Char7. 애플리케이션 테스트 관리

애플리케이션 테스트

애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차

고객의 요구사항을 만족시키는지 **확인validation** 및 **기능**을 정확히 수행하는지 **검증verification**

완벽한 테스트 불가능	애플리케이션 테스트는 소프트웨어의 잠재적인 **결함**을 줄일 수 있지만 소프트웨어에 **결함**이 없다고 증명할 수 없다.
결함 집중 **Defect Clustering**	**애플리케이션의 20%에 해당하는 코드에서 전체 결함의 80%가 발견된다**는 **파레토법칙 Pareto Principle**을 적용하기도 한다.
살충제 패러독스 **Pesticide Paradox**	애플리케이션 테스트에서 **동일한 테스트 케이스로 동일한 테스트를 반복**하면 더 이상 결함이 발견되지 않는 살충제 패러독스 현상이 발생
테스팅은 정황Context 의존	소프트웨어 특징, 테스트 환경, 테스터 역량 등 **정황 Context**에 따라 테스트 결과가 달라질 수 있으므로, 정황에 따라 테스트를 다르게 수행해야 한다.
오류-부재의 궤변 **Absence of Errors Fallacy**	소프트웨어의 결함은 모두 제거해도 **사용자의 요구사항**을 만족시키지 못하면 해당 소프트웨어는 품질이 높다고 할 수 없다.
테스터와 위험은 반비례	
테스터의 점진적 확대	테스트는 작은 부분에서 시작하여 점점 확대하며 진행해야 한다.
테스트의 별도 팀 수행	테스트는 개발자와 관계없는 별도의 팀에서 수행해야 한다.

프로그램의 실행 여부에 따른 테스트

정적 테스트	**프로그램을 실행하지 않고 명세서나 소스 코드 대상 분석하는 테스트** **소프트웨어 개발 초기에 결함을 발견할 수 있어 소프트웨어의 개발 비용을 낮추는데 도움이 된다.**	**워크스루walkthrough** **인스펙션Inspection** **코드 검사**
동적 테스트	**프로그램을 실행하여 오류를 찾는 테스트로 소프트웨어 개발의 모든 단계에서 테스트를 수행할 수 있다.**	**블랙박스 테스트** **화이트박스 테스트**

테스트 기반 Test Bases에 따른 테스트

명세 기반	**사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트**	**동등 분할** **경계 값 분석**
구조 기반	**소프트웨어 내부의 논릴 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트**	**구문 기반** **결정 기반** **조건 기반**
경험 기반	**유사 소프트웨어나 기술 등에 대한 테스터의 경험을 기반으로 수행하는 테스트** **경험 기반 테스트는 사용자의 요구사항에 대한 명세가 불충분하거나 테스트 시간에 제약이 있는 경우 수행하면 효과적**	**에러 추정** **체크 리스트** **탐색적 테스팅**

시각에 따른 테스트

확인 validation 테스트	**사용자의 시각에서 생산된 제품의 결과를 테스트하는 것으로,** 사용자가 요구한대로 제품이 완성됐는지, 제품이 정상적으로 동작하는지를 테스트한다.
검증 verification 테스트	**개발자의 시각에서 제품의 생산 과정을 테스트하는 것으로,** 제품이 명세서대로 완성됐는지를 테스트한다.

목적에 따른 테스트

회복 Recovery 테스트	**시스템에 여러 가지 결함을 주어 실패하도록 한 후 올바르게 복구되는지를 확인하는 테스트**
안전 Security 테스트	**시스템에 설치된 시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지를 확인하는 테스트**
강도 Stress 테스트	**시스템에 과도한 정보량이나 빈도 등을 부과하여 과부하 시에도 소프트웨어가 정상적으로 실행되는지를 확인하는 테스트**
성능 Performance 테스트	**소프트웨어의 실시간 성능이나 전체적인 효율성을 진단하는 테스트로 소프트웨어의 응답시간, 처리량 등을 확인하는 테스트**
구조 structure 테스트	**소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등을 평가하는 테스트**
회귀 Regression 테스트	**소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인하는 테스트**
병행 Parallel 테스트	**변경된 소프트웨어와 기존 소프트웨어에 동일한 데이터를 입력하여 결과를 비교하는 테스트**

