수행해야 한다.

시스템 테스팅은 리스크, 요구사항 명세, 비즈니스 프로세스, 유즈케이스, 시스템의 행동을 묘사하는 기타 개요 수준의 글이나 모델, OS 및 시스템 리소스와의 상호 작용 등을 기반으로 작성된 테스트 케이스를 포함할 수 있다.

시스템 테스팅은 기능 및 비기능 요구사항, 데이터 품질 특성 등을 모두 검증해야 한다. 테스트 엔지니어는 불완전하거나 문서 형태를 갖추지 못한 요구사항을 기반으로 테스트하게 되는 경우도 고려해야 한다. 기능적인 요구사항의 시스템 테스팅은 테스트 대상의 특성에 가장 적절한 명세기반(블랙박스) 기법을 사용한다. 예를 들어, 비즈니스 룰에 표현된 결과들의 조합을 테스트하기 위해 결정 테이블(Decision table) 기법을 사용할 수 있다. 메뉴 구조나 웹 페이지 내비게이션과 같은 구조적인(비기능적인) 요소에 대한 테스팅이 얼마나 완벽하게 수행되었는지 평가하기 위해 구조 기반 기법(화이트박스)을 사용할 수 있다. (4장 참고)

시스템 테스팅은 독립적인 테스트 팀이 수행하는 경우가 대부분이다.

2.2.4 인수 테스팅(Acceptance Testing) (K2)

테스트 베이시스:

- 사용자 요구사항

- 시스템 요구사항
- 유즈케이스
- 비즈니스 프로세스
- 리스크 분석 리포트

일반적 테스트 대상:
- 완전히 통합된 시스템의 비즈니스 프로세스
- 운영 및 유지보수 프로세스
- 사용자 절차
- 양식
- 보고서
- 구성 데이터

인수 테스팅은 시스템을 사용하는 고객이나 사용자가 전담하여 수행하는 경우가 대부분인데, 다른 관련자도 참여할 수 있다.

인수 테스팅의 목적은 시스템이나 시스템의 일부 또는 특정한 비기능적인 특성에 대해 "확신(Confidence)"을 얻는 것이다. 결함을 찾는 것은 인수 테스팅의 주된 관심사가 아니다. 인수 테스팅은 시스템을 배포하거나 실제 사용할 만한 준비가 되었는지에 대해 평가한다. 그러나 인수 테스팅이 반드시 최종 단계의 테스팅이라고 보기는 어렵다. 예를 들어, 대규모의 시스템 통합 테스트를 개별 시스템에 대한 인수 테스트 이후에 실행할 수도 있다.

인수 테스팅은 프로젝트 전(全) 개발 과정에서 수행할 수 있다. 예를 들면,
- 상용(COTS) 소프트웨어 제품은 설치되거나 통합되면 인수 테스팅을 수행할 수 있다
- 컴포넌트의 사용성에 대한 인수 테스팅은 컴포넌트 테스팅 동안에 수행할 수 있다
- 새로운 기능상의 개선에 대한 인수 테스팅은 시스템 테스팅 전에 이루어질 수 있다

인수 테스팅의 전형적인 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
- 사용자 관리
- 유지보수 작업
- 데이터로드 및 마이그레이션 작업
- 보안 취약성에 대한 정기적인 점검

계약 인수 테스팅과 규정 인수 테스팅

계약 인수 테스팅은 맞춤식-개발(Custom-developed) 소프트웨어가 계약상의 인수 통과 조건을 준수하는지 확인하는 테스팅이다. 인수 통과 조건은 인수자가 계약에 동의할 때 정의되어야 한다. 규정 인수 테스팅은 정부 지침, 법률 또는 안전 규정 등 준수해야 하는 규정에 맞게 개발되었는지 확인하는 테스팅이다.

알파 테스팅과 베타(또는 필드) 테스팅

시장에서 판매하는 소프트웨어 또는 상용(COTS) 소프트웨어 개발자는 종종 소프트웨어가 상업적으로 판매되기 전에 목표 시장의 기존 고객이나 잠재 고객으로부터 피드백을 받고 싶어한다. 알파 테스팅은 개발 조직 내에서 수행되지만, 개발팀이 수행해서는 안 된다. 반면 베타 테스팅 또는 필드 테스팅은 실제 환경에서 사용자 혹은 잠재 고객에 의해 수행된다.

고객사의 사이트로 이동하기 전에 개발 완료 후 테스팅하는 소위 '공장 인수 테스팅 (Factory acceptance testing)'은 알파 테스팅과 같은 의미이고, 고객 사이트로 이동한 후에 테스팅하는 '사이트 인수 테스팅(Site acceptance testing)'은 베타 테스팅과 같은 용어이다.

2.3 테스트 유형 (K2) | 40분

용어

블랙박스 테스팅(black-box testing), 코드 커버리지(code coverage), 기능 테스팅(functional testing), 상호운용성 테스팅(interoperability testing), 부하 테스팅(load testing), 유지보수 테스팅(maintainability testing), 성능 테스팅(performance testing), 휴대성 테스팅(portability testing), 신뢰성 테스팅(reliability testing), 보안 테스팅(security testing), 스트레스 테스팅(stress testing), 구조 테스팅(structural testing), 사용성 테스팅(usability testing), 화이트박스 테스팅(white-box testing)

배경

테스트의 구체적인 이유나 대상에 따라 특정 테스트 활동 묶음을 소프트웨어 시스템(또는 시스템의 일부분)을 확인하기 위해 실행할 수 있다.

특정 테스트 유형은 특정한 테스트 목적에 중점을 맞추고, 다음 중 어떤 것이라도 될 수 있다.

- 소프트웨어가 수행하는 기능에 대한 테스팅
- 신뢰성, 사용성과 같은 비기능적인 품질 특성 테스팅
- 소프트웨어 혹은 시스템의 구조나 아키텍처에 대한 테스팅
- 변경 내용에 관련된 테스팅 (예: 결함에 대한 수정이 이루어졌는지에 대한 확인 테스팅(Confirmation testing)과 의도하지 않은 변경을 찾는 리그레션 테스팅(Regression testing)

소프트웨어의 모델은 구조적 테스팅(예: 제어 흐름 모델 또는 메뉴 구조 모델), 비기능적 테스팅 (예: 성능 모델, 사용성 모델, 보안 위협 모델링), 기능적 테스팅(예: 프로세스 흐름 모델, 상태전이 모델 또는 일반 언어 명세) 등의 목적으로 개발/사용될 수 있다.

2.3.1 기능 테스팅(Functional Testing) (K2)

실행되어야 하는 (서브)시스템 또는 컴포넌트의 기능은 요구사항 명세, 유즈케이스 또는 기능적인 명세와 같은 개발 산출물에 기술되어 있거나, 문서화되지 않을 수 있다. 여기서 기능은 시스템이 수행하는 그 "무엇"을 의미한다.

기능 테스팅은 문서화되어 있거나 테스터가 알고 있는 기능과 특징, 그리고 그것들과 특별한 시스템과의 상호운용성을 고려하여 수행하며 모든 테스트 레벨에서 수행될 수 있다(예: 컴포넌트 테스트 레벨에서의 기능 테스팅은 컴포넌트 명세를 기반으로 한다).

명세 기반 기법(Specification-based technique)을 이용해 소프트웨어나 시스템의 기능성에서 테스트 조건과 테스트 케이스를 도출한다(4장 참조). 기능적인 테스팅은 소프트웨어의 외부적인 행동을 고려한다(블랙박스 테스팅).

기능 테스팅의 한가지 형태인 보안성 테스팅(Security testing)은 악의적인 코드(바이러스 등)와 같은 외부로부터의 위협을 감지해 내는 것과 관련이 있는 기능(방화벽)을 확인한다. 기능 테스팅의 또 다른 형태로, 상호운용성 테스팅(Interoperability testing)은 하나 또는 여러 개의 명시된 컴포넌트나 시스템이 서로 상호 작용하는 소프트웨어 제품의 능력을 평가하는 것이다.

2.3.2 비기능 테스팅(Non-functional Testing) (K2)

비기능 테스팅은 성능 테스팅, 부하 테스팅, 스트레스 테스팅, 사용성 테스팅, 유지보수성 테스팅, 신뢰성 테스팅, 그리고 이동성 테스팅(Portability testing) 등을 포함하는 개념이다. 이는 시스템이 "어떻게" 동작하는가를 테스팅한다.

비기능 테스팅은 모든 테스트 레벨에서 수행할 수 있다. 비기능 테스팅이라는 용어는 성능 테스팅에서의 응답 시간과 같이 다양한 척도 또는 스케일(Scale)로 정량화 가능한 소프트웨어나 시스템의 특성을 측정하는 테스트를 의미한다. 이러한 테스트는 '소프트웨어 공학 – 소프트웨어 제품 품질(Software Engineering – Software Product Quality)' (ISO/IEC 9126)에서 정의한 것과 같은 품질 모델을 참고할 수 있다. 비기능 테스팅은 외부로 표출되는 소프트웨어의 습성을 확인할 목적으로 실행되며, 대부분의 경우 목적 달성을 위해 블랙박스 테스트 설계 기법을 활용한다.

2.3.3. 구조적 테스팅(Structural Testing) (K2)

구조적(화이트박스) 테스팅은 모든 테스트 레벨에서 수행할 수 있다. 구조적인 테스트 기법은 특정 유형의 구조 커버리지를 평가하여 테스팅의 보장성 또는 충분함(Thoroughness)을 측정하는 것이므로 명세 기반 기법을 적용한 다음에 사용할 때 가장 좋은 결과를 보여준다.

커버리지는 시스템 또는 소프트웨어의 구조가 테스트 스위트에 의해 테스트된 정도를 말하며, 구조 종류에 대해 커버된 퍼센트로 표시한다. 만일 커버리지가 100%가 아니라면, 누락된 아이템을 테스트하기 위해 더 많은 테스트를 설계하여 커버리지를 높일 수 있다. 커버리지 관련 기법은 4장에서 다룬다.

모든 테스트 레벨에서, 특히 컴포넌트 테스팅과 컴포넌트 통합 테스팅 레벨에서 (자동화)도구(Tools)는 선언(Statements)이나 결정(Decisions)과 같은 코드 커버리지 요소를 측정하는 용도로 사용할 수 있다. 구조적인 테스팅은 코드 레벨뿐만 아니라 호출 체계/구조(Hierarchy)와 같은 시스템의 아키텍처에 기반을 두고 수행할 수 있다.

구조적인 테스팅 접근법은 또한 시스템, 시스템 통합 또는 인수 테스팅의 테스트 레벨(예: 비즈니스 모델이나 메뉴 구조)에 적용할 수 있어 모든 테스트 레벨에서 수행할 수 있다.

2.3.4 변화 관련 테스트: 재테스팅/리그레션 테스팅(Re-testing and Regression Testing) (K2)

결함이 발견되고 수정된 후에 소프트웨어는 원래의 결함이 성공적으로 제거되었는지 확인하기 위해 다시 테스트되어야 한다. 이것을 확인 테스팅이라고 부른다. 여기서 결함을 수정하는 디버깅(결함의 확인 및 수정)은 개발 활동이며 테스팅 활동으로 보지 않는다.

리그레션 테스팅은 이미 테스트된 프로그램의 테스팅을 반복하는 것으로, 결함 수정 이후 변경의 결과로 새롭게 만들어지거나, 이전 결함으로 인해 발견되지 않았던 또 다른 결함을 발견하는 것이다. 이러한 결함은 테스트 중인 소프트웨어에 존재할 수도 있고, 관련이 있거나 전혀 관련이 없는 다른 소프트웨어 컴포넌트에 있을 수 있다. 리그레션 테스팅은 소프트웨어 또는 그 환경이 변경되었을 때 수행한다. 리그레션 테스팅을 수행하는 범위와 정도는 이전에 정상 동작했던 소프트웨어에서 결함을 발견하지 못해 야기될 수 있는 리스크에 바탕을 둔다.

테스트가 확인 테스팅으로 쓰이거나 리그레션 테스팅을 보조한다면 그 테스트는 반복적인 성향을 갖게 된다.

리그레션 테스팅은 모든 테스트 레벨에서 수행할 수 있으며, 기능, 비기능 그리고 구조적 테스팅을 포함한다. 리그레션 테스트 스위트는 여러 번 반복 수행되며 대개는 서서히 변화하기 때문에 리그레션 테스팅은 자동화에 적합한 후보이다.

2.4 유지보수 테스팅 (K2) | 15분

용어

영향 분석(impact analysis), 유지보수 테스팅(maintenance testing)

배경

소프트웨어 시스템이 일단 배포되면, 일반적으로 수 년 또는 수십 년 정도 서비스된다. 이 기간 동안 시스템과 그 환경은 수정되고, 변경되거나 확장된다. 사전에 출시의 계획을 세우는 것이 유지보수 테스트의 성공 여부에 중대한 영향을 미친다. 계획된 출시와 패치(Hot fixes)는 구별되어야 한다. 유지보수 테스팅은 이미 운영되고 있는 시스템에서 수행하며, 소프트웨어나 시스템이 변경, 단종되었거나 마이그레이션될 때 발생한다.

변경(Modification)에는 (출시 기반으로) 계획된 개선 활동에 의한 변경, 요구사항 변경에 의한 수정과 긴급 변경, 환경의 변경 등이 존재한다. 이러한 변경에는 계획된 OS 또는 DB 업그레이드, 계획된 상용 소프트웨어 업그레이드, OS의 새로 드러난 취약점 패치 등이 있을 수 있다.

(한 플랫폼에서 다른 플랫폼으로 옮겨가는) 마이그레이션을 위한 유지보수 테스팅은 변경된 소프트웨어에 대한 운영 테스트뿐만 아니라, 새로운 환경에서의 운영 테스트도 포함해야 한다. 마이그레이션 테스팅은(변환 테스팅) 운영되고 있는 시스템에 다른 응용 프로그램의 데이터가 마이그레이션될 때도 필요하다.

