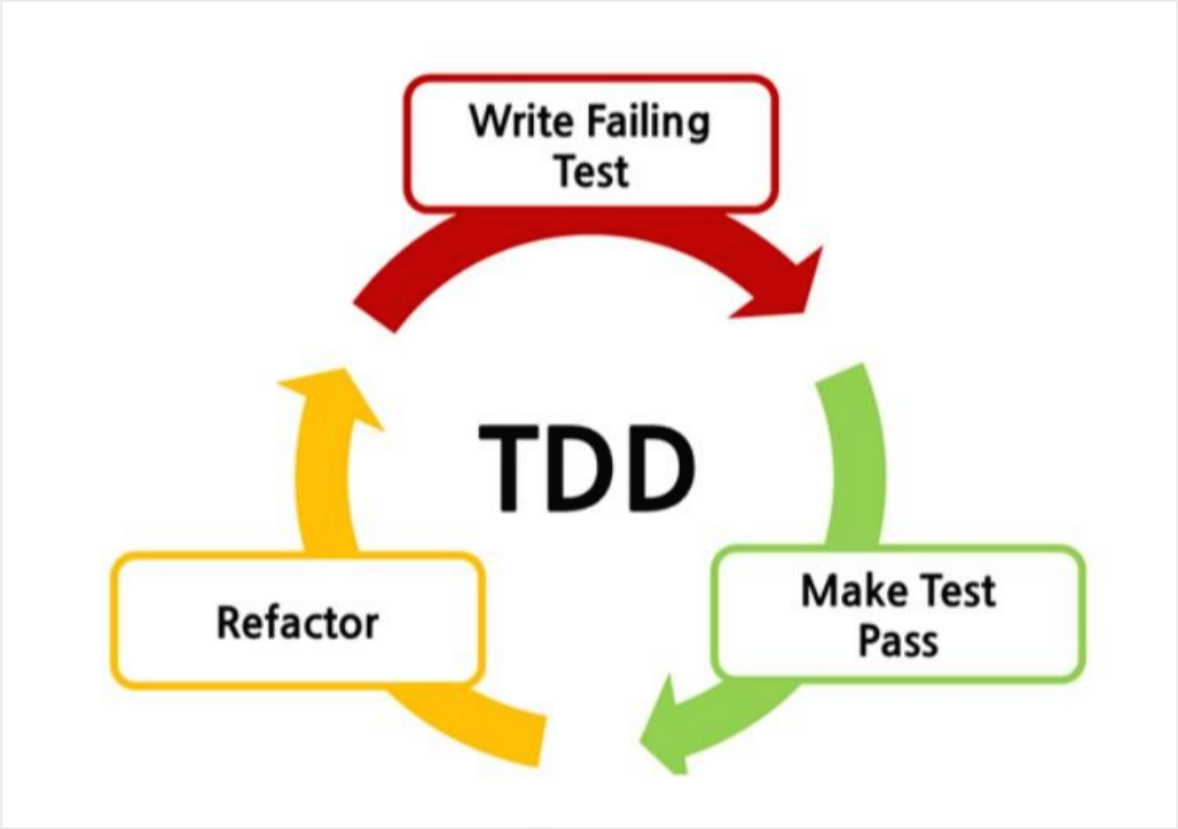
TDD란 무엇인가?

❓ TDD?

TDD(Test Driven Development)는 테스트 주도 개발입니다. 반복 테스트를 이용한 소프트웨어 방법론으로, 작은 단위에 테스트 케이스를 작성하여 이를 통과하는 코드를 추가하는 단계를 반복하여 구현합니다.



❓ TDD 개발주기

[Write Failing Test] 는 실패하는 테스트 코드를 작성합니다
[Make Test Pass]는 테스트를 성공하기 위한 실제 코드를 작성합니다
[Refactor]는 중복 코드를 제거, 일반화 등의 리팩토링을 수행합니다

중요한 것은 실패하는 코드를 작성할 때까지 실제 코드를 작성하지 않는 것과, 실패하는 테스트를 통과할 정도의 최소 실제 코드를 작성해야 합니다. 이를 통하여 실제 코드에 대해 기대되는 바를 보다 명확하게 정의함으로써 불필요한 설계를 피할 수 있고, 정확한 요구 사항에 집중할 수 있습니다.

