Session049 애플리케이션 테스트

1 애플리케이션 테스트의 개념

1.1 애플리케이션에 잠재되어 있는 결함을 찾아는 행위 또는 절차

- 요구사항 만족 확인(Validation)
- 기능 정확히 수행하는지 검증(Verification)
- 개발 소프트웨어 유형 분류, 특성을 정리, 중점적으로 테스트할 사항 정리

- 상용 소프트웨어 : 보통 사용자들이 공통적으로 필요로 하는 기능 제공 소프트웨어
  - ■ 산업 범용 소프트웨어
    - ◆ 시스템 소프트웨어 : 하드웨어 전체 제어, 운영
    - ◆ 미들웨어 : 운영체제와 응용 프로그램 사이에서 추가적인 서비스 제공하
    - ◆ 응용 소프트웨어 : 특정 업무를 처리하기 위한 소프트웨어
  - ■ 산업 특화 소프트웨어
    - ◆ 특정 분야에서 요구하는 기능만 구현
- 서비스 제공 소프트웨어 : 특정 사용자가 필요로 하는 기능 소프트웨어
  - ■ 신규 개발 소프트웨어 : 새로운 서비스
  - ■ 기능 개선 소프트웨어 : 기존 서비스 기능 개선
  - ■ 추가 개발 소프트웨어 : 기존 시스템에 새로운 기능 추가
  - ■ 시스템 통함 소프트웨어 : 원스톱 서비스로 제공 위해 업무 기능, 데이터 등을 통합하여 개발

2 애플리케이션 테스트의 필요성

- 프로그램 실행 전에 오류 발견하여 예방
- 요구사항이나 기대 수준을 만족시키는지 확인, 제품 신뢰도 향상
- 새로운 오류의 유입도 예방
- 최소한의 시간과 노력으로 많은 결함 발견 가능

3 애플리케이션 테스트의 기본 원리

- 결함이 없다고 증명 불가
- 개발자의 특성이나 기능적 특징으로 특정 모듈에 집중 (파레토 법칙)
- 테스트 케이스 지속적 보완 및 개선(살충제 패러독스 Pesticide Paradox)
- 소프트웨어 특징, 테스트 환경, 테스터 역량 등 정황에 따라 테스트를 다르게 수행
- 결함 모두 제거해도 요구사항 만족 못하면 품질이 높다고 할 수 없음 (오류-부재의 궤변 Absence of Errors Fallacy)
- 테스트 위험 반비례
- 작은 부분에서 점점 확대하며 진행
- 개발자와 관계없는 별도의 팀에서 수행

Session050 애플리케이션 테스트의 분류

1 프로그램 실행 여부에 따른 테스트

1.1 실행 여부에 따라 정적 테스트, 동적 테스트

1.1.1 정적 테스트

- 소스 코드를 대상으로 테스트
- 개발 초기에 결함 발견, 비용 낮춤
- 워크스루, 인스펙션, 코드 검사

1.1.2 동적 테스트

- 실행하여 오류 찾는 테스트, 모든 단계에서 테스트 수행 가능
- 블랙박스 테스트, 화이트박스 테스트

2 테스트 기반(Test Bases)에 따른 테스트

2.1 무엇을 기반으로 하느냐, 명세 기반, 구조 기반, 경험 기반 테스트

2.1.1 명세 기반 테스트

- 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 테스트
- 동등 분할, 경계 값 분석

2.1.2 구조 기반 테스트

- 내부의 논리 흐름에 따라 테스트 케이스 작성 확인
- 구문 기반, 결정 기반, 조건 기반, 결정 기반

2.1.3 경험 기반 테스트

- 테스터의 경험
- 요구사항에 대한 명세가 불충분하거나 시간 제약이 있는 경우
- 에러 추정, 체크 리스트, 탐색적 테스팅

3 시각에 따른 테스트

3.1 누구를 기준으로, 검증(Verification), 확인(Validation)

3.1.1 검증(Verification Test)

- 개발자의 시각에서, 제품이 명세서대로 완성됐는지

3.1.2 확인(Validation Test)

- 사용자의 시각에서, 요구대로 제품이 완성됐는지

4 목적에 따른 테스트

4.1 무엇을 목적으로, 회복(Recovery), 안전(Security), 강도(Stress), 성능(Performance), 구조(Structure), 회귀(Regression), 병행(Parallel)

