77일차 강의(소프트웨어 테스트 설계)

[ 소프트웨어 테스트 설계 ]

: 요구사항 분석 -> 테스트 계획 수립 -> 테스트 케이스 작성 -> 테스트 수행 -> 결과 보고

(테스트 전략 수립 진행)

가장 중요한 것, 적은 테스트 케이스로 최대한 많은 결함을 찾아야 함

* 테스트 케이스?

: 특정 프로그램 경로를 실행하거나, 요구사항대로 동작하는지 검증하기 위해 개발된 테스트 대상에 대한 입력 값으로 사전 조건(pre-condition), 우선순위(priority), 재현 경로, 예상 결과, 사후 조건(post\_condition) 등에 대한 명세

Positive Test Negative Test

의도된 대로 동작함을 검증 의도하지 않은 동작에 대한 검증

* 테스트 케이스 작성시 필요한 항목들?

1. 식별번호(ID)

: 테스트 케이스를 구별하기 위한 식별 번호

1. 사전조건

: 테스트 수행 전에 갖춰져야 할 조건, 구동 환경, 테스트 데이터 등

(테스트 데이터 : 가작 작은 단위로 데스트 데이터가 있어야

로그인 같은 경우)

(구동환경은 mock?? 이라는 다른 툴이 따로 필요함 – 일단 생생각하 말 것)

1. 테스트 수행 절차

: 구체적인 테스트 수행 단계, Action

1. 기대 결과

: 정상적으로 동작 시 실행 결과

1. 중요도

: 테스트 케이스의 중요한 정도,

시간적 제약이 있을 경우 우선순위 기준척도로 사용

1. 합격 / 불합격

: 수행 결과에 대해 확인된 최종 결과

1. 비고

: 테스트 케이스의 의도 등 관련 내용 기술

( 이 메서드가 하는 역할이 뭔지 주석으로 적어주는 것)

* 여러 기법 존재
  1. 명세 기반 테스트 기법
     + 가장 많이 사용되는 기법
     + 블랙박스 테스트라고도 불림
* 테스트 설계 기법을 통해 테스트 케이스 작성시 이점

1. 결함 발견 확률 증가
2. 자원의 효율적 활용
3. 테스트 케이스 재사용 용이
4. 프로젝트 일정 준수 가능
5. 높은 품질의 소프트웨어 제품 배포 가능
6. 요구사항 오류 발견 용이

* 명세 기반 기법의 특징
  + - 블랙박스 또는 동작 테스트라고도 함
    - 테스터(우리들)는 소프트웨어의 내부 시스템에 관심 없음
    - 주어진 입력 값(input)에 대한 추력(output)에 주목
    - 소프트웨어 개발 주기(CRUD)의 모든 단계에서 테스트 용이
    - 요구사항 또는 기능 명세서가 마련되어 있어야 함
* 구조 기반 기법의 특징
* 화이트 박스 테스트라고도 함
* 테스터는 시스템 내부의 구현 방식에 관심 있음
* 소프트웨어가 어떻게 동작하는지 주목
* **모든 레벨**의 테스트 단계에서 테스트 가능
* 블랙박스 테스트란?
  + - 시스템이 무엇을 수행하는지 테스트 (What)

(블랙박스 내부는 중요한 것이 아니다)

* + - 시스템의 결과와 동작에서 버그 식별
* 소프트웨어의 내부 구조를 직접 참조하지 않고 요구사항 명세서나 설계 문서를 기반으로 소프트웨어의 기능적 혹은 비기능적 요구 및 지시사항이 제대로 동작하는지 검증하는 방법
* 테스트 케이스를 체계적으로 도출하여 테스트
* 문서 기반

* 블랙박스 테스트 종류

1. 등가 분할
2. 경계 값 분석
3. 결정 테이블 테스트
4. 분류 트리 기법
5. 상태 전이 테스트
6. 유즈 케이스 테스트

* 화이트 박스 테스트란?
  + - 시스템이 어떻게 동작하는 테스트(How)

(까다롭고 어렵다, 광범위 하여 레벨이 높음)

* + - 기능 또는 인터페이스 상의 내보 버그 식별
* 컴포넌트(Component) 혹은 코드(Code)와 같은 내부 구조 분석에 바탕을 두고 테스트 케이스를 도출하여 테스트하는 방법
* 구조 시험이라고도 함
* 프로그램 내의 경로를 수행하여 잠재적인 오류를 찾아냄