화이트박스 테스트 White Box Test

모듈의 원시 코드를 오픈 시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트하여 테스트 케이스를 설계하는 방법

설계 **절차에 초점**을 둔 **구조적 테스트**로 **프로시저 설계의 제어 구조를 사용**하여 **테스트 케이스**를 설계

테스트 과정의 **초기**에 적용

- 종류

기초 경로 검사	대표적인 화이트박스 테스트 기법 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트기법	
제어 구조 검사	**조건 검사 Condition Testing**	프로그램 모듈 내에 있는 논리적 조건을 테스트하는 테스트 케이스 설계 기법
	루프 검사 Loop Testing	프로그램 반복 구조에 초점을 맞춰 실시하는 테스트 케이스 설계 기법
	데이터 흐름 검사 Data Flow Testing	프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 실시하는 테스트 케이스 설계 기법

- 검증 기준

문장 검증 기준 Statement Coverage	소스코드의 모든 구문이 한 번 이상 수행되도록 테스트 케이스 설계
분기 검증 기준 Branch Coverage	소스 코드의 모든 조건문이 한 번 이상 수행되도록 테스트 케이스 설계
조건 검증 기준 Condition Coverage	소스 코드의 모든 조건문에 대해 조건이 True인 경우와 False인 경우가 한 번 이상 수행되도록 테스트 케이스 설계
분기/조건 기준 Branch/Condition	소스코드의 모든 조건문과 각 조건문에 포함된 개별 조건식의 결과가 True인 경우와 False인 경우가 한 번 이상 수행되도록 테스트 케이스 설계

블랙박스 테스트 Black Box Test

소프트웨어가 수행할 기능을 알기 위하여 각 기능이 완전히 작동되는 것을 입증하는 테스트. 기능테스트.

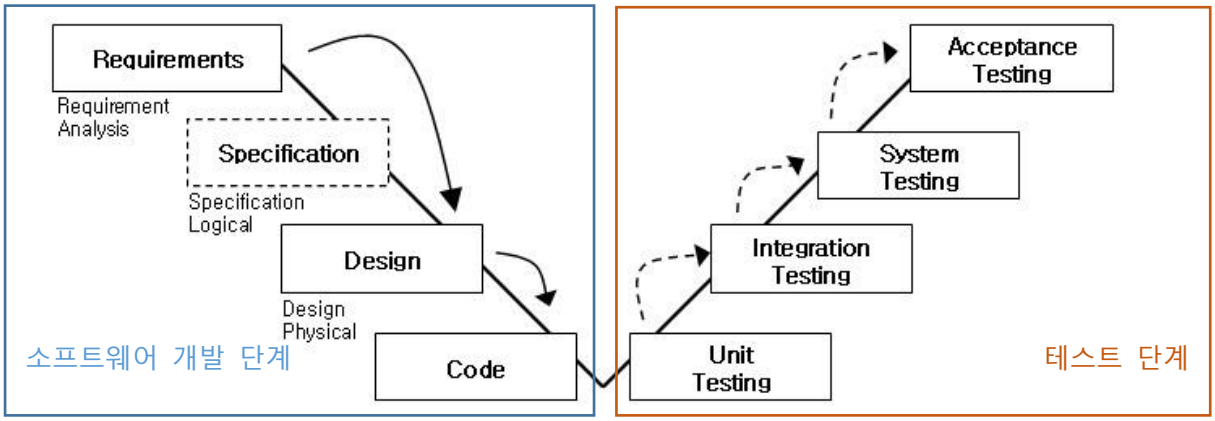
사용자의 요구사항 명세를 보면서 테스트

소프트웨어 인터페이스에서 실시되는 테스트

- 종류

동치 분할 검사 **Equivalence Partitioning Testing**	입력 자료에 초점을 맞춰 테스트 케이스를 만들고 검사하는 방법
경계값 분석 **Boundary Value Analysis**	입력 자료에만 치중한 동치 분할 기법을 보완하기 위한 기법 입력 조건의 중간값보다 경계값에서 오류가 발생될 확률이 높다는 점을 이용하여 입력 조건의 경계값을 테스트 케이스로 선정하여 검사하는 기법
원인-효과그래프검사 **Cause-Effect Graphing Testing**	입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정하여 검사하는 기법
오류 예측 검사 **Error Guessing**	과거의 경험이나 확인자의 감각으로 테스트하는 기법 다른 블랙 박스 테스트 기법에서 찾을 수 없는 오류를 찾아내는 보충적 검사 기법
비교 검사 **Comparison Testing**	여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법

