TDD(Test Driven Development)

Test

좋은 테스트란?

- 읽기 쉬운 테스트 코드 : 소스 코드와 같은 수준의 가독성 유지

- 구조를 잘 갖춘 테스트 : 통짜 클래스가 아닌 적정한 수준의 구성

- 엉뚱한 걸 검사하지 말자 : 테스트 메소드명은 중요

- 테스트가 얼마나 독립적인가? : 방금 개봉한 새 pc에 버전 관리 서버에서 내려받은 테스트 코드를 바로 실행

- 믿고 쓰는 테스트 및 결과 : 경계값으로 만들어진 테스트들, 하지만 assert가 없는 테스트라면?

JUnit

- 테스트 자동화 프레임워크

- 테스트 픽스처(Test fixture)

- 테스트 케이스와 테스트 메소드

- '테스트'자체가 중요시 되면서 편의성과 자동화된 테스트 환경을 제공해주기 위한 프레임워크

- Junit은 Reflection을 사용

JUnit을 사용한 클래스 기본구조

- @Before : 테스트에 필요한 변수나 환경설정

- @Test : 실제 테스트가 진행됨

Given : 테스트와 관련된 조건식

When : 테스트 실행(행위, behavior)

Then : 테스트 실행결과 확인

- @After : 테스트에 사용된 뒷정리

Mock

- 실제 객체를 만들기엔 비용과 시간이 많이들거나 의존성이 길게 걸쳐져 있어 제대로 구현하기 어려울 경우, 이런 가짜 객체를 만들어 사용한다.

Test 간단 실습

TDD

Fact Check

Fact1. 소프트웨어의 소스는 복잡하게 연결되어있다. 작은 부분이 수정된다면? 모든 기능을 다시 테스트 해야한다. - 회귀테스트

Fact2. 소스코드는 시간이 흐르면 조금씩 망가진다.. 끊임없이 OOD와 패턴 구문들을 적용해 가며 리팩토링 해야한다.

Coding vs Design

- 우리가 코딩을 하고 있다고 생각한다면, 우리는 코딩을 하기 위해 생각할 것이고, 코딩을 위한 지식과 스킬을 쌓을 것이다.

- 우리가 설계를 하고 있다고 생각한다면, 우리는 설계를 하기 위해 생각할 것이고, 설계를 위한 지식과 스킬을 쌓을 것이다.

안 좋은 디자인의 징후

- 단위 테스트 케이스 작성이 어렵다.
- 단위 테스트 케이스가 자주 깨진다.
- 단위 테스트 케이스 실행을 위한 준비해야 할 것이 많다.
- 다른 사람의 테스트 케이스를 읽기가 어렵다.

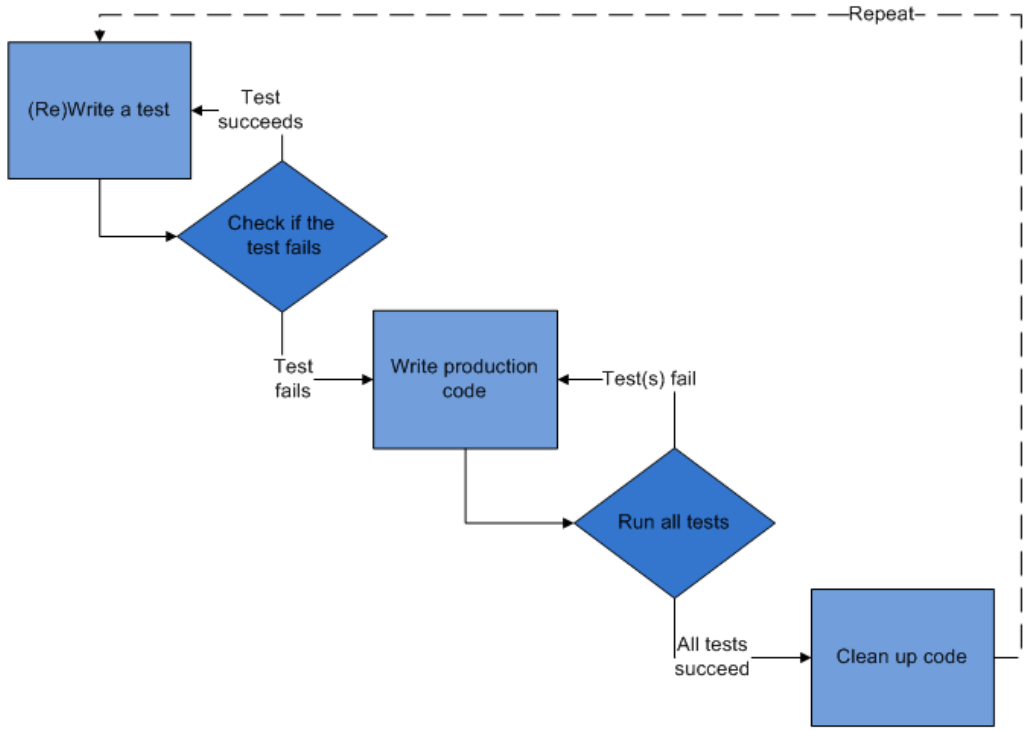
Agile, Agile방법론 그리고 TDD 왜 뭉쳐 다닐까?

