10. 애플리케이션 테스트 관리

애플리케이션 테스트

1. 애플리케이션 테스트
   - 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차
   - 개발된 소프트웨어가 요구사항을 만족시키는지 Validation(확인) 하고 소프트웨어가 기능을 정확히 수행하는지 Verification(검증) 한다.

2. 애플리케이션 테스트 기본 원리
   a. 완벽한 테스트 불가능

      : 소프트웨어의 잠재적 결함은 줄일 수 있지만 결함 자체가 없다고 증명할 수는 없음

   b. 파레토 법칙

      : 애플리케이션의 20%에 해당하는 코드에서 전체 결함의 80%가 발견된다는 원칙

   c. 살충제 패러독스

      : 동일한 테스트 케이스로 동일한 테스트를 반복하면 더 이상 결함이 발견되지 않는 현상

   d. 테스팅은 정황(Context) 의존

      : 정황 (Context)에 따라 테스트를 다르게 수행해야 함

   e. 오류-부재의 궤변

      : 소프트웨어 결함을 모두 제거해도 요구사항을 만족시키지 못하면 소프트웨어의 품질은 낮은 것임

   f. 테스트와 위험은 반비례

      : 테스트를 많이 할수록 미래에 발생할 위험을 줄일 수 있음

   g. 테스트의 점진적 확대

      : 테스트는 작은 부분에서 시작하여 점점 확대하며 진행해야 함

   h. 테스트의 별도 팀 수행

      : 테스트는 개발자와 관계없는 별도의 팀에서 수행해야 함

애플리케이션 테스트 분류

1. 프로그램 실행 여부에 따른 테스트
   a. 정적 테스트

      : 프로그램 실행하지 않고 명세서나 소스코드를 대상으로 분석

      : 워크스루, 인스펙션, 코드검사

   b. 동적 테스트

      : 프로그램을 실행하여 오류를 찾는 테스트

: 블랙박스 테스트, 화이트박스 테스트

2. 테스트 기반에 따른 테스트
    a. 명세 기반 테스트
        : 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하는지 확인하는 테스트
        : 동등 분할, 경계 값 분석
    b. 구조 기반 테스트
        : 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트
        : 구문 기반, 결정 기반, 조건 기반
    c. 경험 기반 테스트
        : 유사 소프트웨어나 기술 등에 대한 테스터의 경험을 기반으로 수행하는 테스트
        : 에러 추정, 체크리스트, 탐색적 테스팅

3. 시각에 따른 테스트
    a. 검증 테스트
        : 개발자의 시각에서 제품의 생산 과정 테스트
        : 명세서대로 완성됐는지 테스트
    b. 확인 테스트
        : 사용자의 시각에서 생산된 제품의 결과를 테스트
        : 사용자가 요구한 대로 제품이 완성됐는지, 정상 동작 하는지 테스트

4. 목적에 따른 테스트
    a. 회복 테스트 - 여러 결함을 주어 실패하도록 한 후 복구 가능한지
    b. 안전 테스트 - 시스템 보호 도구가 불법적 침입으로 부터 보호 가능한지
    c. 강도 테스트 - 과도한 정보량에 따른 과부하 시 정상 실행 가능한지
    d. 성능 테스트 - 실시간 성능이나 효율성을 진단하는 테스트
    e. 구조 테스트 - 내부의 논리적 경로, 소스 코드의 복잡도 평가하는 테스트
    f. 회귀 테스트 - 수정된 코드에 대해 새로운 결함이 없음을 확인
    g. 병행 테스트 - 기존과 변경 후 동일 데이터를 입력하여 결과 비교하는 테스트

테스트 기법에 따른 애플리케이션 테스트

1. 화이트박스 테스트 (White Box Test)
    - 모듈의 원시 코드의 논리적인 모든 경로를 테스트하여 테스트케이스 설계하는 방법
    - 모듈 안의 작동을 직접 관찰
    - 모든 문장을 한 번 이상 실행하여 수행

