**61. 애플리케이션 테스트**

애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차

고객의 요구사항을 만족시키는지 확인(Validation), 기능을 정확히 수행하는지 검증(Verification)

상용 소프트웨어 – 보통의 사용자들이 공통적으로 필요로 하는 기능 제공

Ex) 산업 범용 소프트웨어(시스템 소프트웨어, 미들웨어, 응용 소프트웨어), 산업 특화 소프트웨어

서비스 제공 소프트웨어 – 특정 사용자가 필요로 하는 기능만을 구현해서 제공

Ex) 신규개발 소프트웨어, 기능개선 소프트웨어, 추가 개발 소프트웨어, 시스템 통합 소프트웨어

2. 애플리케이션 테스트의 필요성

프로그램 실행 전에 오류 발견하여 예방 가능, 반복 테스트로 신뢰도 향상, 오류 유입 예방

3. 애플리케이션 테스트의 기본 원리

완벽한 테스팅은 불가능, 잠재적인 결함↓, 결함은 대부분 특정 모듈에 집중됨(파레토 법칙)

테스트↑위험↓, 작은 부분에서부터 진행, 별도의 팀에서 수행, 지속적인 테스트 케이스 개선 필요

**62. 애플리케이션 테스트의 분류**

1. 프로그램 실행 여부에 따른 테스트

정적 테스트 – 명세서나 소스 코드를 대상으로 분석, 개발 초기에 결함 발견 가능, 개발 비용↓

동적 테스트 – 프로그램을 실행하여 오류를 찾음, 개발의 모든 단계에서 테스트 진행 가능

2. 테스트 기반(Test Bases)에 따른 테스트

|  |  |
| --- | --- |
| 명세 기반 | 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 구현중인지 확인하는 것 |
| 구조 기반 | 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인 |
| 경험 기반 | 테스터의 경험을 기반으로 수행하는 테스트, 명세 불충분, 시간 제약시 수행 |

3. 시각에 따른 테스트

검증(Verification) – 개발자의 시각에서 제품의 생산 과정 테스트, 명세서대로 작성됐는지를 테스트

확인(Validation) – 사용자의 시각, 요구한대로 완성됐는지, 정상적으로 동작하는지를 테스트

4. 목적에 따른 테스트

회복(Recovery) : 시스템에 여러 가지 결함을 주어 실패하도록 한 후 올바르게 복구되는지를 확인

안전(Safe) : 시스템에 설치된 시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지 확인

강도(Stress) : 시스템에 과도한 정보량이나 빈도 등을 부과해 과부하 시에도 SW가 정상적으로 실행되는지 확인

성능(Perfomance) : SW의 실시간 성능이나 전체적인 효율성을 진단하는 테스트, 응답시간/처리량을 테스트

구조(Structure) : SW내의 논리적인 경로, 소스 코드의 복잡도 등을 평가하는 테스트

회귀(Regression) : 소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인하는 테스트

병행(Parallel) : 변경된 SW와 기존 소프트웨어에 동일한 데이터를 입력하여 결과를 비교하는 테스트

**63. 테스트 기법에 따른 애플리케이션 테스트**

1. 화이트박스 테스트(White Box Test)

원시코드를 오픈 시킨 상태에서 논리적인 모든 경로를 테스트하여 테스트 케이스 설계하는 방법

2. 화이트박스 테스트의 종류

기초 경로 검사 – 절차적 설계의 논리적 복잡성 측정 가능, 실행 경로의 기초를 정의하는데 사용

제어 구조 검사 – 조건 검사(Condition Testing), 루프 검사, 데이터 흐름 검사(Data Flow Testing)

3. 화이트박스 테스트의 검증 기준

테스트 케이스들이 테스트에 얼마나 적정한지를 판단하는 기준

문장 검증 기준(Statement Coverage) – 소스 코드의 모든 구문이 한번 이상 수행되도록 설계

분기 검증 기준(Branch Coverage) – 소스 코드의 모든 조건문이 한번 이상 수행되도록 설계

조건 검증 기준(Condition) – 모든 조건문에 대해 True인 경우와 False인 경우 한번 이상 수행

분기/조건 기준(Branch/Condition) – 모든 조건문, 조건문에 포함된 개별 조건식의 결과가 True인 경우와 False인 경우가 한번 이상 수행되도록 테스트 케이스 설계

4. 블랙박스 테스트

소프트웨어가 **수행할 특정 기능을 알기 위해 각 기능이 완전히 작동되는 것을 입증**(기능 테스트)

5. 블랙박스 테스트의 종류

동치 분할 검사, 경계값 분석, 원인-효과 그래프 검사, 오류 예측 검사, 비교 검사

※검증 기준(Coverage)의 종류 – 기능 기반 커버리지, 라인 커버리지, 코드 커버리지

**64. 개발 단계에 따른 애플리케이션 테스트**

소프트웨어의 개발 단계에 따라 분류되며 테스트 레벨이라 지칭함. 다양한 오류 발견 가능,V-모델

요구사항 – 분석 – 설계 – 구현 – 단위테스트 – 통합테스트 - 시스템 테스트 – 인수 테스트

2. 단위 테스트(Unit Test)