일반 개발방식과 TDD개발 방식의 차이

일반 개발방식

일반 개발 방식은 [요구사항 분석] -> [설계] -> [테스트] -> [배포] 의 형태의 개발 주기를 갖습니다. 하지만 이러한 방식은 개발을 느리게 하는 잠재적 위험이 존재합니다.

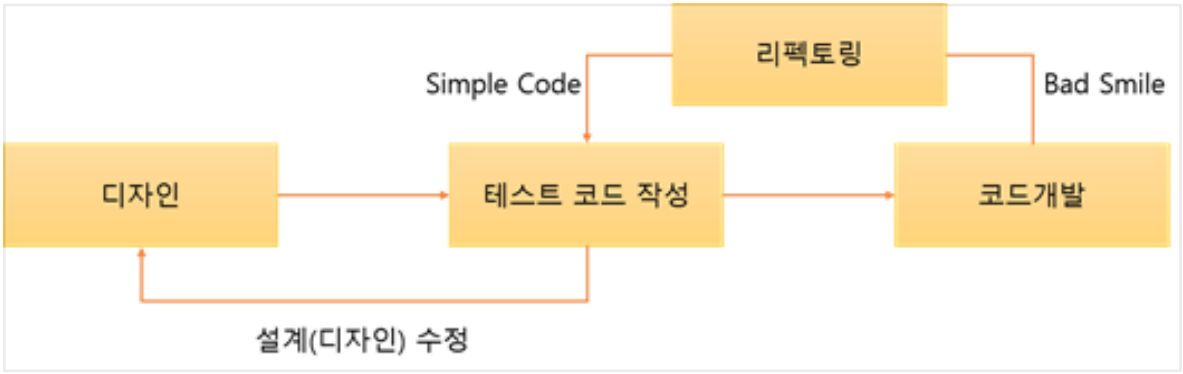
1. 소비자의 요구 사항이 처음부터 명확하지 않을 수 있다.
2. 처음부터 완벽한 설계는 어렵다
3. 자체 버그 검출 능력 저하 또는 소스코드의 품질이 저하될 수 있다.
4. 자체 테스트 비용이 증가할 수 있다.

위의 4 가지의 문제가 발생하는 이유는 어떤 프로젝트든 초기 설계가 완벽하다가 말할 수 없기 때문입니다. 고객의 요구사항 또는 디자인 오류 등등 많은 내외부적인 요인에 의해 재설계하면서 완벽한 설계로 나아갑니다. 재설계로 인해 개발자는 코드를 삽입, 수정, 삭제 하는 과정에서 불필요한 코드가 남거나 중복처리 될 가능성이 큽니다.

결론은, 이러한 코드들은 재사용성이 없고 유지보수가 어려워집니다.

작은 부분의 기능을 수정한다고 하여도 모든 부분을 테스트하게 되면 분명 전체적인 버그를 검출하기가 어려워집니다. 작은 수정에도 모든 기능을 다시 테스트해야 하는 문제가 발새앟여 자체 테스트 비용이 증가하게 됩니다.

TDD 개발 방식



일반 개발방식과 TDD와의 가장 큰 차이는 테스트 코드를 작성한 뒤에 실제 코드를 작성한다는 점입니다.
설계 단계에서 프로그래밍의 목적을 반드시 미리 정의해야만 하고, 또 무엇을 테스트해야 할지 미리 정의해야만 합니다.
테스트 코드를 작성하는 도중 발생하는 예외사항들은 모두 테스트 케이스에 추가하고 설계를 개선합니다. 이후 테스트가 통과된 코드만을 코드 개발 단계에서 실제 코드로 작성합니다.

