@daebalprime

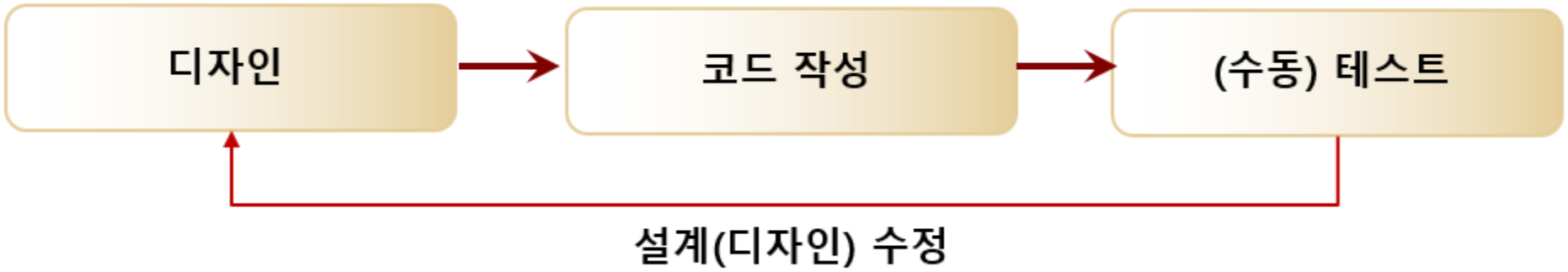
Test Driven Development(TDD) BASIC

목차

1. TDD 개념
2. TDD의 흐름
3. 언제 TDD를 적용하는가?
4. TDD의 장점
5. TDD의 단점
6. 예제

TDD 개념

- 기존의 개발은 디자인 -> 코드 작성 -> 수동 테스트 -> 문제가 있다면 디자인 수정
- 디자인이 바뀌면 코드의 많은 부분이 바뀌어야 한다.

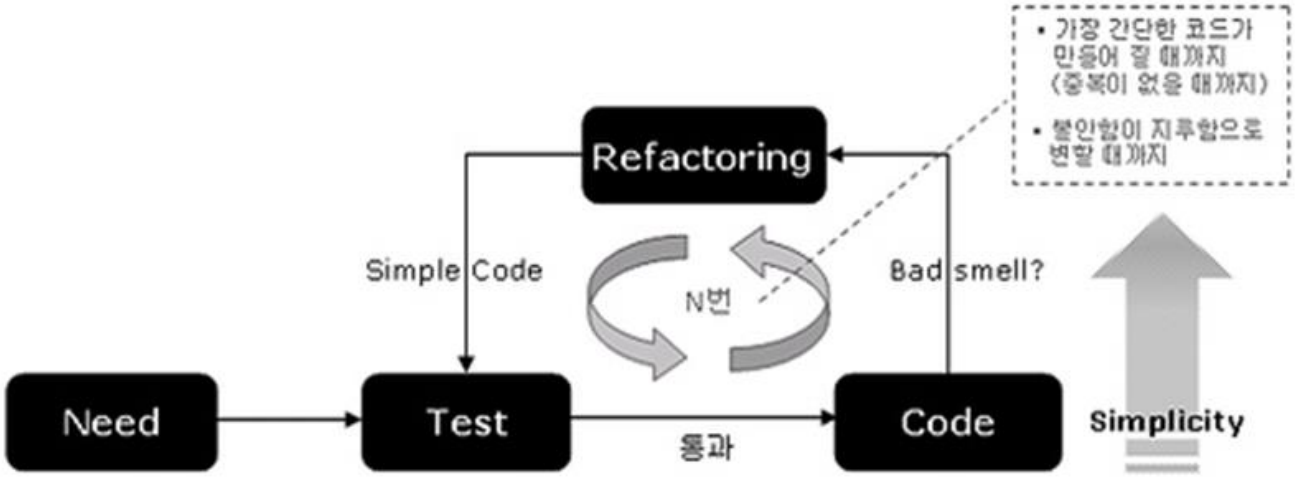


TDD 개념

- TDD는 디자인이 끝나면 개발에 먼저 돌입하는 대신, 기능의 이해를 기반으로 테스트를 먼저 작성하고 테스트를 통과하는 코드 작성을 목적으로 한다.
- 테스트를 수행하고 결과(성공/실패) 피드백을 받고 코드를 수정하는 일련의 과정을 거친다.
- 결정(구현 방법)과 피드백(성공/실패) 사이의 갭을 인식하고 조절하는 테크닉

TDD 흐름

1. 무엇을 테스트할 것인가 생각한다.
2. 존재하지도 않는 메소드를 호출하는 테스트 코드를 먼저 만든다.
3. 실패하는 테스트를 작성한다.
4. 테스트를 통과하는 코드를 작성한다.
5. 테스트를 통과하는 코드 작성을 하였으면 리팩토링한다. (테스트 코드 또한 리팩토링 한다)
6. 위의 작업을 반복한다.



TDD 예제

https://wikidocs.net/224

TDD는 언제 적용해야 할까?

- 처음해보는 프로그램 주제
- 나에 대한 불확실성이 높은 경우
- 고객의 요구조건이 바뀔 수 있는 프로젝트
- 외부적인 불확실성이 높은 경우
- 개발하는 중에 코드를 많이 바꿔야 된다고 생각하는 경우
- 내가 개발하고 나서 이 코드를 누가 유지보수할지 모르는 경우

=> 불확실 할 때

TDD의 장점

유지보수(리팩토링)의 용이성

기본적으로 단위 테스트 기반의 테스트 코드를 작성하기 때문에 추후 문제가 발생하였을 때 각각의 모듈별로 테스트를 진행해보면 문제의 지점을 쉽게 찾을 수 있다.

TDD의 장점

테스트 문서의 대체 가능성

- 대부분의 개발 프로젝트에서 테스트를 진행하는 경우 단순 통합 테스트에 지나지 않는다. TDD를 하게 될 때 테스팅을 자동화시킴과 동시에 더욱 정확한 테스트 근거를 산출해 정의서를 작성할 수 있다.
- 테스트 코드를 읽어봄으로써 개발자의 의사결정과 고민이 나와있음 → 협력 증진
- 쌓이는 테스트들이 추후 환경 변화에서도 어떤 부분이 문제가 되는지 도움이 된다.
- 실행되는 테스트 함수명으로 그 목적을 명확히 알 수 있어야 합니다.

TDD의 장점

설계 수정시간의 단축

- 테스트코드를 먼저 작성하기 때문에 최초 설계안을 만족하게 하며 입출력 구조와 기능의 정의를 명확하게 하게 되므로 설계의 구조적 문제를 바로 찾아낼 수 있다

TDD의 단점

사전준비 기간(학습 기간)

- TDD를 프로젝트에 도입하려면 사전에 필요한 지식을 습득하고 개발 환경을 구축해야 한다. TDD를 효과적으로 사용할 수 있는 수준으로 개발자를 교육하는 데 보통 1~6개월간의 시간이 필요하다.

TDD의 단점

생산성 저하

- TDD를 이용해 테스트 코드를 작성하는 비용이 클 수도 있다.
- MVC 패턴에는 적용이 불가능하다. (Context 등에 영향을 받기 때문)
- TDD를 적용하는 비용과 테스트 가능한 코드 여부, 난이도 등을 파악하여 적용하여야 한다.

Reference

https://wikidocs.net/224
https://velog.io/@kameals/TDD-slow-start
https://jojoldu.tistory.com/306 (심화)
https://gmlwjd9405.github.io/2018/06/03/agile-tdd.html
https://mangkyu.tistory.com/141?category=761303 (에제)