코딩 직후 소프트웨어 설계의 최소 단위인 **모듈이나 컴포넌트에 초점을** 맞춰 테스트

인터페이스, 외부적 I/O, 자료 구조, 독립적 기초 경로, 오류 처리 경로, 경계 조건

구조기반 테스트(화이트박스, 제어흐름, 조건 설정), 명세 기반 테스트(블랙박스, 동등 분할, 경계값)

3. 통합 테스트(Integration Test) - 상호 작용 오류 검사

단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트

4. 시스템 테스트(System Test)

개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행되는가를 점검하는 테스트

기능적 요구사항 – 명세서 기반의 블랙박스 테스트 시행

비기능적 요구사항 – 구조적 요소에 대한 화이트박스 테스트 시행(성능, 회복, 메뉴 구조)

5. 인수 테스트(Acceptance Test)

개발한 소프트웨어가 사용자 요구사항을 충족하는지에 중점을 두고 테스트

사용자 인수 테스트, 운영상의 인수 테스트, 계약 인수/규정 인수 테스트, 알파/베타 테스트

**65. 통합 테스트(Integration Test)**

단위 테스트가 끝난 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트 기법

비점진적 통합 방식(프로그램 전체 테스트), 점진적 통합 방식(단계적으로 통합하면서 테스트)

2. **하향식 통합 테스트**(Top Down Integration Test)

**상위 모듈 → 하위 모듈 통합, 깊이/넓이 우선 통합법**

제어모듈-작성 프로그램 사용, 주요 제어 모듈의 종속 모듈들은 **스텁(Stub)으로 대체** → 통합 방식에 따라 스텁들이 실제 모듈로 교체 → 모듈 통합 때마다 테스트 실시 → 회귀 테스트 실시

3. **상향식 통합 테스트**(Bottom Up Integration Test)

하위 모듈 → 상위 모듈 통합, 하나의 주요 제어 모듈과 관련된 종속 모듈 그룹 클러스터 필요

**하위 모듈들을 클러스터로 결합** → 입∙출력 확인 위해 **더미 모듈인 드라이버 작성** → 통합된 클러스터 단위로 테스트 → 클러스터는 상위로 이동해 결합, 드라이버는 실제 모듈로 대체

4. 혼합식 통합 테스트

하위 수준 → 상향식 통합, 상위 수준 → 하향식 통합을 사용. 최적의 테스트 지원

5. 회귀 테스팅(Regression Testing)

이미 테스트된 프로그램의 테스팅을 반복, 통합 테스트로 변경된 모듈이나 컴포넌트 오류 확인

선정방법 – 모든 기능이 수행가능한 대표 테스트 케이스 선정, 파급효과 분석 후 선정, 실제 수정이 발생한 모듈 또는 컴포넌트에서 시행하는 테스트케이스 선정

**66. 애플리케이션 테스트 프로세스**

개발된 소프트웨어가 사용자의 요구대로 만들어졌는지, 결함은 없는지 등을 테스트하는 **절차**

|  |  |
| --- | --- |
| 테스트 계획 | 목표 정의 및 대상/범위 결정, 시스템 구조 파악, 조직 및 비용 산정 |
| 테스트 분석 및 디자인 | 목적과 원칙 검토, 요구사항 분석, 우선순위 결정, 도구 등 준비 |
| 케이스, 시나리오 작성 | 설계 기법을 따름, 검토 및 확인 후 시나리오 작성, 스크립트 작성 |
| 테스트 수행 | 테스트 환경 구축 후 테스트 수행, 테스트 결과 측정 및 기록 |
| 결과 평가 및 리포팅 | 테스트 결과를 비교 분석하여 테스트 결과서 작성, 결함 위주 |
| 결합 추적 및 관리 | 처리 시간 단축, 결함 재발 방지 / 에러 발견, 에러 등록, 에러 분석, 결함 확정, 결함 할당, 결함 조치, 결함 조치 검토 및 승인 |

테스트 계획서, 테스트 케이스, 테스트 시나리오, 테스트 결과서

**67. 테스트 케이스 / 테스트 시나리오 / 테스트 오라클**

1. 테스트 케이스(Test Case)

입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서, 시스템 설계 시 작성

2. 테스트 케이스 작성 순서

계획 검토 및 자료 확보 → 위험 평가 및 우선순위 결정 → 테스트 요구사항 정의 →구조 설계 및 테스트 방법 결정 → 테스트 케이스 정의 → 테스트 케이스 타당성 확인 및 유지 보수