시스템 단종에 의한 유지보수 테스팅은 데이터를 마이그레이션하는 테스팅을 포함할 수 있으며, 만일 데이터의 보유 기간이 필요하다면 데이터 저장 관련 사항을 테스팅해야 한다.

유지보수 테스팅은 변경된 부분에 대한 테스팅 이외에도 변경되지 않은 시스템 요소에 대한 리그레션 테스팅도 고려한다. 유지보수 테스팅의 범위는 변경 사항의 리스크 및 크기, 기존 시스템의 크기와 관련이 있다. 변경된 내용에 따라서, 유지보수 테스팅은 모든 테스트 유형(Test types)에 대해 모든 테스트 레벨에서 수행할 수 있다. 변경으로 인해 기존 시스템이 어떻게 영향을 받는지 결정하는 것이 영향도 분석(Impact analysis)이며, 얼마나 많은 리그레션 테스팅을 수행할지 결정하는 데 이용된다. 영향도 분석은 리그레션 테스트 스위트를 결정하는 데 사용할 수 있다.

유지보수 테스팅은 명세서(Specifications)가 오래되었거나 없을 경우, 또는 도메인 지식을 가지고 있는 테스터가 없을 경우 수행하기 어려울 수 있다.

제3장 정적 기법 (K2) | 60분

학습목표

3.1 정적 기법과 테스트 프로세스 (K2)

LO-3.1.1 소프트웨어 개발 산출물이 서로 다른 정적 기법에 의해 테스트될 수 있다는 것을 인식한다 (K1)

LO-3.1.2 소프트웨어 개발 산출물에 정적 기법을 적용하는 것에 대한 가치와 중요성에 관해 서술할 수 있다 (K2)

LO-3.1.3 목표, 식별할 결함의 종류 및 소프트웨어 수명주기에서의 역할 등을 기반으로 정적 기법과 동적 기법의 차이를 설명할 수 있다 (K2)

3.2 리뷰 프로세스 (K2)

LO-3.2.1 공식적인 리뷰의 활동, 역할과 책임에 대해 상기한다 (K1)

LO-3.2.2 서로 다른 리뷰(비공식적 리뷰, 기술적 리뷰, 워크쓰루, 인스펙션)의 차이점에 관해 설명할 수 있다 (K2)

LO-3.2.3 성공적인 리뷰를 위한 성공 요소를 설명할 수 있다 (K2)

3.3 도구에 의한 정적 분석 (K2)

LO-3.3.1 정적 분석을 통해 발견할 수 있는 결함과 오류(Error)를 상기하고, 이를 리뷰와 동적 테스팅을 통해 발견할 수 있는 결함 및 오류와 비교할 수 있다 (K1)

LO-3.3.2 정적 분석의 일반적인 장점을 예를 들어 설명할 수 있다 (K2)

LO-3.3.3 정적 분석 도구로 발견할 수 있는 코드와 설계에 존재하는 결함 유형을 열거할 수 있다 (K1)

3.1 정적 기법과 테스트 프로세스 (K2) | 15분

용어

동적 테스팅(dynamic testing), 정적 테스팅(static testing)

배경

소프트웨어의 실행이 필요한 동적 테스팅과는 달리, 정적 테스팅 기법은 코드를 실행하지 않고 코드나 기타 프로젝트 문서를 대상으로 리뷰와 같은 수동적(Manual) 기법이나 자동화 분석(정적 분석)을 기반으로 한다.

리뷰는 코드를 포함하여 소프트웨어 개발 산출물을 테스팅하는 한 가지 방법이며, 동적 테스팅을 실행하기 훨씬 전에 수행할 수 있다. 개발 공정 초기에 리뷰를 통해 발견하는 결함(예: 요구사항에서 발견한 결함)을 수정하는 데 필요한 비용은 실행코드에 대한 동적 테스트를 통해 발견하는 결함의 수정에 필요한 비용에 비해 매우 낮은 경우가 대부분이다.

리뷰는 완전히 수동으로 진행할 수 있지만, 도구의 도움을 받을 수도 있다. 주로 수동으로 진행하는 활동은 개발 산출물을 검사하고 의견을 작성하는 것이다. 요구사항 명세, 설계(디자인) 명세, 코드, 테스트 계획, 테스트 명세, 테스트 케이스, 테스트 스크립트, 사용자 가이드나 웹페이지 등 모든 소프트웨어 개발 산출물은 리뷰의 대상이 될 수 있다.

리뷰의 이점은 조기 결함 발견 및 수정, 개발 생산성 향상, 개발 기간 단축, 테스팅 비용 감소 및 시간 단축, 개발 공정 전체에 걸친 비용 감소, 더욱 적은 결함(품질 향상), 커뮤니케이션 향상을 포함한다. 리뷰를 통해 동적 테스팅에서 발견하기 어려운 개발 산출물(요구사항 명세 등)의 누락(Omissions)과 같은 결함을 발견할 수 있다.

리뷰와 정적 분석, 동적 테스팅은 모두 결함 발견이라는 동일한 목적을 가지고 있다. 이들은 상호 보완적이어서 각각의 기법으로 서로 다른 종류의 결함들을 효과적이고 효율적으로 발견할 수 있다. 정적 기법에서는 동적 테스팅과는 달리 장애(Failure) 자체 보다는 장애의 원인(결함)을 발견한다.

동적 테스팅보다 리뷰를 통해 발견하기 용이한 결함의 종류는 다음과 같다: 표준 위반, 요구사항 결함, 개발 설계(디자인) 결함, 불충분한 유지보수성(Insufficient maintainability), 부정확한 인터페이스 명세.

3.2 리뷰 프로세스 (K2) | 25분

용어

진입 조건(entry criteria), 공식적 리뷰(formal review), 비공식적 리뷰(informal review), 인스펙션(inspection), 메트릭(metric), 사회자(moderator), 동료 리뷰(peer review), 검토자(reviewer), 서기(scribe), 기술적 리뷰(technical review), 워크쓰루(walkthrough)

배경

리뷰는 비공식적인 형태에서부터 체계적인 형태까지 다양한 형태를 갖는다. 비공식적인 리뷰의 특징은 검토자를 위한 문서화된 설명이 없다는 것이고, 체계적인 리뷰의 특징은 팀 참여, 리뷰 결과 문서화, 리뷰 진행 절차가 문서화되어 있다는 점이다. 리뷰 프로세스를 어느 정도까지 공식적/비공식적으로 가져갈지에 대한 형식수준은 개발 프로세스의 성숙도, 테스트 프로세스의 성숙도, 법적 또는 규제적 요구사항(Legal or regulatory requirements) 또는 감사에 대한 필요성(Need for an audit trail)과 관련이 있다.

리뷰 방식은 리뷰를 통해 달성하고자 하는 합의된 목적에 맞는 형태를 취하게 된다(예: 결함 발견, 이해도 증진, 테스터 및 신규 팀원 교육, 합의에 따른 결정과 이에 도달하기 위한 토론).

3.2.1 공식적 리뷰의 활동 (K1)

일반적인 공식적 리뷰의 주요 활동은 다음과 같다.

1. 계획 활동
   - 리뷰 기준 정의
   - 참가 인원 선정
   - 역할 할당
   - 인스펙션과 같은 보다 공식적인 리뷰에서는 진입 및 완료 조건(Entry and exit criteria)을 정의
   - 문서의 어떤 부분이 리뷰 대상인지 선정
   - 보다 공식적인 리뷰인 경우, 진입 조건 확인
2. 시작(킥오프)
   - 문서 배포
   - 리뷰의 목표, 절차 및 문서를 참석자에게 설명
3. 개별 준비
   - 문서를 검토하여 리뷰 회의 준비
   - 잠재적인 결함, 질문, 의견 기록
4. 검토/평가/결과 기록 (리뷰 미팅)
   - 개별 준비 내용을 토의하고 이에 대한 결과를 문서로 기록. 보다 공식적인 리뷰 에서는 상세 회의록(Minutes)을 작성
   - 결함 기록, 결함 처리 방안 건의, 결함에 대한 결정
   - 대면 미팅에서 발견된 이슈를 검토/평가/기록하거나 그룹의 전자 통신 추적
5. 재작업(Rework)
   - 발견된 결함 수정(일반적으로 문서 저자가 수행)

- 결함의 업데이트 사항 기록(공식적인 리뷰인 경우)

6. 후속 처리 확인(Follow-up)
   - 발견된 결함이 처리되었는지를 확인
   - 관련 메트릭(Metric, 측정치)을 수집
   - 리뷰 완료 조건 기준(보다 공식적인 리뷰인 경우)

3.2.2 역할과 책임 (K1)

일반적으로 공식적 리뷰는 다음과 같은 역할을 포함한다.

- 관리자(Manager): 리뷰의 실행 여부를 결정하고, 프로젝트 일정에 리뷰 시간을 할당하고, 리뷰의 목적 달성 여부를 확인하고 승인한다.
- 중재자(Moderator): 문서의 리뷰를 리드한다. 즉, 리뷰를 계획하고, 미팅을 진행하고, 미팅 후 속 조치의 처리 여부 등을 추적하고 관리한다. 그리고 필요할 때 참석자들의 다양한 관점을 중재하며, 많은 경우 리뷰의 성패를 좌우한다.
- 저자(Author): 리뷰 대상 문서(산출물)의 작성자 또는 책임자이다.
- 검토자(Reviewer): 해당 분야의 기술적 또는 비즈니스적 배경을 갖춘 사람으로, 필요한 준비 단계를 거친 후, 리뷰 대상에서 인시던트(결함 포함)를 발견하고 기술하는 사람이다. 검토자는 검사자 또는 인스펙터(Checkers or inspectors)라고도 불린다. 검토자는 리뷰 프로세스에서 다양한 관점과 역할을 대표하도록 선발되어야 하고, 특정 리뷰 미팅에 제한적으로 참여하기보다는 모든 형태의 리뷰 미팅에 참석하여 리뷰 활동을 수행해야 한다.
- 기록자(Scribe or recorder): 리뷰 미팅에서 발견된 모든 이슈, 문제점, 미 해결점 등을 기록하고 문서화한다.

다른 관점에서 소프트웨어 제품 또는 관련 개발 산출물을 보는 것과 체크리스트를 이용하는 것은 리뷰를 좀더 효과적이고 효율적이게 한다. 예를 들어, 사용자, 유지보수자, 테스터 또는 운영자와 같이 다양한 사람들의 관점을 기반으로 한 체크리스트나 전형적인 요구사항의 문제점을 정리해둔 체크리스트는 이전에 발견하지 못한 문제를 발견하는 데 도움이 될 수 있다.

3.2.3 리뷰의 유형 (K2)

하나의 소프트웨어 또는 관련 개발 산출물은 여러 형태로 리뷰될 수 있는데, 만약 한가지 이상의 리뷰 방식을 사용한다면 리뷰의 순서는 다양할 수 있다. 예를 들어, 비공식적 리뷰는 기술적 리뷰 이전에 수행할 수 있고, 인스펙션은 고객과의 워크쓰루 이전에 요구사항 명세를 가지고 수행할 수 있다. 일반적인 리뷰 형식의 주요 특징과 목적은 아래와 같다.

비공식적 리뷰(Informal review)

- 공식적인 절차가 없음
- '페어'(Pair) 프로그래밍에 의한 리뷰 또는 기술 선임자가 설계와 코드를 리뷰하는 것일 수 있음
- 결과는 문서화될 수 있음
- 리뷰하는 사람에 따라 성과가 좌우됨
- 주요 목적: 저렴한 방법으로 일정한 성과 달성

워크쓰루(Walkthrough)

- 작성자에 의한 진행 및 제어

- 시나리오, 예행연습(Dry runs), 동료 집단의 참여 형태를 취할 수 있음
- 시간 및 인원수 등에 제한이 없고 상황에 따라 변경할 수 있는 확장 가능(Open-ended) 세션
  - (선택적) 검토자의 사전 미팅 준비
  - (선택적) 발견 사항의 목록을 포함하여 리뷰 보고서를 준비
- (선택적) 기록자를 지정 (저자는 아님)
- 실무에서는 비공식적일 수 있고 반대로 공식적일 수도 있음
- 주요 목적: 학습, 시스템에 대한 이해 향상 및 결함 발견

기술적 리뷰(Technical Review)

- 동료와 기술 전문가가 참여하는(관리자가 참여할 수 있음), 결함 발견을 위한 문서화되고 정의된 프로세스가 존재함
- 관리자 개입이 없는 동료 검토 형태로 수행할 수 있음
- 이상적으로는 저자가 아닌 중재가가 미팅을 주도함
- 검토자의 미팅 전 사전 준비 단계 필요
- 체크리스트를 사용할 수 있음
- 발견사항 목록, 소프트웨어 제품이 요구사항을 충족하는지 여부, 필요할 경우 발견사항에 대한 건의 사항 등을 포함한 리뷰 보고서 준비
- 실무에서는 비공식적일 수 있고 반대로 공식적일 수도 있음
- 주요 목적: 토론, 의사결정, 대안 평가, 결함 발견, 기술적 문제 해결과 명세서, 계획, 규정 및 표준 준수 여부 확인

인스펙션(Inspection)

- 훈련된 중재자에 의한 진행 및 제어(저자 아님)
- 일반적으로 동료 검사의 형태로 실시
- 역할이 정의되어 있음
- 메트릭을 수집하고 활용함
- 체크리스트와 규칙을 기반으로하는 공식적 프로세스 존재
- 소프트웨어 제품 인수를 위한 정의된 진입 및 완료 조건 기준
- 미팅 전 준비과정 필요
- 발견사항 목록을 포함한 인스펙션 보고서
- 공식적인 후속 처리 확인(프로세스 개선 사항이 포함될 수 있음)
- 글을 읽을 사람이 별도로 지정될 수 있음
- 주요 목적: 결함 발견

워크쓰루, 기술적 리뷰 및 인스펙션은 같은 조직 레벨에서 동등한 그룹 동료들 사이에서 수행될 수 있다. 이러한 종류의 리뷰를 "동료 검토(Peer review)" 라고 부른다.