4.1.1 회복(Recovery)

- 실패하도록 한 후 올바르게 복구되는지

4.1.2 안전(Security)
- 시스템 보호 도구가 불법적인 침입으로부터 보호하는지

4.1.3 강도(Stress)
- 과도한 정보량이나 빈도 부과하여 과부하 시에도 정상적으로 작동하는지

4.1.4 성능(Performance)
- 실시간 성능이나 전체적인 효율성 진단, 응답 시간, 처리량 테스트

4.1.5 구조(Structure)
- 논리적인 경로, 소스 코드의 복잡도 등 평가

4.1.6 회귀(Regression)
- 변경 또는 수정된 코드에 새로운 결함이 없을 확인

4.1.7 병행(Parallel)
- 변경된 SW, 기존 SW에 동일 데이터 입력, 결과 비교

Session051 테스트 기법에 따른 애플리케이션 테스트

1 화이트박스 테스트(White Box Test)

1.1 원시 코드를 오픈시킨 상태에서 코드의 논리적인 모든 경로를 테스트하여 테스트 케이스를 설계하는 방법
- 설계된 절차에 초점 둔 구조적 테스트, 프로시저 설계의 제어 구조 사용하여 테스트 케이스 설계, 테스트 과정 초기에 적용
- 모듈 안의 작동을 직접 관찰
- 모든 문장을 한 번 이상 실행
- 분기점 부분들을 수행함으로써 논리적 경로 제어

2 화이트박스 테스트의 종류

2.1 기초 경로 검사
- 절차적으로 설계의 논리적 복잡성 측정
- 실행 경로의 기초를 정의하는데 지침으로 사용

2.2 제어 구조 검사
- 조건 검사(Condition Testing) : 논리적 조건 테스트
- 루프 검사(Loop Testing) : 반복 구조에 초점
- 데이터 흐름 검사(Data Flow Testing) : 변수의 정의와 변수 사용 위치에 초점

3 화이트박스 테스트의 검증 기준

3.1 문장 검증 기준(Statement Coverage)
- 모든 구문이 한 번 이상 수행되도록 테스트 케이스 설계

3.2 분기 검증 기준(Branch Coverage)
- 모든 조건문이 한 번 이상 수행되도록 설계

3.3 조건 검증 기준(Condition Coverage)

- 모든 조건문에 대한 조건이 True, False 인 경우 모두 한 번 이상 수행되도록 설계

3.4 분기/조건 기준(Branch/Condition Coverage)

- 모든 조건문과 각 조건문에 포함된 조건식의 결과가 True, False 인 경우 모두 한 번 이상 수행 되도록 설계

4 블랙박스 테스트(Black Box Test)

4.1 수행할 특정 기능을 알기 위해서, 완전히 작동되는 입증, 기능 테스트

- 요구사항 명세를 보면서 테스트, 구현된 기능 테스트
- 인터페이스에서 실시
- 부정확하거나, 누락된 기능, 인터페이스 오류, 자료구조, 외부 DB 접근에 따른 오류, 행위나 성능 오류, 초기화와 종료 오류 등을 발견 위해, 테스트 과정 후반에 적용

5 블랙박스 테스트의 종류

5.1 동치 분할검사(Equivalence Partitioning Testing)

- 입력자료에 초점
- 타당한 입력 자료, 타장하지 않은 입력 자료의 개수를 균등하게 테스트 케이스, 결과 확인

5.2 경계값 분석(Boundary Value Analysis)

- 입력에만 치중한 동치 분할 기법 보완
- 입력 조건의 경계값을 테스트 케이스로 선정

5.3 원인-효과 그래프 검사(Cause-Effect Graphing Testing)

- 입력 데이터 간 관계와 출력에 영향을 미치는 상황 체계적 분석 후 효용성 높은 테스트 케이스 선정

5.4 오류 예측 검사(Error Guessing)

