CS study 15주차

TDD 방법론

TDD란?

Test Driven Development의 약자

'테스트 주도 개발'

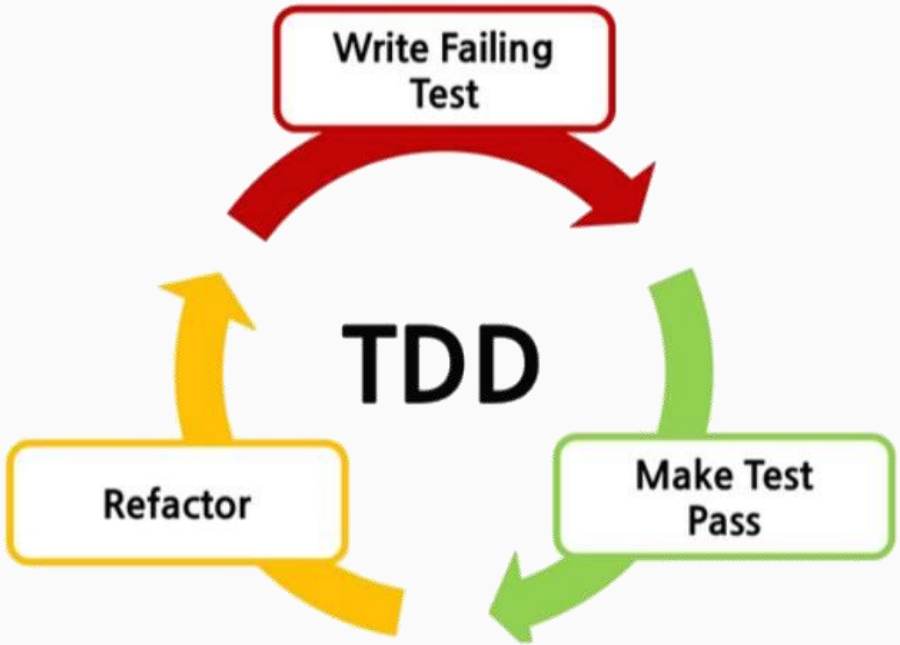
TDD란?

반복 테스트를 이용한 소프트웨어 방법론

작은 단위의 테스트 케이스를 작성하고,

이를 통과하는 코드를 추가하는 단계를 반복하여 구현함.

TDD의 개발 주기



1. 실패하는 테스트 코드 작성
2. 테스트 코드를 성공시키기 위한 실제 코드 작성
3. 중복 코드 제거, 일반화 등의 리팩토링 진행

TDD에서 중요한 점

개발자는 기능의 요구사항과 명세를 분명히 이해하고 있어야 한다.

테스트 코드를 작성하는 이유?

새로운 기능을 추가할 때, 기존에 잘 작동하는 기능이 제대로 작동하지 않는 경우를 방지할 수 있다.

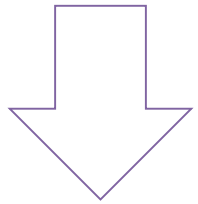
리팩토링을 안심하고 할 수 있다.

깔끔한 코드 작성이 가능하다.

빠른 시간 내에 코드의 동작 방식과 결과를 확인할 수 있다.

테스트 코드를 작성하는 이유?

1. 코드 수정
2. 서버를 동작 시키고 필요에 따라 테스트에 필요한 데이터를 DB에 저장
3. 브라우저를 통해 서버에 접속해 해당 기능을 동작시키는 요청을 보냄
4. 테스트를 마치고, DB의 데이터를 원래대로 돌려놓음