소프트웨어 생명주기 V모델



V - 모델

단위 테스트 Unit Test

소프트웨어 설계의 최소단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트하는 것

사용자 요구사항을 기반으로 한 기능성 테스트를 최우선으로 수행

구조기반 테스트	**주로 사용되는 테스트 방법** 프로그램 내부 구조 및 복잡도를 검증하는 **화이트박스 테스트** 시행	제어흐름 조건 결정
명세기반 테스트	목적 및 실행 코드 기반의 **블랙박스 테스트** 실행	동등 분할 경계값 분석

통합 테스트 Integration Test

단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트

모듈 간 또는 통합된 컴포넌트 간의 상호 작용 오류를 검사

<table>
<tr><td>비점진적
통합 방식</td><td>단계적 통합하는 절차없이 모든 모듈이 미리 결합되어 있는 프로그램 전체를 테스트하는 방식
규모가 작은 소프트웨어
단시간내 테스트 가능
오류 발견 및 장애 위치 파악 어렵</td><td colspan="2">빅뱅 통합 테스트 방식</td></tr>
<tr><td rowspan="3">점진적
통합 방식</td><td rowspan="3">모듈 단위로 단계적 통합하면서 테스트하는 방법
오류 수정이 용이
인터페이스와 연관된 오류를 완전히 테스트할 가능성이 높다.</td><td>하향식
통합 테스트</td><td>프로그램의 상위모듈에서 하위모듈방향으로 통합하면서 테스트하는 기법</td></tr>
<tr><td>상향식
통합 테스트</td><td>프로그램의 하위모듈에서 상위모듈방향으로 통합하면서 테스트하는 기법</td></tr>
<tr><td>혼합식
통합 테스트</td><td>하위수준에서 상향식 통합, 상위수준에서 하향식 통합을 사용하여 최적의 테스트 지원</td></tr>
</table>

하향식 통합 테스트 Top Down Integration Test

깊이 우선 통합법이나 넓이 우선 통합법을 사용

테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있다.

상위모듈에서는 테스트 케이스를 사용하기 어렵다.

- 절차

1. 주요 제어모듈의 종속모듈은 **스텁Stub**으로 대체 상위모듈은 있지만 하위모듈이 없는경우 하위모듈 대체
2. 통합방식에 따라 하위모듈인 스텁들이 한번에 하나씩 실제 모듈로 교체
3. 모듈이 통합될 때마다 테스트 실시
4. 회귀테스트 실시

회귀 테스트 Regression Test

이미 테스트된 프로그램의 테스팅을 반복하는 것으로 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인하는 테스트

상향식 통합 테스트 Bottom Down Integration Test

하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 **클러스터Cluster**가 필요

- 절차

1. 하위모듈들을 클러스터Cluster로 결합
2. 상위모듈에서 데이터 입출력을 확인하기 위해 더미모듈인 **드라이버Driver**로 작성 상위 모듈 없이 하위 모듈이 있는 경우 하위 모듈 구동
3. 통합된 클러스터 단위로 테스트
4. 테스트가 완료되면 클러스터는 프로그램 구조의 상위로 이동하여 결합되고 드라이버는 실제 모듈로 대체
5. 클러스터 검사

- 테스트 케이스 선정 방법

1. 대표적인 테스트 케이스 선정
2. 파급효과가 높은 부분이 포함된 테스트 케이스 선정
3. 실제 수정이 발생한 모듈이나 컴포넌트에서 시행하는 테스트 케이스 선정

시스템 테스트 System Test

개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행되는가를 점검하는 테스트

기능적 요구사항	요구사항 명세서, 비즈니스 절차, 유스케이스 등 명세서 기반의 **블랙박스 테스트** 실행
비기능적 요구사항	성능 테스트, 회복 테스트, 보안 테스트, 내부 시스템의 메뉴 구조, 웹 페이지의 네비게이션 등 구조적 요소에 대한 **화이트박스 테스트** 실행