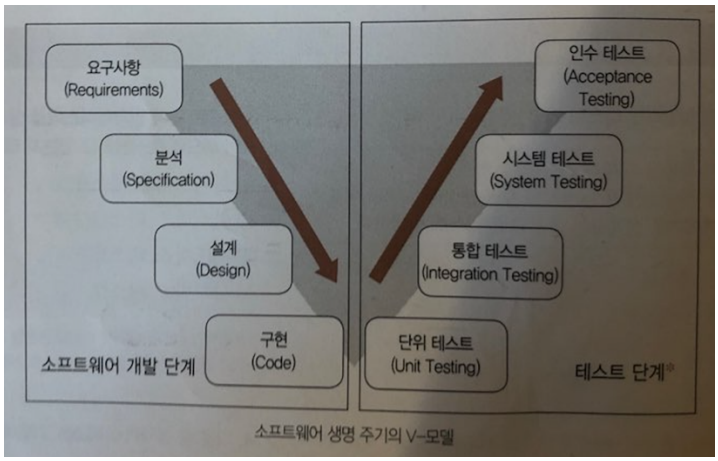
- 경험이나 감각으로 테스트
- 다른 블랙 박스 기법으로 찾을 수 없는 오류를 찾는 보충적 검사 기법, 데이터 확인 검사

5.5 비교 검사(Comparison Testing)

- 여러 버전의 프로그램에 동일한 테스트 자료 제공, 동일한 결과 나오는지

Session052 개발 단계에 다른 애플리케이션 테스트

1 개발 단계에 따른 애플리케이션 테스트



소프트웨어 생명 주기의 V-모델

2 단위 테스트(Unit Test)

2.1 코딩 직후 모듈이나 컴포넌트에 초점

- 인터페이스, 외부적 I/O, 자료 구조, 독립적 기초 경로, 오류 처리 경로, 경계 조건 등 검사
- 구조 기반 테스트
  - ■ 내부 구조 및 복잡도 검증, 화이트박스 테스트 시행
  - ■ 제어 흐름, 조건 결정
- 명세 기반 테스트
  - ■ 목적 및 실행 코드 기반 블랙박스 테스트 시행
  - ■ 동등 분할, 경계 값 분석

3 통합 테스트(Integration Test)

3.1 단위 테스트 완료된 모듈들 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트

- 모듈 간 또는 통합된 컴포넌트 간 상호 작용 오류 검사

4 시스템 테스트(System Test)

4.1 컴퓨터에서 완벽하게 수행되는가

- 환경적인 장애 리스크 최소화 위해 실제 사용 환경과 유사하게 테스트 환경 구축
- 기능적 요구사항
  - ■ 요구사항 명세서, 비즈니스 절차, 유스케이스 등 명세서 기반 블랙박스 테스트 시행
- 비기능적 요구사항
  - ■ 성능 테스트, 회복 테스트, 보안 테스트, 내부 시스템의 메뉴 구조, 웹 페이지

의 네비게이션 등 구조적 요소에 대한 화이트박스 테스트 시행

5 인수 테스트(Acceptance Test)

5.1 사용자의 요구사항을 충족하는지 중점

- 사용자가 직접 테스트
- 문제 없으면 인수하고 프로젝트 종료
- 사용자 인수 테스트
  - ■ 사용의 적절성 여부 확인
- 운영상의 인수 테스트
  - ■ 시스템 관리자가 시스템 인수 시 수행하는 테스트 기법, 백업/복원 시스템, 재난 복구, 사용자 관리, 정기 점검 등을 확인
- 계약 인수 테스트
  - ■ 계약상의 인수/검수 조건을 준수하는지
- 규정 인수 테스트
  - ■ 정부 지침, 법규, 규정 등에 맞게 개발되었는지
- 알파 테스트
  - ■ 사용자가 개발자 앞에서 행하는 테스트
  - ■ 통제된 환경, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인
- 베타 테스트
  - ■ 최종 사용자가 여러 명의 사용자 앞에서 테스트
  - ■ 실업무를 가지고 직접 테스트, 개발자에 의해 제어되지 않은 상태에서 테스트, 발견된 오류와 문제점을 기록, 개발자에게 주기적으로 보고

Session053 통합 테스트

1 통합 테스트(Integration Test)

1.1 모듈 통합 과정에서 발생하는 오류 및 결함 찾는

1.2 비점진적 통합 방식

- 모든 모듈이 미리 결합되어 있는 프로그램 전체를 테스트, 빅뱅 통합 테스트
- 규모 작은 SW에 유리 단시간 내에 테스트 가능
- 오류 발견 및 장애 위치 파악 및 수정이 어려움