3.2.4 리뷰의 성공요소 (K2)

리뷰의 성공 요소에는 다음과 같은 것들이 있다:

- 각각의 리뷰가 명확하게 사전에 정의된 목적이 있어야 함
- 리뷰 목적에 적합한 인력이 선택되어야 함

- 테스터는 리뷰에 공헌하는 귀중한 검토자로 여겨지며 제품에 대해 배움으로써 테스트를 좀 더 일찍 준비할 수 있다
- 결함 발견은 언제나 환영받는 분위기이고 결함은 객관적으로 표현되어야 함
- 사람 관련 이슈(People issue)와 심리적인 측면이 고려되어야 함(예: 작성자가 리뷰를 통해 긍정적인 경험을 하도록 한다).
- 리뷰는 모든 참여자가 서로 믿는 환경에서 진행되어야 함; 리뷰 결과를 가지고 참여자들을 평가해서는 안 된다
- 효과적이고 효율적인 결함 발견을 위해 필요할 때 체크리스트 및 역할 분담 활용
- 리뷰 기법이 소프트웨어 중간 산출물과 검토자의 유형과 레벨에 맞게 적절하게 적용된다
- 리뷰 기법에 대한 교육 훈련 제공. 특히, 인스펙션과 같이 보다 공식적인 기법에 대해서는 교육 훈련 제공이 필수적임
- 경영진이 적극적으로 리뷰 프로세스를 지원해야 함(예: 프로젝트에서 리뷰 기법 적용에 충분한 일정 할애)
- 학습과 프로세스 개선에 대한 강조

3.3 도구에 의한 정적 분석 (K2) | 20분

용어

컴파일러(compiler), 복잡도(complexity), 제어 흐름(control flow), 데이터 흐름(data flow), 정적 분석(static analysis)

배경

정적 분석의 목적은 소프트웨어의 소스코드와 모델에서 결함을 발견하는 것이다. 소프트웨어의 코드를 실행하여 수행하는 동적 테스팅에 비해, 정적 분석은 조사 대상 소프트웨어를 실제적으로 실행하지 않는 상태에서 도구의 지원으로 수행하는 것이다. 정적 분석은 동적 테스팅으로 찾기 힘든 결함을 발견한다. 리뷰와 마찬가지로 정적 분석은 장애(Failures)보다는 결함(Defects)을 발견한다. 정적 분석 도구는 프로그램 코드를 분석(제어 흐름이나 데이터 흐름 분석 등)하는 것은 물론, HTML이나 XML과 같이 생성된 결과물도 분석한다.

정적 분석의 가치는 아래와 같다:

- 테스트 실행 전에 조기 결함 발견
- 높은 복잡도(Complexity) 측정치와 같은 메트릭을 계산하여 코드와 설계의 의심스러운 부분에 대한 조기 경보
- 동적 테스팅으로는 발견하기 어려운 결함 발견
- 소프트웨어 모델상의 의존도와 불일치성(Dependencies and inconsistencies) 발견
- 코드와 설계의 유지보수성 향상
- 결함 예방 가능

정적 분석 도구를 통해 발견되는 전형적인 결함은 아래와 같다.

- 정의되지 않은 값으로 변수 참조
- 모듈과 컴포넌트 간에 일관되지 않은 인터페이스
- 사용되지 않는 변수
- 사용되지 않는 코드(Dead Code)
- 누락 및 에러 로직(Logic) (잠재적인 무한 루프)
- 지나치게 복잡한 구조
- 코딩 표준 위반
- 보안 취약성
- 코드와 소프트웨어 모델의 구문 규칙(Syntax) 위반

정적 분석 도구는 컴포넌트 테스팅과 통합 테스팅 동안이나 형상 관리 도구에 코드를 체크인할 때 주로 개발자에 의해 사용되고(사전에 정의된 규칙이나 코딩 표준을 준수하는지 확인), 소프트웨어 모델링하는 동안에는 설계자에 의해 사용된다. 도구를 효과적으로 사용하기 위해서는 정적 분석 도구가 생성하는 대량의 경고 메시지를 적절히 관리하는 것이 필요하다.

컴파일러도 메트릭의 계산을 포함하는 정적 분석 기능의 일부를 지원할 수 있다.

제4장 테스트 설계 기법 (K4) | 285분

학습목표

4.1 테스트 개발 프로세스 (K3)

LO-4.1.1 테스트 설계 명세, 테스트 케이스 명세, 테스트 프로시저 명세를 구분한다 (K2)

LO-4.1.2 테스트 조건, 테스트 케이스, 테스트 프로시저의 용어를 구분한다 (K2)

LO-4.1.3 테스트 케이스의 품질을 요구사항 및 기대 결과와의 명확한 추적성을 기반으로 평가할 수 있다 (K2)

LO-4.1.4 테스트 케이스를 테스터의 지식수준에 맞는 구체적으로 구조화된 테스트 절차 사양서로 작성할 수 있다 (K3)

4.2 테스트 설계 기법의 종류 (K2)

LO-4.2.1 명세 기반(블랙박스)과 구조 기반(화이트박스)의 접근 방식의 유용성을 상기하고 각각의 대표적인 기법을 나열할 수 있다 (K1)

LO-4.2.2 명세 기반과 구조 기반 그리고 경험 기반 기법들의 특성과 공통점 및 차이점을 설명할 수 있다 (K2)

4.3 명세 기반 / 블랙박스 기법 (K3)

LO-4.3.1 주어진 소프트웨어 모델로부터 동등 분할, 경계값 분석, 결정 테이블과 상태전이 다이어그램/테이블을 활용해서 테스트 케이스를 작성할 수 있다 (K3)

LO-4.3.2 네 종류의 테스팅 기법에 대해 각각의 목적, 어떤 테스트 레벨과 종류에 적합한지 여부와 커버리지를 어떻게 측정하는지 설명할 수 있다 (K2)

LO-4.3.3 유즈케이스 테스팅의 개념과 이점을 설명할 수 있다 (K2)

4.4 구조 기반 / 화이트박스 기법 (K4)

LO-4.4.1 코드 커버리지의 개념과 가치를 설명할 수 있다 (K2)

LO-4.4.2 구문 및 결정 커버리지의 개념을 설명하고, 이 개념들이 컴포넌트 테스팅 이외의 테스트 레벨에서도 활용될 수 있는 이유를 설명할 수 있다 (예: 시스템 레벨의 비즈니스 흐름) (K2)

LO-4.4.3 주어진 제어 흐름으로부터 구문 및 결정 테스트 설계 기법을 활용하여 테스트 케이스를 작성할 수 있다 (K3)

LO-4.4.4 정의된 완료 조건을 기반으로 구문 및 결정 커버리지의 완결성을 평가할 수 있다 (K4)

4.5 경험 기반 기법 (K2)

LO-4.5.1 일반적인 결함에 대한 직관, 경험, 지식을 기반으로 테스트 케이스를 작성하는 이유를 상기할 수 있다 (K1)

LO-4.5.2 경험 테스팅 기법과 명세 기반 테스팅 기법을 비교할 수 있다 (K2)

4.6 테스트 기법의 선택 (K2)

LO-4.6.1 주어진 정황, 테스트 베이시스, 해당되는 모델 및 소프트웨어 특성에 따라 테스트 설계 기법을 분류할 수 있다 (K2)

4.1 테스트 개발 프로세스 (K3) | 15분

용어

테스트 케이스 명세(test case specification), 테스트 설계(test design), 테스트 실행 스케줄(test execution schedule), 시험 절차 명세(test procedure specification), 테스트 스크립트(test script), 추적성(traceability)

배경

이번 장에 기술된 테스트 개발 프로세스는 다양한 방식으로 진행될 수 있다. (아래에서 설명하는 것처럼) 매우 비공식적일 수 있고, 매우 공식적일 수도 있다. 공식성 또는 정형성(Formality)의 정도는 테스팅과 개발 프로세스의 성숙도, 시간적 제약, 안전 및 법규 요구사항, 참여 인원 등을 포함하는 테스트 정황(Context)에 따라 달라진다.

테스트 분석 과정에서 무엇을 테스트할지 결정하기 위해, 즉 테스트 조건(Test Condition)을 식별하기 위해, 테스트 베이시스(Test basis)를 분석한다. 테스트 조건은 하나 이상의 테스트 케이스로 확인 가능한 항목 또는 이벤트이다. (예: 트랜잭션(Transaction), 품질 특성 또는 구조적 요소)

테스트 조건과 명세 및 요구사항 사이에 추적성을 설정하는 것은, 요구사항이 변경되었을 때의 효과적인 영향도 분석과 일련의 테스트에 의한 요구사항 커버리지(Requirement coverage)의 결정을 가능하게 한다. 테스트를 분석하는 동안 상세한 테스트 기법(접근법)은 테스트 설계 기법을 선택하기 위해 식별된 리스크를 기준으로 선정하여 구현한다. (리스크 분석에 대한 보다 자세한 내용은 5장을 참고)

테스트 설계 과정에서 테스트 설계 기법을 이용하여 테스트 케이스와 테스트 데이터를 설계하고 명세화한다. 테스트 케이스는 특정 테스트 목적 또는 테스트 조건을 커버하기 위해 정의한 입력값의 묶음, 실행 사전조건, 기대 결과(Expected result)와 실행 사후조건으로 구성된다. 표준 문서인 '소프트웨어 테스트 문서화 표준(Standard for Software Test Documentation)' (IEEE STD 829-1998)은 테스트 설계 명세(테스트 상황 포함)와 테스트 케이스 명세 내용을 기술한다.

테스트 케이스 명세의 일부로 생성되어야 하는 기대 결과는 결과값, 데이터나 상태의 변화, 다른 테스트 결과로 구성된다. 만약 기대 결과가 정의되어 있지 않다면, 그럴듯하지만 실제로는 틀린 결과가 올바른 것으로 판정받을 수 있다. 이와 같이 테스트 실행 결과가 올바른 것인지 판정할 수 없거나 어려울 수 있다는 것을 감안하여 원칙적으로는 테스트 실행 전에 기대 결과가 정의되어 있어야 한다.

테스트 구현이 진행되는 동안 테스트 케이스를 개발 및 구현하고, 우선순위를 선정하고 배치하여 테스트 프로시저 명세서(Test procedure specification)를 만든다(IEEE STD 829-1998). 테스트 프로시저는 테스트 실행을 위한 동작 순서이다. 만약 테스트 실행 도구를 이용해 테스팅한다면, 테스트 대상 소프트웨어를 동작시키는 순서를 테스트 스크립트(자동화된 테스트 프로시저)에 기술한다.

다양한 테스트 프로시저와 자동화된 테스트 스크립트로 구성된 테스트 실행 스케줄에는 테스트 프로시저와 자동화된 테스트 스크립트의 실행 순서가 정의되어 있다. 테스트 실행 스케줄은 리그레션 테스트 여부나 우선순위, 기술적/논리적 종속 관계와 같은 요소를 고려하여 결정한다.

4.2 테스트 설계 기법의 종류 (K2) | 15분

용어

블랙박스 테스트 설계 기법(black-box test design technique), 경험 기반 테스트 설계 기법(experience-based test design technique), 테스트 설계 기법(test design technique), 화이트박스 테스트 설계 기법(white-box test design technique)

배경

테스트 설계의 목적은 테스트 조건과 테스트 케이스를 식별하는 것이다.

테스트 기법을 블랙박스와 화이트박스로 구분하는 것은 전통적인 방법이다. 블랙박스 기법(명세 기반 기법과 경험 기반 기법을 포함함)은 테스트 대상의 내부 구조(코드)를 참조하지 않고 테스트 베이시스 그리고 개발자와 테스터, 사용자들의 경험을 바탕으로 기능적 혹은 비기능적 테스트 케이스를 도출하고 선택하는 방법이다. 반면 화이트박스 기법(구조 기반 기법)은 컴포넌트(단위) 또는 소프트웨어(시스템)의 구조(코드)를 중심으로 테스트 케이스를 도출하는 방법이다. 또한, 블랙박스 테스트와 화이트박스 테스트는 경험 기반 기법과 함께 개발자, 테스터, 사용자의 경험을 바탕으로 무엇을 테스트할지 결정하는 데 사용될 수 있다.

어떤 방식으로 분류하건 하나의 범주에 명확하게 해당되는 기법들도 있지만, 하나 이상의 범주에 속하는 기법들도 존재한다.

이 실라버스에서는 명세 기반 기법을 블랙박스 기법으로 보고, 구조기반 기법을 화이트박스 기법으로 간주한다. 또한, 경험 기반 테스트 설계 기법에 관해서도 언급하고 있다.

명세 기반 테스트 설계 기법의 일반적인 특징은 다음과 같다.

- 해결할 문제를 명세화하는데 공식적이거나 비공식적인 모델을 사용한다
- 테스트 케이스는 이러한 모델에서 시스템적으로 도출하는 것이 가능하다

구조 기반 테스트 설계 기법의 일반적인 특징은 다음과 같다.

- 소프트웨어 구현 정보를 기반으로 테스트 케이스를 도출한다 (예: 코드와 상세 설계 정보)
- 개발된 테스트 케이스를 가지고 소프트웨어의 커버리지를 측정할 수 있으며, 커버리지를 높이기 위해 추가적인 케이스를 시스템적으로 도출해낼 수 있다

경험 기반 테스트 설계 기법의 일반적인 특징은 다음과 같다.