    - 종류
        1. 기초 경로 검사 (Base Path Testing)
            : 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 하는 테스트 기법

2. 제어 구조 검사 (Control Structure Testing)

   : 조건 검사 - 모듈 내의 논리적 조건을 테스트하는 설계 기법

   : 루프 검사 - 프로그램의 반복 구조에 초점을 맞춰 테스트하는 기법

   : 데이터 흐름 검사 - 변수에 초점을 맞춰 테스트하는 기법

- 검증 기준

1. 문장 검증 기준 (Statement Coverage)

   : 소스 코드의 모든 구문이 한 번 이상 수행 되도록

2. 분기 검증 기준 (Branch Coverage)

   : 모든 조건문이 한 번 이상 수행 되도록

3. 조건 검증 기준 (Condition Coverage)

   : 모든 조건문에 대해 True일 때와 False인 경우가 한 번 이상 수행 되도록

4. 분기/조건 기준 (Branch/Condition Coverage)

   : 모든 조건문과 각 조건문에 포함된 개별 조건식의 결과가 True인 경우와 False인 경우가 한 번 이상 수행 되도록

2. 블랙박스 테스트 (Black Box Test)

- 각 기능이 완전히 작동되는 것을 입증하는 테스트 (= 기능 테스트)
- 사용자의 요구사항 명세를 보면서 테스트
- 구현된 기능을 테스트
- 인터페이스를 통해 실시

- 종류

1. 동치 분할 검사 (Equivalence Partitioning Testing)

   : 입력 조건에 타당한 조건과 타당하지 않은 조건의 개수를 균등하게 하여 테스트 케이스를 정하고, 입력 자료에 맞는 결과가 나오는지 확인

   : = 동등 분할 기법

2. 경계값 분석 (Boundary Value Analysis)

   : 입력 조건의 중간값 보다 경계값에서 오류가 발생될 확률이 높다는 것을 이용하여 입력 조건의 경계값을 테스트

3. 원인-효과 그래프 검사 (Cause-Effect Graphing Testing)

   : 입력 데이터 간의 관계와 출력에 미치는 상황을 체계적으로 분석한 다음 효용성 높은 테스트 케이스를 선정하여 검사

4. 오류 예측 검사 (Error Guessing)

   : 과거의 경험이나 확인자의 감각으로 테스트

5. 비교 검사 (Comparison Testing)

   : 여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일 결과가 나오는지 테스트

개발 단계에 따른 애플리케이션 테스트

1. 단위 테스트 (Unit Test)

- 코딩 직후 모듈이나 컴포넌트에 초점을 맞춰 테스트
- 인터페이스, 외부 I/O, 자료 구조, 경계 조건 등을 검사
- 사용자 요구사항을 기반으로 한 기능성 테스트를 최우선으로 수행

- 구조 기반 테스트, 명세 기반 테스트로 나뉨

2. 통합 테스트 (Integration Test)
   - 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트
   - 모듈 간의 상호 작용 오류 검사

3. 시스템 테스트 (System Test)
   - 개발된 소프트웨어가 완벽하게 수행되는가를 점검하는 테스트
   - 기능적 요구사항과 비기능적 요구사항으로 구분하여 각각을 만족하는지 테스트

4. 인수 테스트 (Acceptance Test)
   - 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두고 테스트
   - 사용자가 직접 테스트

   - 종류
     1. 사용자 인수 테스트 - 사용자가 시스템 사용의 적절성 여부 확인
     2. 운영상의 인수 테스트 - 시스템 관리자가 시스템 인수시 테스트
     3. 계약 인수 테스트 - 계약상의 인수/검수 조건을 준수하는지 테스트
     4. 규정 인수 테스트 - 정부 지침, 법규, 규정에 맞게 개발됬는지 테스트
     5. 알파 테스트 - 개발자의 장소에서 사용자가 개발자 앞에서 테스트
     6. 베타 테스트 - 선정된 사용자가 여러명의 사용자 앞에서 테스트