1.3 점진적 통합 방식

- 단계적으로 통합하면서 테스트, 하향식, 상향식, 혼합식
- 오류 수정 용이, 인터페이스와 연관된 오류 완전히 테스트할 가능성 높다

2 하향식 통합 테스트(Top Down Integration Test)

2.1 상위 모듈 -> 하위 모듈

2.2 주요 제어 모듈을 기준으로 아래 단계로 이동. 깊이 우선 통합법, 넓이 우선 통합법
2.3 테스트 초기부터 시스템 구조를 보여줄 수 있다
2.4 상위 모듈에서 테스트 케이스를 사용하기 어려움
2.5 순서
- 주요 제어 모듈의 종속 모듈들을 스텁(Stub)으로 대체
- 통합 방식에 따라 스텁들이 한 번에 하나씩 실제 모듈로 교체
- 모듈이 통합될 때마다 테스트
- 회귀 테스트 실시

3 상향식 통합 테스트(Bottom Up Integration Test)
3.1 하위 -> 상위
3.2 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터(Cluster)가 필요
3.3 순서
- 하위 모듈들을 클러스터로 결합
- 입출력 확인 위해 드라이버(Driver) 작성
- 통합된 클러스터 단위로 테스트
- 완료 후 클러스터는 구조의 상위로 이동, 결합하고 드라이버는 실제 모듈로 대체

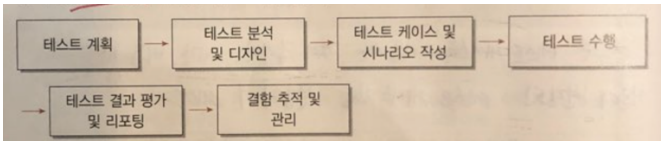
4 혼합식 테스트
4.1 샌드위치식 통합 테스트

5 회귀 테스팅
5.1 테스트된 프로그램의 테스팅 반복
5.2 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인
- 수정 부분이 다른 부분에 영향을 미치는지, 오류가 생겼는지, 새로운 오류가 없음을 보증하기 위해
- 모든 테스트 케이스 이용이 Best, 하지만 시간 비용이 많이 필요, 변경된 부분을 테스트
- 테스트 케이스 선정 방법
  - ■ 모든 기능을 수행할 수 있는 대표 테스트 케이스 선정
  - ■ 파급 효과 분석, 높은 부분이 포함된 테스트 케이스 선정
  - ■ 실제 수정이 발생한 모듈 또는 컴포넌트에서 시행되는 테스트 케이스 선정

Session054 애플리케이션 테스트 프로세스

1 애플리케이션 테스트 프로세스

1.1 테스트하는 절차



1.2 산출물

- 테스트 계획서 : 목적, 범위, 일정, 수행 절차, 대상 시스템 구조, 조직의 역할 및 책임 계획 문서
- 테스트 케이스 : 요구사항 얼마나 준수하는지 입력 값, 실행 조건, 기대 결과 등 테스트 항목의 명세서
- 테스트 시나리오 : 여러 개의 테스트 케이스의 동작 순서 기술
- 테스트 결과서 : 비교 분석한 내용 정리

2 테스트 계획

2.1 계획서, 요구 명세서 등 기반 테스트 목표, 대상, 범위 결정

- 대상의 구조 파악
- 조직 및 비용 산정
- 시작 및 종료 조건 정의
  - ■ 시작 조건 : 모든 조건 만족하지 않아도 시작하도록 지정 가능
  - ■ 종료 조건 : 테스트 완료, 일정 만료, 비용 소진 등 중요도에 따라 종료 조건 다르게 지정 가능
- 계획서 작성