- 테스트 관련 인력의 지식이나 경험에서 테스트 케이스를 도출한다
- 소프트웨어, 소프트웨어의 사용법 및 환경에 관한 테스터, 개발자, 사용자 및 기타 관련자들의 지식은 사용될 수 있는 정보의 한 종류이다
- 소프트웨어의 자주 발생하는 결함이나 결함의 분포와 관련 있는 지식은 사용될 수 있는 정보의 또 다른 종류이다

4.3 명세 기반/블랙박스 기법 (K3) | 150분

용어
경계값 분석(boundary value analysis), 결정 테이블 테스팅(decision table testing), 동등 분할(equivalence partitioning), 상태전이 테스팅(state transition testing), 유즈케이스 테스팅(use case testing)

4.3.1 동등 분할(Equivalence Partitioning) (K3)

동등 분할 테스팅에서는 소프트웨어나 시스템의 입력값은 입력의 결과로 나타날 결과값이 동일한 경우 하나의 그룹(클래스)으로 간주한다. 그리고 이러한 그룹 내의 입력값은 내부적으로 같은 방식으로 처리됨을 가정한다. 동등 분할 클래스는 유효한 입력 데이터, 즉 허용할 데이터와 잘못된 데이터, 거부할 데이터 모두에 대해 존재할 수 있다. 입력값(Input) 이외에도 출력값(Outputs), 내부값(Internal values), 시간관련값(Time-related values, 이벤트 이전과 이후), 인터페이스 파라미터(Interface parameters, 예: 통합 테스트 동안 테스트 대상이 되는 통합된 컴포넌트)에 대해서도 동등 분할을 정의할 수 있다. 테스트는 모든 유효/비유효 분할을 커버하도록 설계될 수 있다. 동등 분할 테스트는 모든 레벨에 적용할 수 있다.

동등 분할은 입력값 및 출력값에 대한 특정 커버리지를 달성하기 위해 사용할 수 있다. 이 기법은 사람에 의한 입력, 인터페이스를 통한 시스템으로의 입력, 통합 테스팅에서 인터페이스 파라미터에 적용할 수 있다.

4.3.2 경계값 분석(Boundary Value Analysis) (K3)

동등 분할의 경계 부분에 해당되는 입력값에서 결함이 발견될 확률이 동등 분할 내의 다른 값에서의 확률보다 경험적으로 높기 때문에 결함을 방지하기 위해 경계값까지 포함하여 테스트하는 기법이다. 해당 분할 영역의 최대값과 최소값은 그 영역의 경계값이 된다. 유효한 분할 영역의 경계값은 유효 경계값이 되며, 비 유효한 분할 영역의 경계값은 비 유효 경계값이 된다. 테스트는 유효 경계값과 비 유효한 경계값 모두 커버할 수 있도록 설계할 수 있다. 테스트 케이스 설계 시에 각 경계값이 선택된다.

경계값 분석은 모든 테스트 레벨에 적용할 수 있다. 상대적으로 적용하기 쉽고 결함 발견율이 높다. 구체적인 명세서가 있으면 신경 써야 하는 경계값을 구분하기 쉽다.

경계값 분석 기법은 종종 동등 분할 또는 다른 블랙박스 설계 기법의 확장으로 여겨지며, 동등 분할과 동일한 방식으로 커버리지를 보장한다. 또한, 스크린에 사용자가 입력하는 값에 대한 동등 클래스를 결정할 때 사용될 수 있다. 예를 들면, 트랜잭션 속도 요구사항과 같은 시간 범위(Time Ranges) 또는 테이블 크기와 같은 테이블 범위(Table ranges)에 적용할 수 있다.

4.3.3 결정 테이블 테스팅(Decision Table Testing) (K3)

결정 테이블은 논리적인 조건이나 상황(Conditions)을 구현하는 시스템 요구사항을 도출하거나 내부 시스템 디자인을 문서화하는 매우 유용한 도구이다. 이것은 시스템이 구현해야 하는 복잡한 비즈니스 규칙(Business rules)을 문서로 만드는 데 사용된다. 결정 테이블을 작성할 때 명세서를 분석하고, 컨디션과 시스템의 조건은 식별된다. 입력조건과 동작은 참(True)과 거짓(False)으로 주로 표현된다. 결정 테이블은 동작을 유발시키는 조건 또는 상황(Triggering conditions – 주로 모든 입력 조건에 대한 참과 거짓의 조합으로 나타남) 그리고 각 해당 조합에 대한 예상 결과까지 포함한다. 테이블의 각 컬럼은 비즈니스 규칙과 대응 관계를 갖는다. 해당 비즈니스

규칙(테이블의 각 칼럼)은 유일한 조건의 조합(Combination of conditions)을 정의하고, 조건의 조합은 해당 비즈니스 규칙과 연관된 동작을 수행하게 한다. 실무에서 결정 테이블을 사용할 때 일반적인 커버리지는 각 테이블 컬럼당 적어도 하나의 테스트 케이스를 생성하는 것이다. 이는 일반적으로 가능한 모든 동작을 유발시키는 조건의 조합을 커버한다.

결정 테이블은 테스팅 기간 동안 결정 테이블을 사용해야만 발견할 수 있는 조건들의 조합을 찾아내고 생성할 수 있다. 해당 기법은 소프트웨어의 동작이 여러 가지 논리적 조건에 의존적인 모든 경우에 적용할 수 있다.

4.3.4 상태전이 테스팅(State Transition Testing) (K3)

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 상태전이 다이어그램(State transition diagram)으로 표현할 수 있다. 상태전이 다이어그램을 통해 테스트 엔지니어는 소프트웨어 또는 시스템을 상태 사이의 관계 즉, 상태 간의 전이, 상태를 변화시키는 이벤트와 입력값, 상태의 변화로 유발되는 동작 등으로 파악한다. 이때 테스트 대상 시스템이나 객체의 상태는 개별적으로 식별 가능하고 유한한 개수로 표현된다.

상태전이 다이어그램을 테이블 형태로 전환한 상태 테이블(또는 상태-이벤트 테이블)은 상태와 이벤트 간의 관계를 보여주고, 개연성은 있지만, 요구사항과 맞지 않는 전이를 찾아낼 수도 있다.

상태전이 테스팅을 통해 전형적인 상태의 순서를 커버하는 방식, 모든 상태를 커버하는 방식, 모든 상태전이를 실행하는 방식, 특정한 상태전이 순서를 실행하는 방식 또는 불가능한 상태전이를 테스트하는 방식으로 테스트를 설계하는 것이 가능하다.

상태전이 테스팅은 일반적으로 임베디드 소프트웨어 산업 분야나 기술적으로 자동화가 필요한 부분에서 사용된다. 뿐만 아니라 특정한 상태를 갖는 비즈니스 객체 모델링이나(예: 인터넷 응용 프로그램 또는 비즈니스 시나리오) 화면-대화창 흐름을 테스팅할 때도 적절히 적용할 수 있다.

4.3.5 유즈케이스 테스팅(Use Case Testing) (K2)

테스트는 유즈케이스로부터 도출 가능하다. 유즈케이스는 액터(사용자 혹은 시스템)와 엑터 사이의 상호 작용을 표현하고, 해당 상호 작용은 시스템 유저 또는 고객에게 결과값을 제공한다. 유스케이스는 원칙적인 레벨(비즈니스 유즈케이스, 비기술(Technology-free), 비즈니스 프로세스 레벨) 또는 시스템 레벨(시스템 기능 수준에서 시스템 유즈케이스)에서 도출될 수 있다. 각각의 유즈케이스는 그 유즈케이스를 성공적으로 수행하기 위한 전제 조건(Preconditions)을 가지고 있다. 또한, 각각의 유즈케이스는 임무를 완수한 후 후속조건(Postconditions – 관찰 가능한 결과와 시스템의 마지막 상태)를 가지면서 종료된다. 유즈케이스는 대게 주류 시나리오 또는 기본 흐름(Mainstream scenario, Basic or Main flows)과 대체 흐름(Alternative branches or Alternative flows)으로 구성되어 있다.

유즈케이스는 시스템을 실제 사용하는 방식에 기반하여 "프로세스 흐름"을 기술하고 있다. 따라서 유즈케이스를 기반으로 생성한 테스트 케이스는 시스템이 실제 사용되는 프로세스 흐름에서 결함을 발견하는 데 상당히 유용하다. 유즈케이스는 고객이나 사용자 그룹이 참여하는 인수테스트(Acceptance Test)를 설계할 때 매우 유용하다. 또한, 개별적인 컴포넌트 테스팅에서는 발견하기 힘든 서로 다른 컴포넌트 사이의 상호 작용과 간섭으로 발생하는 통합 결함을 찾는 데 도움이 된다. 유즈케이스 테스트 설계 기법은 다른 명세 기반 테스트 기법과 혼용해서 사용할 수 있다.

4.4 구조 기반/화이트박스 기법 (K4) | 60분

용어

코드 커버리지(code coverage), 결정 커버리지(decision coverage), 구문 커버리지(statement coverage), 구조기반 테스팅(structure-based testing)

배경

구조 기반 테스팅 및 화이트박스 테스팅은 아래 예시와 같이 소프트웨어나 시스템의 구조(Structure)를 중심으로 테스팅하는 것이다.

- 컴포넌트 레벨: 소프트웨어 컴포넌트의 구조(예: 구문(Statements), 결정(Decisions), 분기문(Branches) 또는 서로 다른 경로(Distinct paths))
- 통합 레벨의 구조: 한 모듈이 다른 모듈을 호출하는 관계를 도식화한 콜 트리(Call tree) 등
- 시스템 레벨의 구조: 메뉴 구조, 비즈니스 프로세스 혹은 웹페이지 구조 등

이번 장에서는, 구문, 분기와 결정을 기반으로 코드 커버리지를 달성하기 위한 코드와 관련된 3가지 구조 기반 테스트 설계 기법을 논의한다. 결정 테스팅을 수행할 때 제어 흐름도를 각 결정문(Decision)의 대체 흐름을 가시화하기 위해 사용할 수 있다.

4.4.1 구문 테스팅과 커버리지(Statement Testing and Coverage) (K4)

컴포넌트 테스팅에서 구문 커버리지는 테스트 케이스 스위트(Suite, 묶음)에 의해 실행한 구문이 몇 퍼센트인지를 측정하는 것이다. 구문 테스팅 기법은 구문 커버리지를 늘리기 위해 특정 구문을 테스트하는 테스트 케이스를 도출하는 것이다.

구문 커버리지는(설계되거나 실행된) 테스트 케이스가 커버하는 실행 가능 구문들의 수를 테스트중인 코드에 포함된 모든 실행 가능 구문의 수로 나눈 값으로 표시한다.

4.4.2 결정 테스팅과 커버리지(Decision Testing and Coverage) (K4)

분기문(Branch) 테스팅과 관련된 결정 커버리지는 테스트 케이스 스위트(Suite, 묶음)에 의해 실행된 조건문 분기(if구문의 참 혹은 거짓)가 몇 퍼센트인지를 측정하고 평가하는 것이다. 결정 테스팅 기법은 특정 조건문의 분기를 테스트하는 테스트 케이스를 도출하는 것이다. 분기문은 코드의 결정 포인트(Decision points)에서 발생하고 컨트롤이 코드의 서로 다른 위치로 이동하는 것을 보여준다.

결정 커버리지는(설계하거나 실행한) 테스트 케이스가 커버하는 결과값들의 수를 테스트중인 코드의 모든 가능한 결과값의 수로 나눈 값으로 표시한다.

결정 테스팅은 결정 포인트(Decision points)에 해당하는 제어흐름을 다루므로 제어흐름 테스팅의 한가지 형태이다. 결정 커버리지는 구문 커버리지보다 강력하여 100% 결정 커버리지를 달성할 경우, 100% 구문 커버리지를 달성함을 보장한다. 그러나 반대의 경우는 성립하지 않는다.

4.4.3 그 외 구조 기반 기법 (K1)

조건 커버리지나 다중 조건 커버리지와 같이 결정 커버리지 보다 강력한 레벨의 구조적 커버리지가 존재한다.

커버리지의 개념은 다른 테스트 레벨에도 적용할 수 있다. (예: 통합 테스팅에서의 모듈, 컴포넌트, 클래스 커버리지는 테스트 케이스 스위트에 의해 몇 퍼센트의 모듈, 컴포넌트, 클래스가 실행했는지를 나타낸다)

코드의 구조를 테스팅할 때는 도구의 지원이 유용하다.

4.5 경험 기반 기법 (K2) | 30분

용어

탐색적 테스팅(exploratory testing), (결점) 공격(fault attack)

배경

경험 기반 테스팅은 이전에 테스터가 다루었던 유사 응용 프로그램이나 기술에서의 경험, 직관, 테스터의 기술 능력으로부터 테스트 케이스를 추출해낸다. 체계적인 기법을 보강하기 위해, 특히 공식적인 기법을 적용한 이후에 추가적으로 경험 기반 기법을 적용하는 것은 공식적인 기법으로 다루기 어려운 특별한 테스트 케이스를 찾아내고 실행하는 데 유용하다. 그러나 이 기법에서는 테스터의 경험에 따라 효율성 및 효과성의 정도가 매우 달라질 수 있다.

일반적으로 사용되는 경험 기반 기법은 오류 추정(Error guessing)이다. 대체로 테스터는 경험에 기반하여 결함을 예측한다. 오류 추정 기법을 구조적으로 사용하려면 가능한 결함(Defects)을 모두 나열하고 이런 유형의 결함 또는 오류를 공격할 수 있도록 테스트를 설계해야 한다. 이러한 체계적인 접근법을 결점 공격(Fault attack)이라고 부른다. 이러한 결함이나 장애 목록 작성에는 경험이나 기존 장애 및 결함에 대한 데이터, 또 소프트웨어 실패에 대한 일반적인 지식이 활용된다.

탐색적 테스팅(Exploratory testing)은 테스트 설계, 테스트 수행, 테스트 기록, 그리고 학습이 동시에 진행되는 것으로, 테스트 목적을 담고 있는 테스트 차터(Test charter)를 기반으로 제한된 시간 내에 수행하는 것이다. 탐색적 테스팅은 명세가 거의 없거나 부적절할 경우와 시간 압박이 극심할 경우, 또는 보다 공식적인 테스팅을 강화/보완할 때 유용한 접근법이다. 탐색적 테스팅은 테스트 프로세스 검증을 함으로써 가장 심각한 결함이 발견되었다는 확신을 갖도록 해주는 역할을 한다.

4.6 테스트 기법의 선택 (K2)	15분

용어
용어 없음

배경
어떤 테스트 기법을 사용할지 결정하는 데 고려해야 할 요인은 시스템의 유형, 강제적인 표준 또는 법적 기준의 존재 여부, 고객 또는 계약상의 요구사항, 리스크 수준, 리스크 유형, 테스트 목표, 문서의 존재 유무, 테스터의 지식 수준, 시간과 예산, 테스트 레벨, 개발 생명 주기, 유즈케이스, 상태 다이어그램 등 모델 존재 유무, 발견된 결함 유형에 대한 이전의 경험 등이 있다.