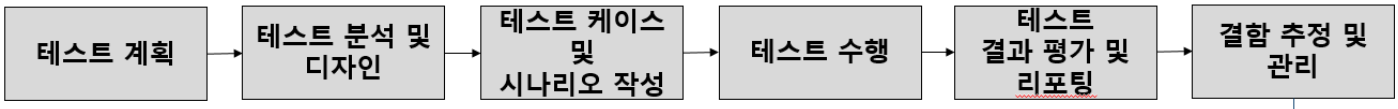
인수 테스트 Acceptance Test

개발된 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두고 테스트하는 방법

사용자가 직접 테스트

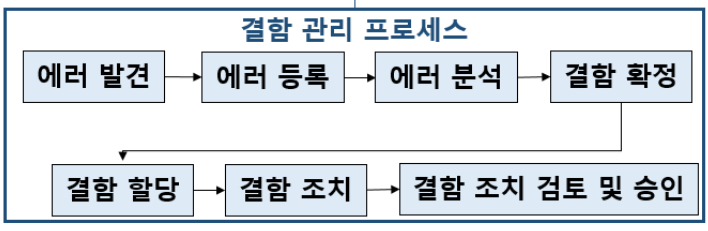
사용자 인수 테스트	사용자가 시스템 사용의 적절성 여부를 확인
운영상 인수 테스트	시스템 관리자가 시스템 인수 시 수행하는 테스트 기법
계약 인수 테스트	계약상의 인수/검수 조건을 준수하는지 여부를 확인
규정 인수 테스트	소프트웨어가 규정에 맞게 개발되었는지를 확인
알파 테스트	**개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 기법**
베타 테스트	**선정된 최종 사용자가 여러 사용자 앞에서 행하는 테스트 기법**. 사용자가 직접 테스트

애플리케이션 테스트 프로세스



애플리케이션 테스트를 마치면 다음 문서들이 산출

- **테스트 계획서**
  테스트 수행을 계획한 문서
- **테스트 케이스**
  사용자 요구사항을 얼마나 준수하는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
- **테스트 시나리오**
  테스트를 수행할 여러 개의 테스트 케이스의 동작 순서를 기술한 문서
- **테스트 결과서**
  테스트 결과를 비교 분석한 내용을 정리한 문서



결함 Fault

소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생되는 것

테스트 오라클 Test Oracle

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

특징1. 제한된 검증: 모든 테스트 케이스에 적용할 수 없다.

특징2. 수학적 기법: 테스트 오라클의 값을 수학적 기법을 이용하여 구할 수 있다.

특징3. 자동화 기능

참 True 오라클	모든 테스트 케이스의 입력값에 대해 기대 결과를 제공하는 오라클
샘플링 Sampling 오라클	특정 몇몇 테스트 케이스의 입력값들에 대해서만 기대 결과를 제공하는 오라클
추정 Heuristic 오라클	샘플링 오라클을 개선한 오라클 특정 테스트 케이스의 입력값에 대해 기대 결과를 제공 나머지 입력 값들에 대해 추정으로 처리하는 오라클
일관성 검사 Consistent 오라클	애플리케이션의 변경이 있을 때 테스트 케이스의 수행 정후의 결과값이 동일한지를 확인하는 오라클

테스트 자동화 도구를 사용함으로써

휴먼 에러 Human Error를 줄이고 **테스트 정확성**을 유지하면서 **테스트의 품질**을 향상 시킬 수 있다.