3 테스트 분석 및 디자인

3.1 테스트 목적과 원칙 검토, 요구사항 분석

- 리스크 분석 및 우선순위 결정
- 데이터, 환경, 도구 등 준비

4 테스트 케이스 및 시나리오 작성

4.1 설계 기법에 따라 TC 작성, 검토 및 확인 후 테스트 시나리오 작성

- 테스트용 스크립트 작성

5 테스트 수행

5.1 환경 구축 후 수행

- 결과 측정하여 기록

6 테스트 결과 평가 및 리포팅

6.1 결과를 비교 분석, 결과서 작성

- 결함 내용, 결함 재현 순서 등 경함을 중점으로
- 실행 절차의 리뷰 및 결과 평가 수행, 실행 절차 최적화하여 다음 테스트 적용

7 결함 추적 및 관리

7.1 어디, 어떤 종류인지, 추적하고 관리

- 동일한 결함 발견 시 처리 시간 단축 및 재발 방지

- 결함 관리 프로세스
  - ■ 에러 발견 : 테스트 전문가, 프로젝트 팀 논의
  - ■ 에러 등록 : 에러를 결함 관리 대장에 등록
  - ■ 에러 분석 : 등록 에러가 실제 결함인지 분석
  - ■ 결함 확정 : 실제 결함이면 결함 확정 상태로 설정
  - ■ 결함 할당 : 담당자에게 결함 할당하고 할당 상태로 설정
  - ■ 결함 조치 : 결함 수정하고, 수정 완료되면 조치 상태로 설정
  - ■ 결함 조치 검토 및 승인 : 확인 테스트 수행, 이상 없으면 결함 조치 완료 상태로 설정
- 에러/오류는 결함/결점/버그의 원인

Session055 테스트 케이스/테스트 시나리오/테스트 오라클

1 테스트 케이스(Test Case)

1.1 요구사항 준수했는지 확인 위해 입력 값, 실행 조건, 기대 결과 등 테스트 항목 명세서 명세 기반 테스트의 설계 산출물

- 미리 설계 시 테스트 오류 방지, 필요 인력, 시간 등의 낭비 줄일 수 있다
- 시스템 설계 시 작성하는게 이상적

2 테스트 케이스 작성 순서

2.1 테스트 전략, 계획성 등을 기반

2.1.1 테스트 계획 검토 및 자료 확보

- 계획서 재검토, 대상 범위 및 접근 방법 이해
- 요구사항 기능 명세서 검토, 대상 시스템의 정보 확보

2.1.2 위험 평가 및 우선순위 결정

- 결함의 위험 정도로 우선순위 결정, 어느 부분에 초점을 둘지 결정

2.1.3 테스트 요구사항 정의

- 요구사항, 테스트 대상을 재검토하고 특정, 조건, 기능 등을 분석

2.1.4 테스트 구조 설계 및 테스트 방법 결정

- 형식과 분류 방법 결정
- 절차, 장비, 도구, 문서화 방법 결정

2.1.5 테스트 케이스 정의
- TC 작성, 입력 값, 실행 조건, 예상 결과 등을 기술

2.1.6 테스트 케이스 타당성 확인 및 유지 보수
- 기능, 환경 변화에 따라 TC 갱신
- 유용성 검토

3 테스트 시나리오(Test Scenario)

3.1 TC 적용 순서에 따라 여러 개의 TC 묶은 집합. 적용하는 구체적인 절차를 명세한 문서
- 테스트 순서에 대한 구체적 절차, 사전 조건, 입력 데이터 등 설정되어 있다
- 테스트 순서를 미리 정해 빠짐없이 수행

4 테스트 시나리오 작성 시 유의 사항
- 시스템별, 모듈별, 항목별 등 여러 개의 시나리오로 분리 작성
- 요구사항, 설계 문서 등을 토대로 작성
- 각 테스트 항목은 식별자 번호, 순서 번호, 테스트 데이터, 테스트 케이스, 예상 결과, 확인 등을 포함
- 유스케이스 간 업무 흐름이 정상적인지 테스트 할 수 있어야 한다
- 개발된 모듈 또는 프로그램 간 연계가 정상적인지 테스트 할 수 있어야한다

5 테스트 오라클(Test Oracle)

5.1 결과가 올바른지 판단 위해 정의된 참 값을 대입하여 비교하는 기법 및 활동.