일부기법은 특정한 상황과 테스트 레벨에 좀 더 적합할 수 있으나 대부분의 기법은 모든 테스트 레벨에 적합하게 사용할 수 있다.

테스트 케이스를 작성할 때, 테스터는 일반적으로 테스트 대상에 대한 적절한 커버리지를 보장하기 위해서 프로세스, 법칙(Rule) 및 데이터 기반 기법(Data-driven techniques) 등의 테스트 기법을 적절히 조합해서 사용한다.

제5장 테스트 관리 (K3) | 170분

학습목표

5.1 테스트 조직 (K2)

LO-5.1.1 테스트 조직의 독립성에 대한 중요성을 인지한다 (K1)
LO-5.1.2 조직 내에서 독립적 테스트 팀 운영에 대한 장점과 단점을 설명할 수 있다 (K2)
LO-5.1.3 테스트 팀을 구축할 때 고려해야 할 팀구성원의 차이를 인지한다 (K1)
LO-5.1.4 테스트 리더와 테스터의 전형적인 임무를 상기한다 (K1)

5.2 테스트 계획과 추정 (K3)

LO-5.2.1 테스트 계획 활동의 다양한 단계와 목표를 인지한다 (K1)
LO-5.2.2 '소프트웨어 테스트 문서 표준(IEEE 표준 829-1998)'에 정의된 테스트 계획, 테스트 설계 명세 및 테스트 절차 문서의 목적과 내용을 요약할 수 있다 (K2)
LO-5.2.3 분석적, 모델 기반, 방법론적, 프로세스/표준 준수, 동적/휴리스틱, 자문 및 리그레션 회피 등과 같이 개념적으로 서로 다른 테스트 방법을 구별할 수 있다 (K2)
LO-5.2.4 테스트 계획과 실제 테스트 수행 계획 간의 차이를 구별할 수 있다 (K2)
LO-5.2.5 우선순위, 기술적 그리고 논리적 의존성을 고려한 테스트 케이스 실행 일정을 작성할 수 있다 (K3)
LO-5.2.6 테스트 계획 시 고려되어야 할 테스트 준비 및 수행 활동을 나열할 수 있다 (K1)
LO-5.2.7 테스트 노력에 영향을 주는 일반적인 요소들을 상기할 수 있다 (K1)
LO-5.2.8 개념적으로 다른 2개의 추정 접근법을 구별할 수 있다: 메트릭 기반의 접근법과 전문가 기반의 접근법 (K2)
LO-5.2.9 특정 테스트 케이스 묶음 및 테스트 레벨에 대한 적절한 진입과 완료 조건을 인식/정의할 수 있다 (예: 통합 테스트, 인수테스트, 사용성 테스트를 위한 테스트 케이스) (K2)

5.3 테스트 경과 모니터링과 제어 (K2)

LO-5.3.1 테스트 준비와 실행을 모니터링하기 위해 사용하는 일반적인 메트릭을 상기할 수 있다 (K1)
LO-5.3.2 테스트 보고와 테스트 제어를 위한 테스트 메트릭(예: 발견 및 수정된 결함, 성공적으로 완수한 테스트, 실패한 테스트)을 목적과 사용법에 따라 비교하고 설명할 수 있다 (K2)
LO-5.3.3 '소프트웨어 테스트 문서 표준(IEEE 표준 829-1998)'에 정의된 테스트 요약 보고서의 목적과 내용을 요약할 수 있다 (K2)

5.4 형상 관리 (K2)

LO-5.4.1 테스팅에서 형상 관리의 역할을 이해하고 설명할 수 있다 (K2)

5.5 리스크와 테스팅 (K2)

LO-5.5.1 프로젝트 이해관계자에게 목적을 달성하는 데 위협이 될 수 있는 리스크에 대해 설명할 수 있다 (K2)

LO-5.5.2 리스크의 수준은 장애 발생 가능성과 (발생했을 경우) 장애로 인한 영향도로 결정된다는 것을 기억할 수 있다 (K1)

LO-5.5.3 프로젝트 리스크와 제품 리스크를 구별할 수 있다 (K2)

LO-5.5.4 일반적인 프로젝트 리스크와 제품 리스크를 식별할 수 있다 (K1)

LO-5.5.5 리스크 분석 및 리스크 관리가 어떻게 테스트 계획에 활용될 수 있는지 예를 들어 설명할 수 있다 (K2)

5.6 인시던트 관리 (K3)

LO-5.6.1 '소프트웨어 테스트 문서 표준(IEEE 표준 829-1998)'에 정의된 인시던트 보고서의 내용을 인식할 수 있다 (K1)

LO-5.6.2 테스트에서 관찰된 장애에 대한 인시던트 보고서를 작성할 수 있다 (K3)

5.1 테스트 조직 (K2) | 30분

용어
테스터(tester), 테스트 리더(test leader), 테스트 매니저(test manager)

5.1.1 테스트 조직과 독립성 (K2)

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

- 독립적이지 않은 테스터, 즉 개발자가 자신의 코드를 직접 테스트함
- 개발팀 내 독립적 테스터
- 프로젝트 관리자 혹은 상위 관리자에게 직접 보고하는 조직 내 독립적 테스트 팀 또는 그룹
- 비즈니스 조직 또는 사용자 커뮤니티에서 채용한 독립적 테스터
- 사용성, 보안성, 인증 테스터(소프트웨어 제품이 표준과 규정을 준수하는지 입증하는 전문가)처럼 특정 테스트 분야를 전문으로 하는 독립적 테스트 전문가
- 아웃소싱(Out-sourcing)한 또는 조직 외부의 독립적 테스터

거대하고 복잡한 또는 안전 최우선(Safety critical) 프로젝트를 테스팅할 경우, 일반적으로 다수의 테스트 레벨에 걸쳐 테스팅하는 것이 최선의 방법이 될 수 있으며, 일부 또는 모든 테스트 레벨을 독립적인 테스터가 수행하게 된다. 개발 담당자는 테스팅(특히 하위 레벨 테스팅)에 효율적으로 참여할 수 있으나 객관성이 부족하여 효과적인 면에서는 한계점을 갖는다. 독립적 테스터는 테스트 프로세스와 규칙의 필요성을 강권하고 그 일부 또는 전부에 대해 정의하는 권한을 가질 수 있으나, 명확한 관리권한 위임(Management mandate) 하에서만 프로세스 관련 역할을 수행해야 한다.

독립성(테스팅 조직을 독립적으로 운영하는 것)의 장점은 다음과 같다.
- 독립적 테스터는 편향적이지 않으며, 개발자와는 다른 시각에서 다른 종류의 결함을 본다
- 독립적 테스터는 시스템의 명세와 구현 단계에서 개발 관련자가 만든 가정을 검증할 수 있다

독립성의 단점은 다음과 같다.
- 철저히 독립체로 간주되면 개발팀과 개발 및 제품 관련 정보로부터 고립될 수 있다
- 개발자가(자신들이 개발하는 제품의) 품질에 대한 책임감을 잃을 수 있다
- 독립적인 테스터는 병목 현상 또는 출시 지연에 대한 비난을 받을 수 있다

테스팅 작업은 특정 테스팅 역할을 부여받은 인력뿐만 아니라 프로젝트 관리자, 품질 관리자, 비즈니스와 해당 분야 전문가, 인프라 또는 IT 운영자 등 다른 역할을 가진 인력에 의해서도 수행할 수 있다.

5.1.2 테스트 리더와 테스터의 역할 (K1)

이 실라버스에서는 테스트 리더와 테스터에 대해서만 살펴보도록 한다. 해당 인력이 수행하는 활동과 임무는 프로젝트, 제품 정황(Context), 역할을 수행하는 인력, 해당 조직에 따라 달라질 수 있다.

테스터 리더는 테스트 매니저 또는 테스트 조정자(Coordinator)라고도 한다. 프로젝트 관리자, 개발 관리자, 품질

보증 관리자 또는 테스트 그룹 관리자가 테스트 리더의 역할을 수행할 수 있다. 규모가 큰 조직에서는 테스트 리더와 테스트 매니저가 동시에 존재할 수도 있다. 1.4장에 명시된 것과 같이 일반적으로 테스트 리더는 테스팅 활동과 업무를 계획하고 모니터링하고 제어하는 역할을 수행한다.

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
- 테스팅 측면에서 통합 계획 활동 등 다른 프로젝트 활동에 기여한다
- 정황을 감안하고 테스트 목적과 리스크를 이해하면서 테스트 계획을 수립한다. 테스트 계획은 테스트 접근법을 선택하고, 테스팅 시간 및 노력과 비용을 추정하고, 자원을 획득하고, 테스트 레벨 및 테스트 사이클과 목표를 정의하고, 인시던트 관리를 계획하는 활동을 포함한다
- 테스트 명세(설계), 준비, 구현과 실행, 테스트 결과 모니터링 및 완료 조건 점검의 시작을 주도한다
- 테스트 결과와 진척(때때로 상태 리포트로 문서화되어 있음)을 감안하여 계획 활동을 조정하고 문제점을 보완하기 위해 필요한 행동을 취한다
- 추적성(Traceability) 확보를 위해 적절한 테스트웨어(Testware) 형상관리(Configuration management) 방안을 구상한다
- 테스트 진척을 측정하는 메트릭(Metric)은 물론, 테스팅 자체의 품질과 대상 제품의 품질을 평가하는 적절한 메트릭을 도입한다
- 자동화되어야 할 대상, 범위, 방법 등을 결정한다
- 테스트 지원 도구를 선정하고 사용자 교육 및 훈련을 계획한다
- 테스트 환경 구축과 관련된 사항을 결정한다
- 테스트 수행 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 명세(Test specification)를 작성한다
- 테스트 환경(Test environment)을 구축한다 (많은 경우 시스템 및 네트워크 관리자와 협업)
- 테스트 데이터 준비 및 획득을 담당한다
- 테스트 구현, 실행, 로그 기록 및 테스트 실행 결과 평가, 결함 보고의 임무를 수행한다
- 테스트 운영 및 관리 도구, 테스트 모니터링 도구를 사용한다
- 테스트 자동화 구현(네비게이터) 및 자동화 요구사항을 구체화한다
- 컴포넌트나 시스템의 성능을 측정한다
- 테스트 명세서를 리뷰한다

테스트 분석, 테스트 설계, 특정 테스트 타입 또는 테스트 자동화 업무를 수행하는 담당자는 이러한 역할을 전담하는 전문가일 수 있다. 테스트 레벨에 따라, 그리고 개발 대상 제품 및 프로젝트와 관련된 리스크에 따라 다양한 분야의 담당자가 일정 수준의 독립성을 유지하면서 테스터의 역할을 담당할 수 있다. 전형적으로 컴포넌트 테스트와 통합 테스트 레벨에서의 테스터는 개발자이고, 인수 테스트 레벨에서의 테스터는 비즈니스 전문가 또는 사용자일 것이다. 그리고 운영상의 인수 테스팅(Operational acceptance testing)을 위한 테스터는 운영자일 것이다.

5.2 테스트 계획과 추정 (K3) | 40분

용어
테스트 접근법(test approach), 테스트 전략(test strategy)

5.2.1 테스트 계획 (K2)

여기서는 개발과 구현 프로젝트 내에서 테스트 계획 활동이 갖는 목적과 함께 유지보수 활동에 대한 테스트 계획 활동의 목적에 대해 다룬다. 테스트 계획 활동은 마스터(또는 프로젝트) 테스트 계획, 그리고 시스템 테스팅과 인수 테스팅 등 각 테스트 레벨별 테스트 계획에 문서화할 수 있다. 테스트 계획 활동 문서의 윤곽은 소프트웨어 테스트 문서화 표준(Standard for Software Test Documentation IEEE Std 829- 1998) 문서에 수록되어 있다.

테스트 계획 활동은 조직의 테스트 정책, 테스팅 범위, 목표, 리스크, 제약 사항, 심각성, 테스트 용이성과 자원의 가용성에 영향을 받는다. 프로젝트 및 테스트 계획이 진행되면 될수록 보다 많은 정보가 가용해지고 보다 상세한 내용을 테스트 계획에 포함할 수 있다.

테스트 계획 활동은 모든 테스트 수명주기의 프로세스와 활동에 관여하는 지속적인 활동이다. 변하는 리스크를 감지하기 위해 테스트 활동으로부터 받은 피드백을 활용하고, 이를 통해 테스트 계획 활동을 조정할 수 있다.

5.2.2 테스트 계획 활동 (K3)

전체 시스템 또는 시스템 일부에 대한 테스트 계획 활동은 다음과 같다.

- 테스트 범위(Scope)와 리스크를 결정하고, 테스팅의 목적을 식별한다
- 테스트 레벨과 각 테스트 레벨별 진입과 완료 조건 정의를 포함하는 테스팅의 총체적인 접근법(테스트 전략)을 정의한다
- 테스팅 활동과 소프트웨어의 획득, 공급, 개발, 운영, 유지보수 단계의 수명주기 활동을 통합하고 조정한다
- 무엇을 테스트하고, 누가 해당 테스트 활동을 수행하고, 어떻게 수행하고, 테스트 결과는 어떻게 평가할지를 결정한다
- 테스트 분석과 설계 일정을 계획한다
- 테스트 구현, 실행 및 평가 일정을 계획한다
- 정의된 다양한 테스트 활동에 자원을 할당한다
- 테스트 문서의 분량, 상세함의 정도, 구조와 템플릿을 정의한다
- 테스트 준비와 실행, 결함과 리스크 이슈를 모니터링하고 제어하기 위한 측정 기준을 선정한다
- 충분한 정보를 제공하여 테스트 준비와 실행이 재현 가능하도록 테스트 프로시저의 상세 수준을 결정한다

5.2.3 진입 조건 (K2)

진입 조건은 특정 테스트 레벨의 시작 또는 특정 테스트들의 실행 준비 시점과 같이 테스팅을 언제 시작할지를 정의하는 것이다.