컴포넌트 : 모듈의 집합체

* 화이트 박스 테스트 종류

1. 구문 커버리지
2. 결정 커버리지
3. 결정/조건 커버리지
4. 복수 조건 커버리지
5. 블랙박스 테스트 VS 2. 화이트박스 테스트

1\_1. 요구사항 명세서에 명시되어 있지 않은 기능

* + - 테스트 불가능

1\_2. 요구사항 명세서가 불완전한 경우,

블랙박스 테스트로 모든 기능 검증 불가능

2\_1. 너무 많은 수의 경로 고려

2\_2. 모든 경로를 실행하고도 오류를 발견하지 못하는 경우 발생

2\_3. 경로가 존재하는 경우에만 테스트 가능

* 구문 커버리지
* 테스트 케이스에 의해 프로그램 내의 모든 구문을 한번 이상 실행시키는 것을 기준으로 하는 커버리지
* Junit 을 테스트하기 위한 규칙!!!
  + - 리턴타입은 무조건 없어야 -> void 여야!!
    - 클래스명 끝에 Test를 붙인다
    - 메서드명 앞에 test를 붙인다
    - 매개변수는 무조건 없다
    - @Test 어노테이션 적고, import 해준다
    - 기본적인 것만 구현하고 실패가 뜨는 것을 확인하고 시작하는 것이다

**[ 최소로 증명해야 할 범위는 ? ]**

* 입력 조건이 범위를 나타낼 때
  1. 입력 조건이 특정한 값일 때 -> 1개의 valid / 2개의 invalid
  2. 입력 조건이 집합일 때 -> 1개의 valid / 1개의 invalid
  3. 입력 값이 Boolean 값일 때 -> 1개의 valid / 1개의 invalid
* 범위의 값: 1~100

Valid EC : Invalid EC

1 <= 값 <= 100 : 값 < 1 , 값 > 100

특정한 값: 10

Valid EC : Invalid EC

값 = 10 : 값 < 10 , 값 > 10

* 집합 : { c. o , u, n, t, e, r}

Valid EC Invalid EC

c. o , u, n, t, e, r 이것 중에 하나 : c. o , u, n, t, e, r 이외의 값

-> EqualArray 사용하면 된다)

* Boolean 값 : 사람의 성별

Valid EC : Invalid EC

남, 여 : 남, 여 이외

**[ 경계 값 분석 ]**

입력 조건의 중간 값 보다는 경계 값에 오류가 발생할 확률이 경험적으로 높다는 점을 감안하여 경계 값을 포함하여 테스트 케이스를 설계하는 방법

* 일반적으로 개발자가 비교 연산자의 사용에서 실수할 여지가 많음.

Ex) 누군가 입력한 값 N 이라면,

1~100까지라면 무엇을 테스트 해야하나?

(아래 6개정도 테스트 해야한다)

* N이 1일경우 : N, N-1, N+1 정도 테스트해야

: 1, 0, 2

* N이 100일 경우 : N, N-1, N+1 정도 테스트해야

: 100, 99, 101

* **JUnit 은 오로지 자바를 테스트하기 위한 것이다!!!!**
* 단위 테스트지 경로 테스트가 아니다? (뭔말?

(서비스단에서 테스트를 하면 서비스단에서만 테스트해야지 다오나 데이터베이스까지 가서 테스트하지 않는다.

하나의 테스트는 하나만 한다.

서비스단 테스트는 서비스만 되면된다 – 가작 작은 단위로 (이것이 단위 테스트)

* 이슈 관리

이슈 상태 변화

* 1. New

: 테스터나 개발자에 의해서 발견된 이슈가 처음 등록된 상태

* 1. Assign

: 해당 이슈에 대해서 담당자가 할당 되었을 때의 상태

* 1. Open

: 담당자에 의해서 해당 이슈가 진성 이슈가 맞으면 이슈를 수정할 예정인 상태

* 1. Fixed

: 담당자에 의해서 해당 이슈가 수정된 상태

* 1. Closed

: 처음 이슈를 발견한 사람이나 테스터에 의해서 해당 이슈가 수정된 것을 확인한 상태

* 1. Reopen

: Fixed 상태의 이슈를 테스트했을 때 이슈가 수정되지 않았거나 사이드 이펙트가 발견되어 다시 수정이 필요한 상태

* CRUD 를 만들었다면 Junit 에서 무엇을 테스트 해야할까?
* Create
* Read
* Update
* Delete
* 셀렉리스트 일 경우?

범위

List.size

숫자

(리스트가 하나 이상이다

숫자>=0 - 셀렉으로하면 하나 이상 들고 왔다

숫자 < -1 - 셀렉으로하면 하나도 들고오지 않았다

(객체 비교를 하고싶다면 객체를 하나 들고와서 get(adf) -> assertNotNull 한다)

* 하나짜리? selecOne( )

하나 VO

Ex) 1

홍길동

* Insert(vo)

Update

Delete

* Void 타입으로 생각해서, 확인할수 있는 (후행확인을 통해서)
* Select dfadf

Adfaf

Where no = 1;

이런식으로 의미있게 후행확인을 할 수 있는 무언가가 필요하다