5.2 테스트 결과 판단 위함

5.3 테스트 오라클 특징
- 제한된 검증 : 모든 테스트 케이스에 적용 X
- 수학적 기법 : 수학적 기법을 이용하여 구할 수 있음
- 자동화 기능 : 대상의 실행, 결과 비교, 커버리지 측정 등을 자동화 가능

6 테스트 오라클의 종류

6.1 참(True) 오라클
- 모든 TC의 입력 값에 기대하는 결과를 제공, 발생된 모든 오류 검출 가능
- 미션 크리티컬한 업무에 사용

6.2 샘플링(Sampling) 오라클
- 특정 TC의 입력 값들에 대해서만 기대하는 결과 제공
- 일반적 업무, 게임, 오락 등에 사용

6.3 추정(Heuristic) 오라클
- 샘플링 개선, 입력 값에 대해 기대하는 결과를 제공, 나머지 입력 값은 추정 처리
- 일반적 업무, 게임, 오락 등에 사용

6.4 일관성 검사(Consistent) 오라클

- 변경이 있을 때, TC 수행 전, 후의 결과 값이 동일한지를 확인

Session056 테스트 자동화 도구

1 테스트 자동화의 개념

1.1 테스트 절차를 스크립트 형태로 구현하는 자동화 도구 적용, 쉽고 효율적 수행 가능

1.2 휴먼 에러 줄이고, 정확성 유지, 품질 향상

2 테스트 자동화 도구의 장점/단점

2.1 장점

- 인력 및 시간 줄임
- 향상된 테스트 품질
- 일관성 있게 검증
- 객관적인 평가 기준 제공
- 다양한 표시 형태로 제공
- UI 없는 서비스도 정밀 테스트 가능

2.2 단점

- 교육 및 학습 필요
- 적용하기 위한 시간, 비용, 노력 필요
- 비공개 상용 도구는 고가

3 테스트 자동화 수행 시 고려사항

3.1 재사용 및 측정 불가능한 테스트 프로그램 제외

3.2 모든 과정 자동화 도구는 없다. 용도에 맞는 적절한 도구 선택

3.3 도구의 환결 설정 및 습득 기간 고려해 프로젝트 일정 계획

3.4 테스트 엔지니어 투입 시기 늦으면 프로젝트 이해 부족으로 인한 분완전 테스트 초래

4 테스트 자동화 도구의 유형

4.1 정적 분석 도구(Static Analysis Tools)

- 소스 코드에 대한 코딩 표준, 스타일, 복잡도, 남은 결함 등 발견 위해
- 테스터가 소스 코드를 이해하고 있어야 분석 가능

4.2 테스트 실행 도구(Test Execution Tools)

- 스크립트 언어 사용, 데이터와 수행 방법 등이 포함된 스크립트 작성 후 실행
- 데이터 주도 방식
  - ■ 스프레드시트에 데스트 데이터 저장, 이를 읽어 실행
  - ■ 다양한 데이터를 동일한 케이스로 반복 실행 가능
  - ■ 스크립트 익숙지 않은 사용자도 미리 작성된 스크립트에 테스트 데이터만 추가하여 테스트 가능
- 키워드 주도 접근 방식
  - ■ 스프레드시트에 수행할 동작 나타내는 키워드와 데이터 저장하여 실행
  - ■ 키워드 이용하여 테스트 정의 가능

4.3 성능 테스트 도구(Performance Test Tools)

- 처리량, 응답 시간, 경과 시간, 자원 사용률 등을 인위적으로 적용한 가상의 유저를 만들어 수행하여 목표 달성 여부 확인

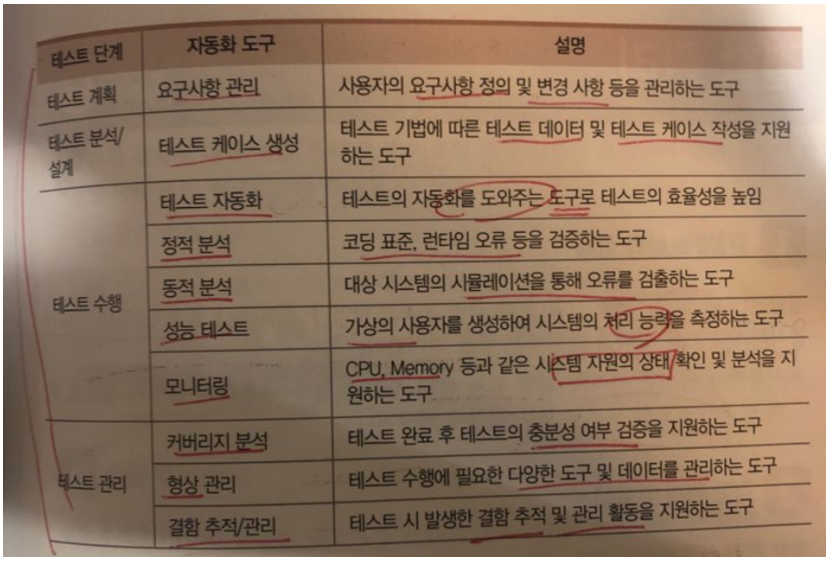
4.4 테스트 통제 도구(Test Control Tools)