전형적인 진입 조건은 다음과 같은 사항들로 구성된다.

- 테스트 환경의 가용 및 준비 여부
- 테스트 환경에서 테스트 도구 준비 여부
- 테스트할 수 있는 코드의 가용 여부

- 테스트 데이터 가용 여부

5.2.4 완료 조건 (K2)

완료 조건은 특정 테스트 레벨의 종료 또는 특정 테스트들의 목표 달성 여부와 같이 테스팅을 언제 종료할지를 정의하는 것이다.

전형적인 테스트 완료 조건은 아래와 같은 사항들로 구성된다.

- 코드의 커버리지, 기능(성) 커버리지, 리스크 커버리지와 같은 보장성 또는 완전성 측정치
- 추정된 결함 밀도 또는 신뢰성 측정치
- 테스트 비용과 예산
- 잔여 리스크(예: 수정되지 않은 결함 또는 일부 테스트 대상에 대한 불충분한 테스트 커버리지 등)
- 시장 출시 시기에 따른 스케줄

5.2.5 테스트 추정 (K2)

테스트 노력을 추정하는 두 가지 방법은 다음과 같다.

- 메트릭 기반(Metrics-based) 접근법: 과거 프로젝트나 유사 프로젝트의 메트릭을 근거로 또는 매우 비공식적일 경우 전형적인 값을 근거로 테스팅 업무량(Effort) 예측
- 전문가 기반 접근법: 업무의 소유자 또는 전문가 주체에 의한 예측을 근거로 예측하는 방법

일단 테스트 업무량(노력)이 산정되면 테스트에 투입할 리소스를 식별하고 스케줄링을 할 수 있다.

테스트 업무량은 아래와 같이 다양한 요소에 영향을 받을 수 있다.

- 제품의 특성: 테스트를 모델링하는 데 사용하는 다양한 정보와 명세서(테스트 베이스)의 품질, 제품 사이즈, 문제 도메인의 복잡성, 신뢰성 및 보안성 요구수준, 문서화 요구수준
- 개발 프로세스의 특성: 조직의 안정성, 사용한 도구, 테스트 프로세스, 투입될 인력의 스킬 수준, 시간적 압박 정도
- 테스팅 결과물: 결함 수와 요구되는 재작업량

5.2.6 테스트 전략, 테스트 접근법 (K2)

테스트 접근법이란 특정 프로젝트에 대한 테스트 전략의 구현을 말하는 것이다. 테스트 접근법은 테스트 계획에서 우선 정의되고 테스트 설계서에서 다시 한번 정의된다. 일반적으로 (테스트)프로젝트의 목표와 리스크 평가를 기반으로 한 의사결정 사항을 포함한다. 이 테스트 접근법을 기반으로 테스트 프로세스를 계획하고, 테스트 설계 기법과 적용할 테스트 종류를 선택하고, 진입 및 완료 조건을 정의하게 된다.

접근법은 정황(Context)을 기반으로 선정되며 리스크, 위험(Hazards) 및 안전(Safety) 사항, 사용 가능한 리소스 및 스킬, 기술적 배경, 시스템의 특성(예: 사용자 지정 구축 vs COTS), 테스트 목적 및 규정을 고려한다.

일반적인 접근법은 다음과 같다:

- 분석적(Analytical) - 리스크 기반 테스팅(Risk-based testing)과 같이 제품의 리스크 분석을 통해 집중적으로 테스팅해야 할 곳을 결정하는 것
- 모델기반(Model-based) - 확률론적 테스팅(Stochastic testing, random testing)을 통하여 장애율 통계를 바탕으로 설계하는 테스팅

- 방법론적(Methodical) - 오류 추측(Error guessing) 또는 장애 공격(Fault attack)과 같은 장애기반, 경험 기반, 체크리스트 기반, 소프트웨어 품질 특성 기반 테스팅
- 프로세스 및 표준 준수(Process or standard - compliant) - 산업 특화 표준 또는 다양한 애자일 개발 방법론에서 제시한 방법에 의한 테스팅
- 동적/발견적(Dynamic and heuristic) - 탐색적 테스팅(Exploratory testing)처럼 미리 계획되고 설계된 테스팅이라기보다는 사후적 접근법의 하나로, 실행과 평가가 동시에 이루어지는 테스팅
- 자문기반(Consultative) - 외부 전문가의 조언과 가이드를 바탕으로 테스트 커버리지 등의 기준을 정하는 테스팅
- 리그레션 기피형(Regression-averse) - 보유한 테스트 관련 자료, 리그레션 테스트 자동화 스크립트, 표준 테스트 스위트 등의 재사용을 통한 테스팅

리스크 기반 동적 접근법(Risk-based dynamic approach)과 같이 서로 다른 접근법들을 조합적으로 사용할 수 있다.

5.3 테스트 경과 모니터링과 제어 (K2) | 20분

용어

결함 밀도(defect density), 실패율(failure rate), 테스트 제어(test control), 테스트 모니터링(test monitoring), 테스트 요약 보고서(test summary report)

5.3.1 테스트 경과 모니터링 (K1)

테스트 모니터링의 목적은 테스트 활동에 대한 피드백과 가시성(Visibility)을 제공하는 것이다. 모니터링하는 정보는 수동 또는 자동적으로 수집할 수 있으며, 수집된 정보는 커버리지 같은 테스트 완료 조건(Exit criteria)을 측정하기 위해 사용할 수 있다. 메트릭(Metrics, 측정 요소)은 제품의 품질 평가는 물론 계획된 일정과 예산 대비 진척도를 평가하기 위해 사용할 수 있다. 테스트 진척도를 측정하는 일반적인 테스트 메트릭(Test metrics)은 다음과 같은 내용을 포함한다.

- 테스트 케이스를 작성(하는 작업에서 완료된 업무의 비율(혹은 계획된 테스트 케이스가 작성된 비율)
- 테스트 환경을 갖추는 작업에서 완료된 업무의 비율
- 테스트 케이스 실행 관련 메트릭(수행되거나 수행되지 않은 테스트 케이스의 수, 통과되거나 실패(Pass/Fail)한 테스트 케이스의 수)
- 결함 정보(예: 결함 밀도(Defect density), 발견된 결함과 수정된 결함, 장애율, 재테스트 결과)
- 요구사항, 리스크, 코드의 테스트 커버리지
- 제품에 대한 테스터의 주관적 자신감(Subjective confidence)
- 테스트 마일스톤(Milestone) 날짜
- 테스팅 비용 관련 메트릭(다음 번 결함을 발견하거나 다음 번 테스트를 수행하는 것의 이득과 비용을 비교)

5.3.2 테스트 보고 (K2)

테스트 보고는 테스팅 동안의 노력 및 활동에 대한 정보를 요약하는 것으로 다음과 같은 요소들이 포함된다.

- 테스팅 기간 동안 처리했던 주요 업무(예: 테스트 완료 조건(Exit criteria)이 충족된 시점)
- 향후 업무에 대한 의사결정을 지원할 만큼 의미 있게 분석된 정보와 메트릭(예: 잔존 결함의 평가, 테스팅 수행의 경제적 이득, 드러난 리스크, 테스트된 소프트웨어에 대한 자신감의 정도)

테스트 요약 리포트의 기본적인 형태는 '소프트웨어 테스트 문서화 표준('Standard for Software Test Documentation (IEEE Std 829-1998)'에 정의되어 있다.

메트릭은 특정 테스트 레벨에서 테스팅을 진행하는 도중이나 마감 단계에서 다음의 내용들을 평가하기 위해 수집되어야 한다.

- 해당 테스트 레벨에 대한 테스트 목적의 적합성(Adequacy)
- 적용된 테스트 접근법의 적합성
- 테스트 레벨 별 목적에 대한 테스팅의 효과성

5.3.3 테스트 제어 (K2)

테스트 제어는 테스트 진행 보고서나 수집된 메트릭을 근거로 테스트가 계획대로 수행되도록 가이드하거나 정정하는 활동이다. 이는 테스팅 전 영역을 범위로 하며 소프트웨어 수명주기 활동이나 업무에도 영향을 줄 수

있다.

테스트 제어 활동의 예는 다음과 같다.

- 테스트 모니터링으로부터 얻은 정보를 기반으로 의사결정을 한다
- 식별된 리스크(소프트웨어 개발과 수정의 지연 등) 발생 시 테스트의 우선순위를 변경한다
- 테스트 환경의 가용성(Availability) 또는 비가용성(Unavailability)에 따라 테스트 일정을 변경한다
- 수정 사항을 빌드(Build)에 반영하기 전에, 해당 수정 사항을 개발자가 재테스트하도록 요구하는 등의 테스트 진입 조건을 정한다

5.4 형상 관리 (K2) | 10분

용어

형상 관리(configuration management), 버전 관리(version management)

배경

형상 관리의 목적은 프로젝트나 제품의 전체 수명주기에 걸쳐 시스템이나 소프트웨어(컴포넌트, 데이터, 문서)의 상태를 그대로 보전(Integrity), 보호(Umbrella)하고 유지하기 위함이다.

테스팅 측면에서 형상 관리는 다음의 내용을 보장한다.

- 테스트웨어(Testware) 식별, 버전 관리, 변경 추적, 상호 연관성, 개발 아이템(테스트 대상)과의 연계 관리 등을 통해 테스트 프로세스 전반에 걸친 추적성(Traceability) 확보.
- 테스트 문서에서 참조하고 있는 모든 관련 문서 또는 소프트웨어 아이템이 명확하게 관리됨.

테스터는 형상 관리를 통해 테스트된 품목, 테스트 문서, 테스트(케이스)와 테스트 하네스 등을 혼선 없이 관리하고 효율적으로 재사용할 수 있다.
형상 관리 절차와 인프라(도구 지원)는 테스트 계획 단계에서 결정하고 문서화하여, 구현해야 한다.

5.5 리스크와 테스팅 (K2) | 30분

용어

제품 리스크(product risk), 프로젝트 리스크(project risk), 리스크(risk), 리스크 기반 테스팅(risk based testing)

배경

리스크는 이벤트, 위험 요소(Hazard), 위협(Threat) 혹은 상황의 발생 가능성과, 발생했을 경우의 바람직하지 못한 결과 즉, 잠재적인 문제로 정의될 수 있다. 리스크 수준은 의도하지 않은 이벤트가 발생할 가능성과 그 영향(의도하지 않은 이벤트로 인한 해로운 결과)에 의해 결정된다.

5.5.1 프로젝트 리스크 (K2)

프로젝트 리스크는 프로젝트 목적을 달성하기 위한 프로젝트의 역량(Capability) 전반과 관계된 리스크로, 다음과 같은 리스크가 존재한다.

- 조직적인 요소
  - 테스팅 전문 스킬과 인력의 부족
  - 개인적 이슈
  - 다음과 같은 정치적인 이슈
    - 테스터가 자신의 요구(Needs)와 테스트 결과를 전달하면서 발생하는 문제
    - 테스팅과 리뷰에서 발견할 정보를 개발 프로젝트에 반영(Follow-up)하는 것에 실패(이 경우, 개발과 테스팅 업무가 개선되지 않음)
  - 테스팅에 대한 비현실적인 태도나 기대치(테스팅 중에 검출한 결함의 가치를 인정하지 않음)

- 기술적인 이슈
  - 완성도 높은 요구사항을 정의하면서 발생하는 문제
  - 요구사항이 주어진 기존의 제약 조건을 수용할 수 없는 범위
  - 시간 내에 준비되지 않은 테스트 환경
  - 지연된 데이터 변환, 마이그레이션 계획 및 개발과 테스팅 데이터 변환 / 마이그레이션 도구
  - 설계, 코드, 구성 데이터, 테스트 데이터 및 테스트의 낮은 품질

- 공급자 이슈
  - 공급자인 제3자 협력 업체(Third party)가 역할 수행에 실패
  - 계약상의 이슈

리스크를 분석하고, 관리하고, 완화하고자 할 때 테스트 매니저는 체계적으로 정립된 프로젝트 관 리 원칙(Project management principles)을 따라야 한다. '소프트웨어 테스트 문서 표준(Standard for Software Test Documentation - IEEE 829- 1998)'에서는 테스트 계획에 리스크와 리스크로 인해 우발적으로 발생할 수 있는 상황(Risk and contingencies)에 관해 기술하라고 요구한다.

5.5.2 제품 리스크 (K2)

소프트웨어나 시스템에서 의도하지 않은 향후 이벤트나 위험 요소가 존재하는 잠재적인 장애 영역(Potential failure

areas)을 제품 리스크(Product risks)라고 하는데, 이는 곧 제품 품질에 대한 리스크이다. 잠재적 장애 영역은 다음과 같다.

- 개발팀에서 전달받은 장애 포함 가능성이 높은(Failure-prone) 소프트웨어
- 소프트웨어 및 하드웨어가 개인이나 회사에 손실을 끼칠 가능성
- 취약한 소프트웨어 특성(Characteristics – 예: 기능성, 신뢰성, 사용성, 성능)
- 취약한 데이터 무결성 및 품질 (예: 데이터 마이그레이션 이슈, 데이터 변환 이슈, 데이터 전송 이슈, 데이터 표준 위반)
- 의도된 기능을 수행하지 못하는 소프트웨어

리스크는 테스팅을 시작하는 시점을 결정하고, 더 강하고 심도 있게 테스트할 부분을 결정하는 데 사용되는 반면, 테스트는 의도하지 않은 결과를 야기시키는 리스크를 줄이거나, 의도하지 않은 결과로 일으키는 영향(Impact)과 손실을 줄이기 위해 사용된다.