통합 테스트

1. 통합 테스트
   - 단위 테스트가 끝난 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트
   - 종류
     1. 비점진적 통합 방식

        : 단계적으로 통합하는 절차 없이 미리 결합되어있는 프로그램 전체를 테스트

        : 빅뱅 통합 테스트 방식
     2. 점진적 통합 방식

        : 모듈 단위로 단계적으로 통합하면서 테스트

        : 하향식, 상향식, 혼합식 통합 테스트

2. 하향식 통합 테스트 (Top Down Integration Test)
   - 상위 모듈에서 하위 모듈로 통합하면서 테스트
   - 깊이 우선 통합법, 넓이 우선 통합법 사용
   - 주요 제어 모듈은 작성된 프로그램 사용, 종속 모듈은 스텁(Stub)으로 대체

3. 상향식 통합 테스트 (Bottom Up Integration Test)

- 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트
- 하위 모듈들을 클러스터(Cluster)로 결합
- 상위 모듈에서 데이터 입출력을 확인하기 위해 더미 모듈인 드라이버(Driver) 작성

4. 혼합식 통합 테스트
   - 하위 수준에서는 상향식, 상위 수준에서는 하향식 통합을 사용하여 최적의 테스트 지원
   - = 샌드위치식 통합 테스트

5. 회귀 테스팅 (Regression Testing)
   - 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인하는 테스트
   - 이미 테스트된 프로그램의 테스팅을 반복

애플리케이션 테스트 프로세스

1. 애플리케이션 테스트 프로세스
   a. 테스트 계획
   b. 테스트 분석 및 디자인
   c. 테스트 케이스 및 시나리오 작성
   d. 테스트 수행
   e. 테스트 결과 평가 및 리포팅
   f. 결함 추적 및 관리

2. 결함 관리 프로세스
   a. 에러 발견
   b. 에러 등록
   c. 에러 분석
   d. 결함 확정
   e. 결함 할당
   f. 결함 조치
   g. 결함 조치 검토 및 승인

테스트 케이스 / 시나리오 / 오라클

1. 테스트 케이스 (Test Case)
   - 사용자의 요구사항을 정확하게 준수했는지 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서

2. 테스트 시나리오 (Test Scenario)
   - 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스를 묶은 집합

3. 테스트 오라클 (Test Oracle)

- 테스트 결과가 올바른지 판단하기 위해 사전에 정의된 참 값을 대입하여 비교하는 기법

- 특징
  1. 제한된 검증 - 테스트 오라클은 모든 테스트 케이스에 적용 불가
  2. 수학적 기법 - 테스트 오라클의 값을 수학적 기법으로 구할 수 있음
  3. 자동화 가능 - 테스트 대상 프로그램의 실행, 결과 비교 등을 자동화 가능

- 종류
  1. 참 (True) 오라클

     : 모든 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하는 오라클

     : 발생된 모든 오류 검출 가능
  2. 샘플링 (Sampling) 오라클

     : 특정 몇몇 테스트 케이스의 입력 값들에 대해서만 기대하는 결과를 제공하는 오라클

     : 전수 테스트 불가능 한 경우 사용
  3. 추정 (Heuristic) 오라클

     : 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공, 나머지는 추정으로 처리하는 오라클
  4. 일관성 (Consistent) 오라클

     : 애플리케이션 변경이 있을 때, 테스트 케이스 수행 전 후의 결과 값이 동일한지 확인하는 오라클

테스트 자동화 도구

1. 테스트 자동화
   - 스크립트 형태로 구현하는 자동화 도구를 적용하여 쉽고 효율적인 테스트를 수행하는 것

2. 정적 분석 도구 (Static Analysis Tools)
   - 프로그램 수행 없이 분석하는 도구
   - 소스 코드의 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함 발견
   - pmd, cppcheck, SonarQube, checkstyle, ccm, cobertura

3. 동적 분석 도구 (Dynamic Analysis Tools)
   - 작성한 소스 코드를 실행하여 코드에 존재하는 메모리 누수, 스레드 결함 등을 분석하는 코드
   - Avalanche, Valgrind