3. 테스트 시나리오(Test Scenario) - 테스트 케이스들을 적용시에 구체적인 절차를 명세한 문서

4. 테스트 시나리오 작성시 유의사항

시스템, 모듈, 항목 동으로 분리 후 작성, Use Case 간 업무흐름 테스트, 동작 테스트(모듈 등)

5. 테스트 오라클(Test Oracle)

테스트 결과가 올바른지 판단하기 위해 정의된 참 값을 대입하여 비교하는 기법 및 활동

제한된 검증, 수학적 기법, 자동화 가능

6. 테스트 오라클의 종류

참(True), 샘플링(Sampling), 추정(Heuristic), 일관성 검사(Consistent)

**68. 테스트 자동화 도구**

테스트 절차를 스크립트 형태로 구현하는 자동화 도구를 적용해 쉽고 효율적으로 테스트 수행

장점 – 인력, 시간↓, 품질 보장, 평가 기준 제공, 정밀 테스트 가능, 일관성 있게 검증

단점 – 사용방법 학습 필요, 시간/비용/노력 필요, 비공개 상용 도구 고가 비용 필요

3. 테스트 자동화 수행 시 고려 사항

재사용 및 측정이 불가능한 프로그램 제외, 용도에 맞는 적절한 도구 선정, 일정 계획

4. 테스트 자동화 도구의 유형

정적 분석 도구(프로그램x), 테스트 실행 도구(스크립트언어), 성능 테스트 도구, 테스트 통제 도구, 테스트 하네스 도구

5. 테스트 수행 단계별 테스트 자동화 도구

테스트 계획 – 요구사항 관리, 테스트 분석/설계 – 테스트 케이스 작성

테스트 수행 – 테스트 자동화, 정적 분석, 동적 분석, 성능 테스트, 모니터링

테스트 관리 – 커버리지 분석, 형상 관리, 결함 추적/관리

**69. 결함 관리**

2. **결함 관리 프로세스**

결함 관리 계획→기록→검토→수정→재확인→상태 추적 및 모니터링 활동→분석 및 보고서 작성

3. 결함 상태 추적

결함 분포(속성 수 측정), 결함 추세(진행 시간 결함 수 추이 분석), 결함 에이징(지속 시간 측정)

4. **결함 추적 순서**

등록 → 검토 → 할당 → 수정(Resolved) → 보류(Differred) → 종료(Closed) → 해제(Clarified)

5. 결함 분류 – 시스템 결함, 기능 결함(기획, 설계 유입), GUI 결함(화면설계에서 발생), 문서 결함

6. 결함 심각도 – High(프로세스 진행 불가 결함), Midium(시스템 흐름 영향), Low(시스템 영향x)

7. 결함 우선순위 – 결함 처리에 대한 신속성을 나타내는 척도

8. 결함 관리 도구 – Mantis, Trac, Redmine, Bugzilla

**70. 애플리케이션 성능 분석**

1. 애플리케이션 성능

사용자 요구 기능을 최소한의 자원을 사용해 최대한 많은 기능을 신속하게 처리하는 정도

**처리량(Throughput), 응답 시간(Response Time), 경과 시간(Tum Around Time), 자원 사용률**

2. 성능 테스트 도구

앱에 부하나 스트레스를 가하면서 성능 측정 지표를 점검하는 도구, JMeter, LoadUI, OpenSTA

3. 시스템 모니터링(Monitoring) 도구 - 시스템 자원의 사용량을 확인하고 분석하는 도구

성능 저하의 원인 분석, 시스템 부하량 분석, 사용자 분석 / 종류 – Scouter, Zabbix

4. 애플리케이션 성능 저하 원인 분석

Connection 객체를 생성하거나 쿼리를 실행하는 앱 로직에서 많이 발생

데이터 요청 과다, 타임 아웃, 연결 누수, Commit 과다 발생, 읽기 오류, 데이터 손실

**71. 애플리케이션 성능 개선**

1. 소스 코드 최적화

클린 코드(Clean Code) : 잘 작성된 코드 / 나쁜 코드(Bad Code) : 스파게티 코드, 로직 중복 코드

작성 원칙 : 가독성, 단순성, 의존성 배제, 중복성 최소화, 추상화

2. 소스 코드 최적화 유형

클래스 분할 배치(응집도↑, 크기↓), 느슨한 결합(인터페이스 클래스 이용), 코딩 형식 준수, 좋은 이름 사용, 적절한 주석문 사용

3. 소스 코드 품질 분석 도구

정적 분석 도구 : 실행x, 코딩 표준, 스타일, 결함 등을 확인

동적 분석 도구 : 소스 코드를 실행하여 코드에 존재하는 메모리 누수, 스레드 결함 등을 분석

4. 소스 코드 품질 분석 도구의 종류

Pmd, cppcheck, SonarQube, checkstyle, ccm, cobertura, Avalanche, Valgrind