- 계획 및 관리, 수행, 결함 관리 등을 수행하는 도구, 형상 관리 도구, 결함 추적/관리 도구

4.5 테스트 하네스 도구(Test Harness Tools)

- 컴포넌트 및 모듈을 테스트하는 환경의 일부분, 테스트 지원 위해 생성된 코드와 데이터를 의미
- 테스트가 실행될 환경을 시뮬레이션, 컴포넌트 및 모듈이 정상적으로 테스트 되도록 한다

5 테스트 수행 단계별 테스트 자동화 도구



테스트 단계	자동화 도구	설명
테스트 계획	요구사항 관리	사용자의 요구사항 정의 및 변경 사항 등을 관리하는 도구
테스트 분석/설계	테스트 케이스 생성	테스트 기법에 따른 테스트 데이터 및 테스트 케이스 작성을 지원하는 도구
테스트 수행	테스트 자동화	테스트의 자동화를 도와주는 도구로 테스트의 효율성을 높임
	정적 분석	코딩 표준, 런타임 오류 등을 검증하는 도구
	동적 분석	대상 시스템의 시뮬레이션을 통해 오류를 검출하는 도구
	성능 테스트	가상의 사용자를 생성하여 시스템의 처리 능력을 측정하는 도구
	모니터링	CPU, Memory 등과 같은 시스템 자원의 장태 확인 및 분석을 지원하는 도구
테스트 관리	커버리지 분석	테스트 완료 후 테스트의 충분성 여부 검증을 지원하는 도구
	형상 관리	테스트 수행에 필요한 다양한 도구 및 데이터를 관리하는 도구
	결함 추적/관리	테스트 시 발생한 결함 추적 및 관리 활동을 지원하는 도구