제품 리스크는 프로젝트의 성공과 연관된 리스크이다. 리스크 제어 활동으로서의 테스팅은 심각한 결함을 얼마나 효과적으로 제거했는지를 측정하여 잔존 리스크(Residual risk)에 대한 피드백을 제공한다. 이때, 해당 결함으로 야기될 수 있는 사고에 대비한 계획 (Contingency plan)의 효과성도 측정하여 잔존 리스크에 대한 피드백을 제공한다.
리스크 기반 테스팅은 프로젝트의 초기 단계부터 리스크에 미리 대처할 수 있게 하여 제품의 리스크 수준을 줄여준다. 리스크 기반 테스팅은 또한 제품 리스크를 식별해 주고, 해당 리스크를 이용해 테스트 계획 및 제어, 테스트 설계 및 명세화, 테스트 준비 및 실행을 가이드한다. 리스크 기반 접근법에서 식별된 리스크는 다음과 같이 사용될 수 있다.

- 사용할 테스트 기법 결정
- 테스트 수행 범위 결정
- 심각한 결함을 조기에 발견하기 위해 테스트의 우선순위 결정
- 테스트와 관련되지는 않았지만 리스크를 줄이기 위해 필요한 활동(경험이 적은 개발 설계자에게 교육 훈련 제공 등)을 수행해야 할지 결정

리스크 관리 활동은 다음과 같이 체계적으로 접근하여 프로젝트 실패의 가능성을 최소화해야 한다.

- 잘못될 수 있는 사항(Risk)을 평가하고 정기적으로 재평가한다.
- 어떤 리스크가 중요시되어야 하는지 결정한다.
- 리스크 수준에 맞게 적절하게 대응한다.

여기에서 테스팅은 새로운 리스크를 식별할 수 있게 지원하고, 어떤 리스크를 감소시킬지 결정하는 데 도움이 되며, 리스크의 불확실성을 낮출 수 있다.

5.6 인시던트 관리 (K3) | 40분

용어

인시던트 로깅(incident logging), 인시던트 관리(incident management), 인시던트 리포트(incident report)

배경

테스팅의 한 가지 목적은 결함을 발견하는 것이기 때문에 실제 결과와 기대 결과 사이의 불일치를 인시던트(Incident)로써 기록하고 관리할 필요가 있다. 인시던트는 조사되어야 하며, 그 결과 결함으로 판정될 수 있다. 인시던트와 결함을 제거하기 위한 적절한 방법이 정의되어야 한다. 테스터는 인시던트를 발견하고 분류하는 것부터 이를 수정하고 인시던트나 결함이 해결되었는지 확인하는 것까지 추적해야 한다. 모든 인시던트를 능숙하게 관리하기 위해 조직은 인시던트 관리 프로세스와 인시던트 분류 규칙을 제정해야 한다.

인시던트는 프로그램 개발, 리뷰, 테스팅 혹은 소프트웨어 제품 사용 중에 발견되거나 발생할 수 있다. 인시던트가 발생하는 곳은 프로그램 코드, 운영 중인 시스템, 요구사항 문서, 개발 문서, 테스트 문서, 도움말이나 설치 가이드(사용자용 정보 또는 문서) 등이 있다.

인시던트 리포트는 다음과 같은 목적을 갖는다.

- 개발자와 프로젝트 관련자가 문제점을 식별하고, 격리하고, 필요하면 정정할 수 있도록 피드백을 제공
- 테스트 리더가 시스템의 품질 또는 테스트 진척을 추적할 수 있도록 정보를 제공
- 테스트 프로세스 개선(Test process improvement)에 대한 아이디어 제공

인시던트 리포트에는 다음과 같은 내용들이 포함될 수 있다.

- 이슈 발생 날짜, 이슈를 제기한 조직, 작성자(Author)
- 기대 결과와 실제 결과
- 테스트 항목(Configuration item)과 환경(Environment)의 식별
- 소프트웨어나 시스템의 수명주기에서 인시던트가 발견된 시점
- 로그, 데이터베이스 덤프, 화면 덤프 등을 포함한 인시던트 재현이 가능한 상세 설명 자료
- 이해관계자의 이익에 영향을 주는 범위 또는 정도
- 결함이 시스템에 미치는 심각도
- 수정 우선순위(긴급하게 처리해야 하는 순서)
- 인시던트의 상태(Status): 오픈, 보류, 중복, 수정 대기, 수정 완료, 종료
- 결론, 의견, 승인 여부
- 관련 이슈(Global issues): 인시던트로 인한 변경이 영향을 줄 수 있는 다른 영역
- 변경 이력(Change history): 인시던트를 격리하고, 수정하고, 수정 여부를 확인하는 등의 활동을 순서대로 기록
- 참조 사항(References): 관련된 테스트 케이스 식별 정보

인시던트 리포트의 구조는 소프트웨어 테스트 문서 표준 (Standard for Software Test Documentation - IEEE 829-1998)를 참조할 수 있다.

제6장 테스트 지원 도구 (K2) | 80분

학습목표

6.1 테스트 도구의 종류 (K2)

LO-6.1.1 테스트 도구를 사용 목적과 테스트 프로세스 단계 및 소프트웨어 수명주기의 활동에 따라 유형별로 구별할 수 있다 (K2)

LO-6.1.3 테스트 도구라는 용어와 테스트를 지원하기 위해 도구를 사용하는 목적을 설명할 수 있다 (K2)

6.2 도구의 효과적인 사용: 잠재 가치와 위험 (K2)

LO-6.2.1 테스트 자동화 적용 및 테스트 지원 도구의 사용과 관련된 기대 효과와 잠재 위험을 요약할 수 있다 (K2)

LO-6.2.2 테스트 실행 도구, 정적 분석, 테스트 관리 도구를 활용할 경우 특별히 고려해야 하는 사항을 기억할 수 있다 (K1)

6.3 조직에 도구 도입 및 배포 (K1)

LO-6.3.1 조직에 도구를 도입하는 주요한 원칙을 이해하고 설명할 수 있다 (K1)

LO-6.3.2 도구 평가를 위한 사전 검증 단계와 도구 도입을 위한 파일럿 프로젝트 단계를 가져야 하는 이유와 목적에 관해 설명할 수 있다 (K1)

LO-6.3.3 성공적인 도구 지원을 위해서는 도구를 획득하는 것 이상으로 여러 가지 중요한 요소를 고려해야 함을 인식한다 (K1)

6.1 테스트 도구의 종류 (K2) | 45분

용어

형상관리 도구(configuration management tool), 커버리지 도구(coverage tool), 디버깅 도구(debugging tool), 동적 분석 도구(dynamic analysis tool), 인시던트 관리 도구(incident management tool), 부하 테스팅 도구(load testing tool), 모델링 도구(modeling tool), 모니터링 도구(monitoring tool), 성능 테스팅 도구(performance testing tool), 탐사 효과(probe effect), 요구사항 관리 도구(requirements management tool), 리뷰 도구(review tool), 보안성 도구(security tool), 정적 분석 도구(static analysis tool), 스트레스 테스팅 도구(stress testing tool), 테스트 비교 도구(test comparator), 테스트 데이터 준비 도구(test data preparation tool), 테스트 설계 도구(test design tool), 테스트 하네스(test harness), 테스트 실행 도구(test execution tool), 테스트 관리 도구(test management tool), 단위 테스트 프레임워크 도구(unit test framework tool)

6.1.1 테스팅 지원 도구 (K2)

테스트 도구는 테스팅을 지원하는 하나 또는 그 이상의 활동에 활용될 수 있다. 예를 들어 다음과 같은 활동이 있다.

1. 테스트에 직접 사용되는 도구(예: 테스트 실행 도구, 테스트 데이터 생성 도구, 결과 비교 도구)
2. 테스팅 프로세스를 관리하는 데 사용되는 도구 (예: 테스트 관리, 테스트 결과, 데이터, 요구사항, 인시던트, 결함 등을 관리하기 위해 사용되는 도구, 또한 테스트 실행을 리포트 및 모니터링하는 데 사용되는 도구)
3. 정찰(Reconnaissance) 혹은 탐사(Exploration)에 사용되는 도구(예: 응용 프로그램의 파일 작업을 모니터링 하는 도구)
4. 테스팅을 지원하는 모든 도구. 이런 의미에서 테스트에 활용될 경우 엑셀 스프레드시트도 테스트 도구로 분류될 수 있음

테스트를 위해 도구를 사용하는 것은 다음과 같은 목적들이 있으며, 정황에 따라 이들 중 하나 또는 그 이상을 목표로 할 수 있다.

- 반복적인 작업을 자동화하거나 테스트 계획, 테스트 설계, 테스트 리포트 및 모니터링과 같은 수동 테스트 활동을 지원함으로써 테스트 활동의 효율성을 높인다
- 수동으로 진행할 경우 많은 자원을 필요로 하는 작업의 자동화(예: 정적 테스팅)
- 수동으로 진행할 수 없는 작업의 자동화(예: 클라이언트-서버 응용 프로그램의 대규모 성능 테스트)
- 테스팅의 신뢰성 증가(예: 대규모 데이터 비교 자동화 또는 동작 시뮬레이션)

"테스트 프레임웍"라는 용어도 업계에서 통용되고 있으며, 적어도 세 가지 의미가 있다.

- 재사용 가능하고 확장 가능하면서 테스팅 도구를 구축하는 데 활용될 수 있는 테스트 라이브러리 (테스트 하네스라고 지칭하기도 함)
- 테스트 자동화 설계의 유형(예: 데이터 기반(Data-driven), 키워드 중심(Keyword-driven))
- 테스팅 실행의 전체적인 프로세스

이 실라버스에서는 6.1.6에서 설명하고 있는 바와 같이 "테스트 프레임워크"는 처음 두 가지 의미로 사용된다.

6.1.2 테스트 도구의 분류 (K2)

테스팅의 여러 가지 측면을 지원하기 위한 많은 도구가 존재한다. 도구는 상용인지, 무료인지, 오픈 소스인지, 셰어웨어인지, 사용된 기술이 무엇인지 등과 같이 여러 기준에 따라 분류될 수 있다. 이 실라버스에서는 테스트 도구가 지원하는 테스트 활동에 따라 도구를 분류하였다.

어떤 도구는 명확하게 하나의 테스트 활동만을 지원하고, 어떤 도구는 하나 이상의 활동을 지원하기도 한다. 여기에서의 분류는 가장 밀접하게 관련된 하나의 활동을 기준으로 하였다. 하나의 공급자가 여러 개의 도구를 제공하는 경우, 특히 이 도구들이 함께 동작하도록 설계되어있는 경우에는 하나의 패키지로 묶어서 제공할 수 있다.

어떤 종류의 도구는 도구 자체가 테스트의 실제 결과에 영향을 미친다는 측면에서 침입적(Intrusive)이라고 할 수 있다. 예를 들어, 도구가 실행하는 추가적인 명령 때문에 실제 타이밍이 달라질 수 있고, 어떤 도구를 사용하느냐에 따라 코드 커버리지의 측정치가 달라질 수 있다. 이러한 침입적 도구를 사용하여 달라진 결과를 탐사 효과(Probe effect)라고 부른다.

어떤 도구들은 컴포넌트 테스트나 컴포넌트 통합 테스트를 하는 동안 개발자에게 더욱 유용한 도구들이 있다. 이런 도구들은 아래의 분류에 "(D-개발자 지원)"으로 표기되어 있다.

6.1.3 테스트 관리 지원 도구 (K1)

테스트 관리 지원 도구(Management tools)는 소프트웨어 수명주기(Software life cycle) 전체에 걸쳐 모든 테스팅 활동에 사용된다.

테스트 관리 도구

테스트 관리 도구는 정량 분석 및 테스트 대상에 관한 보고에 대한 지원을 제공한다. 또한, 결함 추적 및 요구사항 관리, 테스트 실행을 위한 인터페이스를 제공한다. 또한, 요구사항 명세와 테스트 대상간의 추적성을 제공하며, 독립적인 버전 관리 기능을 가지고 있거나 외부 버전 관리 도구를 위한 인터페이스를 제공할 수 있다.

요구사항 관리 도구

요구사항 관리 도구는 요구사항 명세를 저장하고, 요구사항 속성을 저장(우선순위 포함)하고, 고유 식별자를 제공하고, 개별 테스트와 요구사항 간의 추적을 지원한다. 또한, 이런 도구는 일관되지 않거나 누락된 요구사항을 식별하는 데 도움이 될 수 있다.

인시던트 관리 도구(결함 관리 도구)

인시던트 관리 도구는 인시던트 리포트, 예를 들어, 결함, 장애, 변경요청, 인지된 문제와 이상현상을 저장하고 관리한다. 그리고 인시던트의 수명주기 관리를 보조하는 데, 통계 분석을 위한 지원을 받을 수도 있다.

형상 관리 도구

형상 관리 도구는 엄밀히 말하자면 테스팅 도구가 아니긴 하지만, 테스트웨어와 관련된 소프트웨어의 저장 및 버전 관리에 필요하다. 특히 운영 체제 버전, 컴파일러, 브라우저 등이 다른 하나 이상의 하드웨어/소프트웨어 환경을 구성할 경우 더욱 필요하다.

6.1.4 정적 테스팅 지원 도구 (K1)

정적 테스팅 도구는 개발 과정 초기 단계에서 비용대비 효과적인 방법으로 더 많은 결함을 찾는 방법을 제공한다.

리뷰 도구

리뷰 도구는 리뷰 프로세스, 체크리스트, 리뷰 가이드라인을 지원하며, 리뷰 의견과 결함 및 노력의 보고서를 저장하고 전달하는 데 사용된다. 또한 규모가 크거나 지역적으로 분산된 팀이 사용하는 온라인 리뷰를 지원하기도 한다.

정적 분석 도구 (D-개발자 지원)

정적 분석 도구는 코딩 표준 준수(보안 코딩 포함), 구조 분석, 의존성 등을 지원함으로써 개발자 및 테스터가 동적 테스팅 이전에 결함을 발견할 수 있게 한다. 또한 코드(예: 복잡성)에 대한 통계를 제공하여 계획 또는 리스크 분석에 도움이 될 수 있다.

모델링 도구 (D-개발자 지원)

모델링 도구는 불일치(모순) 및 결함을 찾아 소프트웨어 모델의 유효성을 검사하는 데 사용된다. (예: 관계형 데이터베이스에 대한 물리적 데이터모델(PDM)). 이 도구는 많은 경우 모델을 기반으로 테스트 케이스를 도출하는 것을 지원하기도 한다.