- TDD는 Agile원칙을 지키는 방법론 중의 하나인 XP가 제시하는 실천방법 중 하나

TDD가 어려운 이유?

- TDD는 설계 방법이다.
- TDD를 잘하려면 설계를 생각해야하고, 설계를 생각하려면 설계 전문용어를 알아야한다.

TDD 란 무엇인가?


Repeat
(Re)Write a test
Test succeeds
Check if the test fails
Test fails
Write production code
Test(s) fail
Run all tests
All tests succeed
Clean up code

TDD의 장점

1. 지금 당장 구현해야하는 목표를 뚜렷하게 보여주고,

2. 반복되는 짧은 개발 패턴(테스트 실패 -> 테스트를 성공 시키는 최소한의 구현 -> 리팩토링)을 통한 개발 리듬 ♬ 을 만들어 주고,

3. 해당 인터페이스의 훌륭한 API 문서로서 원활한 소통을 가능하게 해주고(물론 잘 만들어진 테스트에 한해서),

4. 문제점에 대한 빠른 개선을 가능하게 해주며

5. 빠른 개발 패턴 덕에 개발자의 성취감을 크게 고취시켜 준다.

그외에도?

- 높은 소스코드 품질(MS와 IBM사의 조사 결과 TDD약 15~35% 정도의 개발시간 증가 버그는 약 40-90% 정도 줄어듬)
- 재설계 시간의 절감
- 손쉬운 테스트 근거 산출 및 문서화(test coverage, performance)
- 디버깅 시간의 절감

- 정확한 사용 시나리오가 포함된 자동화된 실행 가능한 코드
- 군더더기 없는 제품 코드

TDD에 대한 오해

1. TDD는 비용이 더 들고, 결국 개발 속도를 저하시킨다.

    - TDD는 개발하고자 하는 코드에 명백한 가이드를 제시해준다.

    - 빠른 피드백 & 즉각적인 테스트

2. 코드 커버리지가 높으면 좋은 코드이다.

    - TDD로 개발하다 보면 코드가 원하는 방향으로 잘 동작하는지에 대한 클린 테스트를 진행하게 된다. 이런 클린 테스트만을 가지고 모든 조건, 구문, 결과에 대해서 완벽하게 커버되는 테스트를 만들기는 힘들다.

TDD에 대한 진실

1. TDD != Unit Testing

- TDD 는 테스트 코드를 이용하여 짧은 개발 주기를 반복하는 소프트웨어 개발 방법론이다.

- 테스트 코드를 통해 내가 구현하고자 하는 목적 코드를 확실하게 하고, 목적코드를 구현하기 위한 최소한의 구현(테스트 통과를 위한 최소한의 구현)만 진행하라는 것이 TDD 에서 말하는 테스트 코드의 가치라고 말할 수 있다. 테스트 그 자체에 목적을 두고 있지 않다.

2. TDD는 설계 개선에 도움을 준다.

- 우리는 특정 소프트웨어를 개발하기 전에 해당 소프트웨어에 대한 큰 그림을 그리고 시작한다. 이것을 보통 설계 라고 부르는데, 이런 설계 이후에 개발을 하게 될 때 빠른 피드백을 받으면서, 리팩토링해 나가게 되는데 이 때 TDD가 많은 도움이 된다.

TDD의 최종목적

- '잘 동작하는 깔끔한 코드' 작성

TDD를 잘하려면?

- IDE를 잘 사용해야 한다.
- 많이 연습해야한다.
- 테스트하기 편한 환경을 조성한다.
- 잘 쪼개야 한다.
- 더 이상 동작하지 않는 테스트 케이스는 제거한다.
- TDD는 자동화된 테스트를 만드는 것이 최종 목표가 아니다.
- 모든 상황에 대한 테스트 케이스를 만들 필요는 없다.
- 여러 개의 실패하는 테스트 케이스를 한번에 만들지 않는다.

TDD with Spring

- 스프링 프레임워크의 Unit Test 지원
- 의존관계 주입을 통한 객체 생성
- 웹 컨테이너 없는 웹 애플리케이션 테스트
- 단위 테스트 지원 유틸리티
- Injection과 Mock

BDD

동작 지향 개발(behavior driven development; 이하 BDD)는 프로그램 개발 방법의 일종으로, 테스트 주도 개발(test driven development; 이하 TDD)에서 파생되었다.

BDD는 테스트 케이스를 먼저 작성하고 실제 동작 코드를 나중에 작성하는 TDD에서 한발 더 나아가 테스트 케이스 자체가 요구사양이 되도록 하는 개발 방식이다. 테스트케이스가 사양과 일체화 됨으로서 , 사양작성 -> 코딩 -> 테스트 라는 일련의 흐름이 테스트케이스작성 작업을 중심으로 자연스럽게 구현된다.

TDD 실습