Session057 결함 관리

1 결함(Fault)의 정의

1.1 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생되는 것 의미

- 예상 결과와 실행 결과 간 차이나 업무 내용과의 불일치 등으로 인해 변경이 필요한 부분도 결함

2 결함 관리 프로세스

2.1 결함 관리 계획

- 전체 프로세스에 대한 결함 관리 일정, 인력, 업무 프로세스 등을 확보, 계획 수립

2.2 결함 기록

- 발견된 결함을 결함 관리 DB에 등록

2.3 결함 검토

- 테스터, 프로그램 리더 , 품질 관리 담당자 등은 등록된 결함 검토, 수정할 개발자에게 전달

2.4 결함 수정

- 개발자는 결함을 수정

2.5 결함 재확인

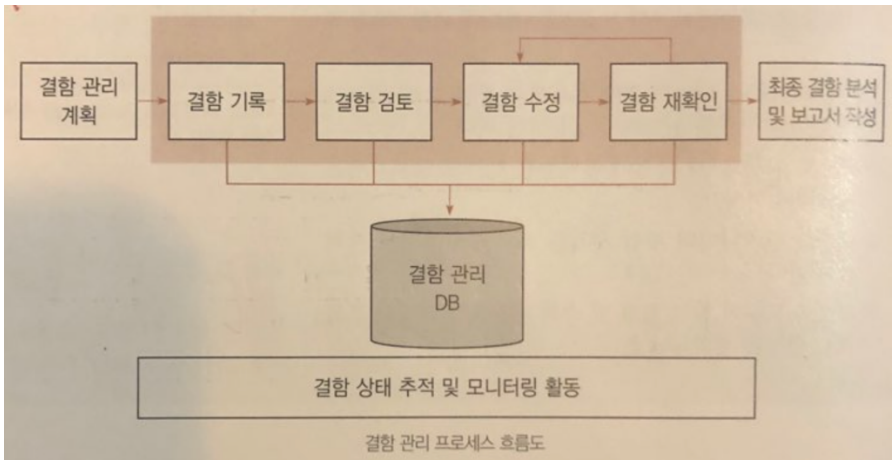
- 수정 내용 확인하고 다시 테스트 수행

2.6 결함 상태 추적 및 모니터링 활동

- DB를 이용 결함 유형, 발생률 등을 대시보드 또는 게시판 형태의 서비스를 제공

2.7 최종 결함 분석 및 보고서 작성

- 결함 정보와 이해관계자들의 의견이 반영된 보고서 작성, 결함 관리 종료



결함 관리 프로세스 흐름도

3 결함 상태 추적

3.1 지속적으로 상태 변화를 추적, 관리해야 함

3.2 결함 관리 측정 지표를 분석, 향후 결함 발견될 모듈 또는 컴포넌트 추정 가능

3.3 결함 관리 측정 지표

- 결함 분포 : 특정 속성에 해당하는 결함 수 측정
- 결함 추세 : 진행 시간에 따른 결함 수의 추이 분석
- 결함 에이징 : 결함 상태로 지속되는 시간 측정

4 결함 추적 순서

4.1 결함이 발견된 때 ~ 해결될 때

4.1.1 결함 등록(Open) : QA에 의해 발견되 결함 등록된 상태

4.1.2 결함 검토(Reviewed) : 테스터, QA, 프로그램 리더, 개발자에 의해 검토된 상태

4.1.3 결함 할당(Assigned) : 수정 위해 개발자, 문제 해결 담당자에게 할당된 상태

4.1.4 결함 수정(Resolved) : 결함 수정을 완료한 상태

4.1.5 결함 조치 보류(Deferred) : 수정 불가능, 연기된 상태, 우선순위, 일정에 따라 재오픈을 준비중인 상태

4.1.6 결함 종류(Closed) : 해결되어 QA가 종료 승인

4.1.7 결함 해제(Clarified) : 종료 승인 된 결함을 검토, 결함이 아니라고 판명

5 결함 분류

5.1 유형별 결함

- 시스템 결함 : 애플리케이션 환경, 데이터베이스 처리에서 발생된 결함
- 기능 결함 : 애플리케이션의 기획, 설계, 업무 시나리오 등의 단계에서 유입된 결함
- GUI 결함 : 사용자 화면 설계에서 발생된 결함
- 문서 결함 : 기획자, 사용자, 개발자 간의 의사소통 및 기록이 원할치 안하 발생한 결함

5.2 단계별 유입 결함

- 기획 시 : 사용자 요구사항 표준 미준수, 요구사항 불명확/불완전/불일치 결함
- 설계 시 : 설계 표준 미준수, 기능 설계 불명확/불완전/불일치
- 코딩 시 : 코딩 표준 미준수, 기능의 불일치/불완전, 데이터 결함, 인터페이스 결함
- 테스트 부족 : 테스트 완료 기준의 미준수, 테스트팀 개발팀의 의사소통 부족, 개발자 코딩 실수

6 결함 심각도

6.1 결함이 시스템에 미치는 치명도의 척도

- High : 더 이상 프로세스를 진행할 수 없도록 만드는 결함
- Medium : 시스템 흐름에 영향을 미치는 결함
- Low : 시스템 흐름에는 영향을 미치지 않는 결함

7 결함 우선순위

7.1 결함 처리에 대한 신속성의 척도, 결함의 중요도와 심각도에 따라 설정, 수정 여부 결정된다

- 보통 심각도 우선순위 비례, 애플리케이션 특정에 따라 우선순위 결정될 수도, 무조건 심각도 우선순위 비례는 아니다

- Critical, High, Medium, Low

8 결함 관리 도구

8.1 결함을 체계적으로 관리 도와주는 도구

- Mantis : 결함 및 이슈 관리 도구, 설계 시 단위별 작업 내용 기록, 결함 추적 가능
- Trac : 결함 추적, 결함 통합 관리 할 수 있는 도구
- Redmine : 프로젝트 관리 및 결함 추적 가능한 도구
- Bugzilla : 결함 신고, 확인, 처리 등 지속적 관리할 수 있는 도구, 심각도 우선순위 지정할 수 있다