정적 분석 도구 **Static Analysis Tools**	프로그램을 실행하지 않고 분석하는 도구 테스트를 수행하는 사람이 작성된 소스코드를 이해하고 있어야만 분석이 가능	
테스트 실행 도구 **Test Execution Tools**	스크립트 언어를 사용하여 테스트를 실행하는 방법 데이터 주도 접근 방식과 키보드 주도 접근 방식으로 분류	
성능 테스트 도구 **Performance Test Tools**	애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률 등을 인위적으로 적용한 가상의 사용자를 만들어 테스트를 수행함으로써 성능의 목표 달성 여부를 확인	
테스트 통제 도구 **Test Control Test**	테스트 계획 및 관리, 테스트 수행, 결함 관리 등을 수행하는 도구 종류로는 형상 관리 도구, 결함 추적/ 관리 도구 등이 있다.	
테스트 하네스 도구 **Test Harness Tools**	테스트 하네스는 애플리케이션의 컴포넌트 및 모듈을 테스트하는 환경의 일부분으로, 테스트를 지원하기 위해 생성된 코드와 데이터를 의미	
	테스트 드라이버 **Test Driver**	테스트 대상의 하위 모듈을 호출하고, 파라미터를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 도구
	테스트 스텁 **Test Stub**	제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구, 일시적으로 필요한 조건만을 가지고 있는 테스트용 모듈
	테스트 슈트 **Test Suites**	테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합
	테스트 케이스 **Test Case**	사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
	테스트 스크립트 **Test Script**	자동화된 테스트 실행 절차에 대한 명세서
	목 오브젝트 **Mock Object**	사전에 사용자의 행위를 조건부로 입력해 두면, 그 상황에 맞는 예정된 행위를 수행하는 객체

- 결함은 오류 발생, 작동 실패 등과 같이 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생되는 것을 의미한다.
- **결함 관리 프로세스 처리 순서** : 결함 관리 계획 → 결함 기록 → 결함 검토 → 결함 수정 → 결함 재확인 → 결함 상태 추적 및 모니터링 활동 → 최종 결함 분석 및 보고서 작성
- **결함 관리 측정 지표**

결함 분포	모듈 또는 컴포넌트의 특정 속성에 해당하는 결함 수 측정
결함 추세	테스트 진행 시간에 따른 결함 수의 추이 분석
결함 에이징	특정 결함 상태로 지속되는 시간 측정

- **결함 추적 순서** : 결함 등록(Open) → 결함 검토(Reviewed) → 결함 할당(Assigned) → 결함 수정(Resolved) → 결함 종료(Closed) → 결함 해제(Clarified)

※ Fixed(고정) : 개발자가 필요한 변경 작업을 수행하여 결함 수정 작업을 완료한 상태

애플리케이션 성능 지표

처리량 (Throughput)	일정 시간 내에 애플리케이션이 처리하는 일의 양
응답 시간 (Response Time)	애플리케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간
경과 시간 (Turn Around Time)	애플리케이션에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
자원 사용률 (Resource Usage)	애플리케이션이 의뢰한 작업을 처리하는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등 자원 사용률

결함 관리 도구

Mantis	결함 및 이슈 관리 도구로, 소프트웨어 설계 시 단위별 작업 내용을 기록할 수 있어 결함 추적도 가능한 도구
Trac	결함 추적은 물론 결함을 통합하여 관리할 수 있는 도구
Redmine	프로젝트 관리 및 결함 추적이 가능한 도구
Bugzilla	결함을 지속적으로 관리할 수 있는 도구로 결함의 심각도와 우선순위를 지정할 수 있다.

소스코드 최적화

- 나쁜 코드(Bad Code)를 배제하고, 클린 코드(Clean Code)로 작성하는 것이다.
- **클린 코드(Clean Code)** : 누구나 쉽게 이해하고 수정 및 추가할 수 있는 단순, 명료한 코드
- **나쁜 코드(Bad Code)** : 코드의 로직이 서로 얽혀 있는 스파게티 코드 등 프로그램의 로직(Logic)이 복잡하고 이해하기 어려운 코드
- 나쁜 코드로 작성된 애플리케이션의 코드를 클린 코드로 수정하면 애플리케이션의 성능이 개선된다.

- **클린 코드 작성 원칙** : 가독성, 단순성, 의존성 배제, 중복성 최소화, 추상화
- **소스 코드 최적화 유형**
  - 클래스 분할 배치 : 하나의 클래스는 하나의 역할만 수행하도록 응집도를 높이고, 크기를 작게 작성함
  - Loosely Coupled(느슨한 결합) : 인터페이스 클래스를 이용하여 추상화된 자료 구조와 메소드를 구현함으로써 클래스 간의 의존성을 최소화함
  - 코딩 형식 준수, 좋은 이름 사용, 적절한 주석문 사용