이러한 중간중간 반복적으로 테스트를 거치면서 자연스럽게 코드의 버그가 줄어들게 되고, 소스코드는 간결해집니다.

또한, 테스트 케이스 작성으로 인해 자연스럽게 설계가 개선됨으로 재설계 시간이 절감됩니다.

TDD의 대표적인 Tool

❓ JUnit

JUnit은 현재 전 세계적으로 가장 널리 사용되고 있는 "Java 단위 테스트 프레임워크" 입니다.

TDD 개발 방식의 장점

- 보다 튼튼한 객체 지향적인 코드 생산

TDD는 코드의 재사용 보장을 명시하므로 TDD를 통한 소프트웨어 개발 시 기능 별 철저한 모듈화가 이뤄집니다. 이는 종속성과 의존성이 낮은 모듈로 조합된 소프트웨어 개발을 가능하게 하며 필요에 따라 모듈을 추가하거나 제거해도 소프트웨어 전체 구조에 영향을 미치지 않게 됩니다.

- 재설계 시간의 단축

테스트 코드를 먼저 작성하기 때문에 개발자가 지금 무엇을 해야하는지 명확하게 정의하고 개발을 시작하게 됩니다. 또한 테스트 시나리오를 작성하면서 다양한 예외사항도 찾고 생각해볼 수 있습니다. 이는 후에 일어날 예외사항에 대처가 가능합니다.

- 디버깅 시간의 단축

유닛 테스트를 하는 이점이기도 합니다. 예를들면 사용자의 데이터가 잘못 나왔다면 DB의 문제인지. 비즈니스 레이어의 문제인지 UI의 문제인지 실제 모든 레이러들을 모두 디버깅 해야하지만, TDD의 경우 자동화된 유닛테스팅을 전재하므로 특정 버그를 손 쉽게 찾을 수 있습니다.

- 테스트 문서의 대체 가능

주로 SI프로젝트 진행 과정에서 어떤 요소들이 테스트 되었는지 테스트 정의서를 만듭니다. 이것은 단순한 통합 테스트 문서에 지나지 않지만, TDD를 하게 될 경우 테스팅을 자동화 시킴과 동시에 보다 정확한 테스트 근거를 산출할 수 있습니다.

- 추가 구현의 용이함

개발이 완료된 소프트웨어에 어떤 기능을 추가할 때 가장 우려 되는 점은 해당

기능이 기존 코드에 어떤 영향을 미칠 지 알지 못한다는 것입니다. 하지만 TDD의 경우 자동화된 유닛 테스팅을 전재하므로 테스트 기간을 획기적으로 단축시 킬 수 있습니다.

하지만 이렇게 좋은 TDD가 있는데 왜 처음부터 TDD를 배우지 않는 것일까요? 단점이 존재하기 때문입니다.

TDD 개발 방식의 단점

- 생산성 저하

개발 속도가 느려진다고 생각하는 사람이 많기 때문에 TDD에 대해 반신반의 합니다. 왜냐하면 처음부터 2개의 코드를 짜야하고, 중간중간 테스트를 하면서 고쳐나가야 하기 때문입니다.
TDD 방식의 개발은 일반적인 개발 방식에 비해 대략 10~30% 늘어나게 됩니다. 또한 SI 프로젝트에서는 소프트웨어의 품질보다 납기일 준수가 훨씬 중요하기 때문에 TDD 방식을 선호하지 않습니다.

TDD를 하기 어려운 이유?

- 이제까지 자신이 개발하던 방식을 많이 변경해야함

예를 들어, 지금까지 오른손으로 기타를 치던 사람에게 왼손으로 치라고 하면 분명 어색해 하고 어려워 할 것입니다. 이와 같이 몸에 체득한 것이 많을 수록 변경하기가 어렵습니다. 오히려 처음 배우는 사람들이 적용하기가 쉽습니다.

- TDD는 이렇게 해야된다는 이미지/틀이 있다

참고 블로그
(TDD로 코드 짜는법)
https://frozenpond.tistory.com/149

(참고한 블로그)
https://wooaoe.tistory.com/33