6.1.5 테스트 명세 지원 도구 (K1)

테스트 설계 도구

테스트 설계 도구는 테스트 입력값이나 실행 가능한 테스트 또는 요구사항, 그래픽 사용자 인터페이스, 설계 모델(상태, 데이터, 오브젝트), 코드 등으로부터의 테스트 오라클을 생성하기 위해 사용된다.

테스트 데이터 준비 도구

테스트 데이터 준비 도구는 데이터베이스, 파일, 데이터 전송을 적절히 조작하여 데이터의 익명성을 통해 보안을 보장하고 테스트 실행 시 사용될 테스트 데이터를 마련하는 도구이다.

6.1.6 테스트 실행 및 로깅 지원 도구 (K1)

테스트 실행 도구

테스트 실행 도구는 저장된 입력값과 예상 결과를 이용하여 테스트를 자동 혹은 반자동으로 실행해 준다. 스크립팅 언어를 사용하며 대부분의 경우 각각의 테스트 실행에 대한 테스트 로그를 제공한다. 또한, 테스트를 기록하기 위해 사용될 수 있으며, 일반적으로 스크립트 언어 또는 테스터의 데이터를 매개 변수화하고 기타 다른 요청사항(Customization)을 반영할 수 있도록 GUI기반 구성을 지원한다.

테스트 하네스/유닛 테스트 프레임워크 도구 (D-개발자 지원)

유닛 테스트 하네스 및 프레임워크는 스텁 또는 드라이버와 같은 모의 오브젝트를 통해 테스트 대상이 실행될 환경을 시뮬레이션함으로써 컴포넌트나 시스템 일부에 대한 테스팅을 가능하게 한다.

테스트 비교자

테스트 비교자는 파일이나 데이터베이스 혹은 테스트 결과의 차이점 알려준다. 테스트 실행 도구는 일반적으로 내부에 동적인 비교자를 포함하고 있지만, 테스트 실행 이후의 비교는 별도의 비교 도구를 사용해서 할 수 있다.

특히 자동화된 경우, 테스트 비교자는 테스트 오라클을 이용한다.

커버리지 측정 도구 (D-개발자 지원)
커버리지 측정 도구는 침입적(Intrusive) 또는 비침입적(Non-intrusive) 방법을 통해 일련의 테스트가 실행한 특정 유형의 코드 구조(예: 구문, 분기 또는 결정, 모듈 또는 함수 호출)가 몇 퍼센트인지 측정한다.

보안 테스팅 도구
보안 테스팅 도구는 소프트웨어의 보안 특성을 평가하는 데 사용된다. 즉, 소프트웨어가 데이터 기밀성, 무결성, 인증, 권한 부여, 가용성, 부인 방지(Non-repudiation) 등을 보호하는 능력을 평가한다. 보안 도구는 대부분의 경우 특정 기술, 플랫폼, 또는 목적에 초점을 맞추고 있다.

6.1.7 성능과 모니터링 도구 (K1)

동적 분석 도구 (D-개발자 지원)
동적 분석 도구는 소프트웨어 실행 도중에만 발생하는 시간 의존성(Time dependencies)과 메모리 누수(Memory leaks)와 같은 결함을 발견해 낸다. 이런 도구는 일반적으로 컴포넌트나 컴포넌트 통합 테스팅에서 사용되거나 미들웨어를 테스트할 때 사용된다.

성능/부하/스트레스 테스팅 도구
성능 테스팅 도구는 동시 사용자 수, 램프업(Ramp-up) 패턴, 빈도 및 트랜잭션의 상대적인 빈도 등이 시뮬레이션된 다양한 사용 조건 하에서 시스템이 어떻게 동작하는가를 모니터링하고 보고 한다. 부하 시뮬레이션은 가상 사용자들을 생성해서 달성한다. 이들은 일반적으로 부하 발생기로 불리는 다양한 테스트 장비에서 특정 종류의 트랜잭션을 수행한다.

모니터링 도구
모니터링 도구는 특정 시스템 리소스의 사용량을 지속적으로 확인하고 분석해서 보고하거나 예상되는 서비스상의 문제점에 대해 경고 메시지를 제공한다.

6.1.8 특정 목적 테스팅 지원 도구 (K1)

데이터 품질 평가
데이터 변환/마이그레이션 프로젝트, 또 웨어하우스 등과 같은 응용 프로그램 프로젝트에서는 데이터가 가장 중요한 요소이며, 중요도와 데이터의 양은 프로젝트에 따라 다양할 수 있다. 이러한 측면에서 봤을 때, 데이터 전환 및 마이그레이션 법칙을 검토하고 확인해서 가공한 데이터가 정확하고 완벽한지, 또 기존에 정의된 정황 기반 표준(Context-specific standard)에 부합하는지를 검증하는 도구가 데이터 품질 평가를 위해 도입되어야 한다.

사용성 테스트를 위한 테스팅 도구도 있다.

6.2 도구의 효과적인 사용: 잠재 가치와 위험 (K2) | 20분

용어
데이터 주도 테스팅(data driven testing), 키워드 주도 테스팅(keyword driven testing), 스크립트 언어(scripting language)

6.2.1 도구 지원 테스팅의 잠재 이익과 위험 (모든 도구에 해당) (K2)

단순히 도구를 구입하거나 임대한다고 하여 그 도구를 성공적으로 사용할 것이라는 보장은 없다. 도 구의 실제적이고 지속적인 가치를 달성하기 위해 추가적인 노력이 반드시 필요하다. 테스팅에서 도구를 사용하여 잠재적인 가치를 얻을 수 있는 기회가 존재하는 반면 더불어 그 위험도 존재한다.

도구 사용의 잠재적인 가치는 다음과 같다.
- 반복적인 업무 감소(예: 리그레션 테스트 수행, 동일 테스트 데이터 재입력, 코딩 표준 준수여부 점검 등)
- 월등한 일관성과 반복성 제공(예: 동일한 주파수를 가진 동일한 순서로 도구에서 실행하는 테스트 및 테스트 요구사항에서 파생된 테스트 등)
- 객관적인 평가 기준 제공(예: 정적 측정치(Static measures), 커버리지 등)
- 테스트 또는 테스팅 진행 정보에 대한 쉬운 접근성 제공(예: 테스트 진척도, 결함률 그리고 성능에 대한 통계와 그래프 등)

반면 도구 사용의 잠재적인 위험은 다음과 같다.
- 도구에 대한 비현실적인 기대(원하는 기능이 모두 있고 사용하기 쉬워 도구만 구입하면 자동화가 이루어질 것이라는 기대)
- 도구의 도입 초기에 필요한 시간, 비용, 노력(예: 교육 훈련이나 외부 전문가 활용 등)을 과소 평가하여 산정
- 도구를 통해 중대하고 지속 가능한 성과를 얻기 위해 필요한 시간이나 노력(예: 테스팅 프로세스의 변경이나 도구가 사용된 방식에 대한 지속적인 개선 등)을 적게 산정하는 것
- 도구에 의해 생성된 테스트 자산(Assets)을 유지보수하는 데 필요한 노력을 적게 산정하는 것
- 도구에 대한 지나친 의존(테스트 디자인을 도구로 대체 또는 수동 테스트가 더 효율적일 수 있는 부분에도 도구 사용)
- 도구 내에 존재하는 테스트 자산의 버전 관리를 하지 않음
- 요구사항 관리 도구, 버전 관리 도구, 인시던트 관리 도구, 결함 추적 도구, 또는 다수의 공급 업체에서 제공하는 도구와 같이 중요한 도구 간의 관계 및 상호운용성 문제를 관리하지 않음
- 도구 공급 업체의 폐업, 해당 도구의 판매 중지, 또는 해당 도구가 다른 공급 업체에게 매각되는 등의 리스크
- 지원, 업그레이드, 결함 수정에 대한 공급 업체의 적절하지 못한 대응
- 오픈 소스/무료 도구 프로젝트의 중단 위험
- 새로운 플랫폼을 지원하지 못하는 것과 같은 예상하지 못한 리스크

6.2.2 도구 유형별 고려사항 (K1)

테스트 실행 도구
테스트 실행 도구는 자동화된 테스트 스크립트를 활용해서 테스트 대상을 실행한다. 대부분의 경우 이런 종류의 도구는 가치를 끌어내기 위해 상당한 노력이 필요하다.

테스터의 수동적인 조작을 기록하여 테스트를 자동화하는 방식은 매력적으로 보이지만, 이러한 접근 방식은 대규모의 테스트를 자동화하는 데는 적절치 않다. 저장된 스크립트는 각 스크립트 마다 특정 데이터와 행위를 선형적으로 표현하고 있다. 따라서 이런 종류의 스크립트는 미처 예상하지 못한 이벤트에 취약할 수 있다.

데이터 주도 방식(Data-driven approach)은 일반적으로 테스트 입력값(데이터)을 스프레드시트에 저장하고, 해당 테스트 데이터를 읽어 들여서 동일한 테스트 스크립트를 매번 다른 데이터로 반복적으로 실행시키는 더 포괄적인 스크립트를 활용한다. 따라서 스크립트 언어에 익숙하지 않은 테스터라도 기존 스크립트의 테스트 데이터를 변경할 수 있다.

데이터 주도 기법이라고 반드시 스프레드시트에 저장된 데이터 조합을 사용해야 하는 것은 아니다. 사전에 정의된 데이터가 아닌 실행할 때 설정 가능한 변수들을 특정 알고리즘을 통해 데이터로 전환하고 이를 응용 프로그램에 공급할 수도 있다. 예를 들어, 어떤 도구는 임의의 사용자 ID를 생성하는 알고리즘을 활용하고, 패턴의 반복성을 지키기 위해 임의 정도(Randomness)를 제어하는 특정 시드(Seed)를 사용할 수도 있다.

키워드 주도 접근 방식에는 스프레드시트에 수행할 동작을 나타내 는 키워드(액션 워드)와 테스트 데이터가 포함되어 있다. 이 방식에서는 스크립트 언어에 익숙하지 않은 테스터라도 키워드를 이용하여 테스트를 정의할 수 있는데, 키워드는 테스트 대상 응용 프로그램에 맞게 개조할 수 있다.

어떤 방식의 테스트 자동화를 선택하더라도 테스트 실행 자동화에 참여하는 테스터 혹은 전문가는 스크립트 언어에 대한 기술적인 전문성이 필요하다.

어떤 스크립트 기법을 사용하더라도 결과 비교를 위해서는 각 테스트의 기대 결과는 저장되어야 한다.

정적 분석 도구

정적 분석 도구는 소스 코드에 코딩 표준을 강제하는 데 사용할 수 있지만, 실제 적용 시 매우 많은 경고 메시지를 발생시킬 수도 있다. 경고 메시지는 소스 코드를 실행 가능한 프로그램으로 변환하는 과정에는 영향을 끼치진 않지만, 향후 소스 코드의 유지보수를 용이하게 하기 위해서는 적절한 조치를 취하는 게 좋다. 정적 분석 도구 사용시 초기에는 필터를 적용하여 불필요한 메시지를 제거하고, 이를 점차 확대 적용하는 것이 효과적인 접근 방식이다.

테스트 관리 도구

테스트 관리 도구는 조직의 요구에 맞는 최적화된 형태의 정보를 생성하기 위해 다른 도구나 스프레드시트 등과 연계하여 사용할 필요가 있다.

6.3 조직에 도구 도입 및 배포 (K1) | 15분

용어
용어 없음

배경
조직에 맞는 도구의 선택을 위한 주요 고려사항은 다음과 같다.
- 조직의 성숙도, 강점과 약점을 평가한 후, 도구 지원으로 테스팅 프로세스를 개선할 기회를 발견한다
- 명확한 요구사항과 객관적인 도구 평가 기준을 가지고 평가한다
- 도구가 테스트 대상 소프트웨어 및 현재의 기반 환경에 효율적으로 적용될 수 있는지, 또는 도구를 효율적으로 사용하기 위해서 기반 환경에 어떤 변화가 필요한지를 식별하기 위한 사전검증(Proof-of-concept) 단계를 가진다
- 교육 제공, 지원 능력, 상업적 능력 등의 기준으로 도구 제작 회사나 판매 회사의 능력을 평가한다. 비상업적 도구인 경우 서비스 지원 공급자의 능력을 평가한다
- 도구 사용과 관련된 조직 내부 교육과 (개별) 지도에 대한 요구사항을 식별한다
- 테스트 팀의 현재 자동화 스킬을 고려해서 도구에 대한 교육 수요를 평가한다
- 확실한 비즈니스 시나리오에 근거해서 비용 대비 효과를 예측한다

조직에서 선택한 도구는 파일럿 프로젝트를 적용해서 도입해야 한다. 파일럿 프로젝트는 다음의 목적을 가져야 한다.
- 도구와 관련된 상세한 사항을 습득한다
- 현 프로세스나 업무에 도구를 어떻게 적용할 수 있는지 평가하고, 도구 적용을 위해 현 프로세스나 업무의 무엇을 변경해야 하는지 결정한다
- 도구의 사용, 관리, 저장, 유지보수에 대한 표준 방식을 결정한다(예: 파일이나 테스트 케이스의 명명 규칙을 결정하거나, 라이브러리를 만들고, 테스트 스위트의 모듈방식을 정의한다)
- 합리적인 비용으로 목표하는 성과에 도달할 수 있는지 평가한다

도입한 도구를 조직 내부에 배포(전파)할 때의 성공 요인은 다음과 같다.
- 조직의 다른 부서에 대한 도구 사용 전파는 점진적으로 한다
- 도구의 사용법에 맞게 프로세스를 수정하고 개선한다
- 새로운 사용자를 위해 교육과 훈련(멘토링)을 한다
- 사용 가이드라인을 정의한다
- 실제 도구 사용을 통해 얻은 교훈을 수집할 방법을 마련한다
- 도구 사용 현황과 성과를 모니터링한다
- 도구 사용을 테스트하는 팀에 대한 지원을 제공한다
- 모든 팀으로부터 사용 후 교훈을 수집한다