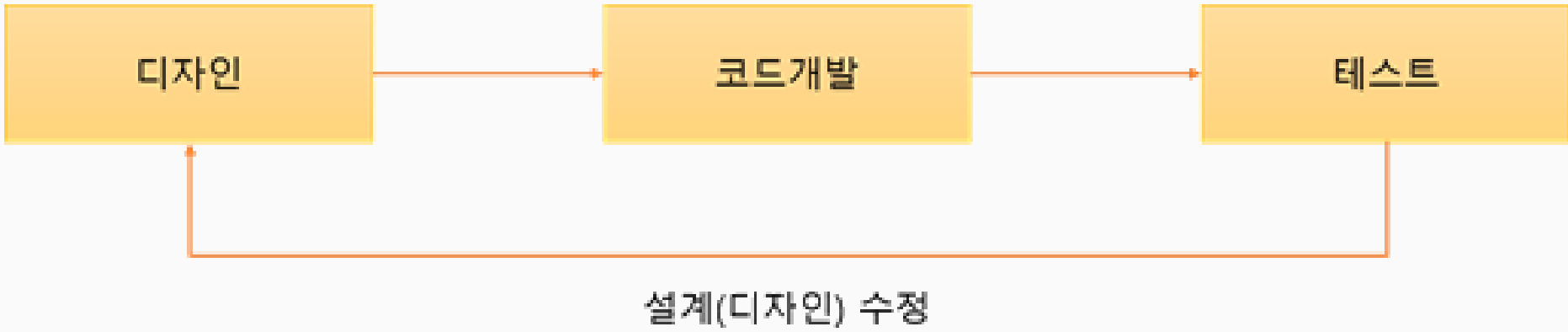
해당 과정 반복

리팩토링

좋은 코드? coding convention, 네이밍 규칙, 확장성 고려 등 신경쓸 부분이 많다.

TDD를 통해 개발을 해왔다면 테스트 코드가 그 중심을 잡아줄 수 있다.
리팩토링 속도가 빨라지고 코드의 퀄리티가 향상된다.

일반 개발 방식 VS TDD

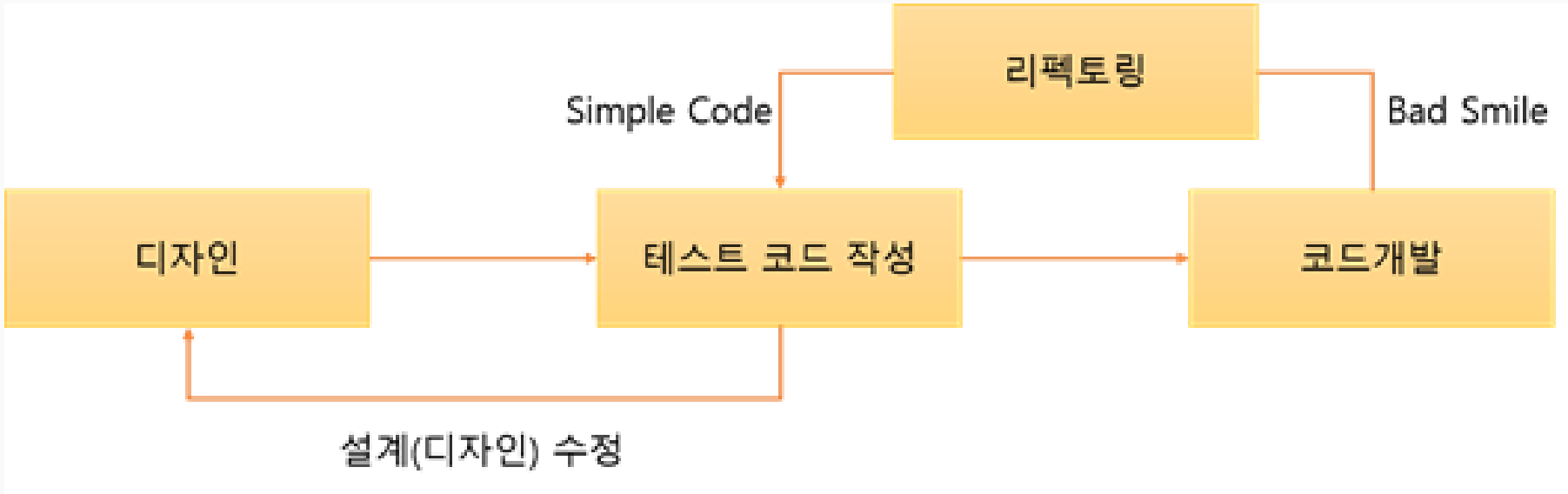


분석 -> 설계 -> 개발 -> 테스트 -> 배포

일반 개발 방식 VS TDD

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

일반 개발 방식 VS TDD



코드의 버그가 줄어들고, 소스코드가 간결해진다.

설계가 개선됨으로서 재설계 시간이 절감된다.

TDD의 장점

1. 보다 튼튼한 객체 지향적인 코드 생산
2. 재설계 시간의 단축
3. 디버깅 시간의 단축
4. 테스트 문서의 대체 가능
5. 오버 엔지니어링을 방지한다.
6. 추가 구현의 용이함

TDD의 단점

1. 생산성 저하
2. 테스트 코드를 작성하기 어렵다.

TDD를 해야 하는 상황

1. 처음해보는 프로그램 주제
2. 고객의 요구조건이 바뀔 수 있는 프로젝트
3. 개발하는 중에 코드를 많이 바꾸어야 한다고 생각하는 경우
4. 내가 개발하고 나서, 이 코드를 누가 유지보수할 지 모르는 경우