4. 테스트 실행 도구 (Test Execution Tools)
   - 스크립트 언어를 사용하여 테스트를 실행하는 도구
   - 데이터 주도 접근 방식

     : 스프레드시트에 테스트 데이터를 저장하고 읽어 실행하는 방식
   - 키워드 주도 접근 방식

     : 스프레드 시트에 테스트를 수행할 동작을 나타내는 키워드와 테스트 데이터를 저장하여 실행하는 방식

5. 성능 테스트 도구 (Performance Test Tools)
    - 가상의 사용자를 만들어 테스트를 수행함으로써 성능의 목표 달성 여부를 확인하는 도구

6. 테스트 통제 도구 (Test Control Tools)
    - 테스트 계획 및 관리, 테스트 수행, 결함 관리등을 수행
    - 형상 관리 도구, 결함 추적/관리 도구

7. 테스트 하네스 도구 (Test Harness Tools)
    - 테스트가 실행될 환경을 시뮬레이션 하여 컴포넌트 및 모듈이 정상적으로 테스트되도록 하는 도구
    - 테스트 하네스 - 테스트를 지원하기 위해 생성된 코드와 데이터
    - 구성 요소
        1. 테스트 드라이버 (Test Driver)
        2. 테스트 스텁 (Test Stub)
        3. 테스트 슈트 (Test Suites)
        4. 테스트 케이스 (Test Case)
        5. 테스트 스크립트 (Test Script)
        6. 목 오브젝트 (Mock Object)

애플리케이션 성능 분석

1. 애플리케이션 성능
    - 최소의 자원을 사용하여 최대한 많은 기능을 신속하게 처리하는 정도
    - 지표
        1. 처리량 (Throughput)

           : 일정 시간 내에 처리하는 양
        2. 응답 시간 (Response Time)

           : 요청을 전달한 시간부터 응답이 도착할 때 까지 걸린 시간
        3. 경과 시간 (Turn Around Time)

           : 작업을 의뢰한 시간부터 처리가 완료될 때 까지 걸린 시간
        4. 자원 사용률 (Resource Usage)

           : 의뢰한 작업을 처리하는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등 자원 사용률

2. 성능 테스트 도구
    - 애플리케이션 성능을 테스트하기 위해 부하나 스트레스를 가하면서 성능 측정 지표를 점검하는 도구
    - 종류
        1. JMeter - Cross Platform 지원

           : HTTP, FTP등 다양한 프로토콜을 지원하는 부하 테스트 도구
        2. LoadUI - Cross Platform 지원

           : 서버 모니터링, Drag&Drop 등 사용자의 편리성이 강화된 부하 테스트 도구
        3. OpenSTA - Windows 지원

: HTTP, HTTPS 프로토콜에 대한 부하 테스트 및 생산품 모니터링 도구

3. 시스템 모니터링 도구
   - 애플리케이션이 실행되었을 때 시스템 자원의 사용량을 확인하고 분석하는 도구
   - 종류
     1. Scouter - Cross Platform 지원

        : 단일 뷰 통합 / 실시간 모니터링, 튜닝에 최적화된 인프라 통합 모니터링 도구
     2. Zabbix - Cross Platform 지원

        : 웹기반 서버, 서비스, 애플리케이션 등의 모니터링 도구

애플리케이션 성능 개선

1. 소스 코드 최적화
   - Bad Code를 배제하고 Clean Code로 작성하는 것
   - Clean Code : 누구나 쉽게 이해하고 수정 및 추가할 수 있는 잘 작성된 코드
   - Bad Code : 로직이 복잡하고 이해하기 어려운 코드
     - 스파게티 코드 - 로직이 복잡하게 얽혀 있는 코드
     - 외계인 코드 - 오래되거나 참고문서나 개발자가 없어 유지보수가 힘든 코드

2. 클린 코드 작성 원칙
   a. 가독성
   b. 단순성
   c. 의존성 배제
   d. 중복성 최소화
   e. 추상화