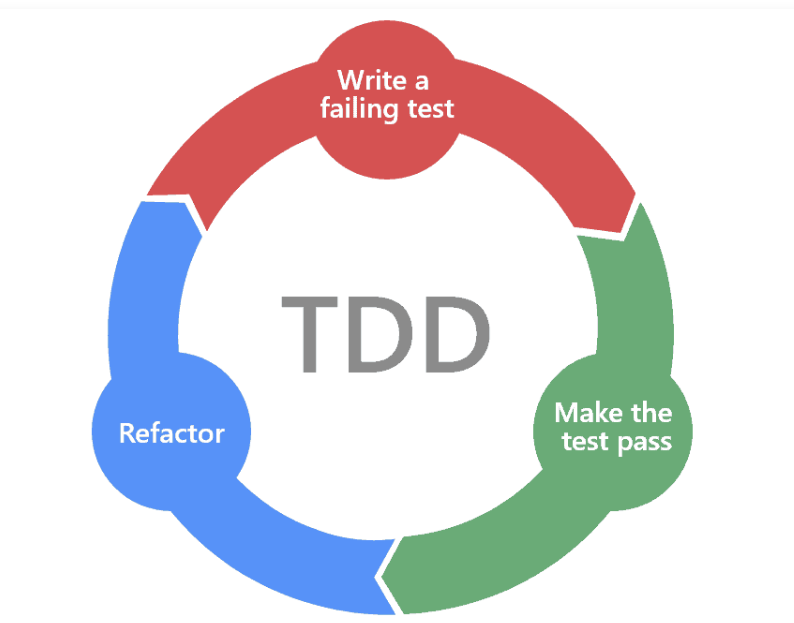
TDD(Test Driven Development): 테스트 주도 개발



ㅇ Red: 실패하는 테스트 코드 작성

Green: 테스트 코드를 성공시키기 위한 실제

코드 작성

Blue: 중복 코드 제거, 일반화 등 리팩토링

\*주의\*

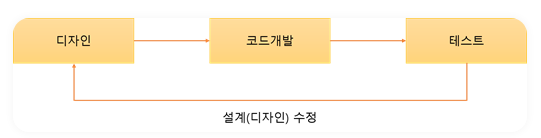
1. 실패하는 테스트 코드를 작성할 때까지 실제 코드 작성 X

2. 실패하는 테스트를 통과할 정도의 최소 실제 코드를 작성해야함

Why? 이를 통해 실제 코드에 기대되는 바를 보다 명확하게 정의하고 불필요한 설계를 피할 수 있어 정확한 요구 사항에 집중할 수 있음.

**일반 개발 방식과 TDD 개발 방식 비교**

**-일반 개발 방식**



일반적인 형태: 요구사항 분석 🡪 설계 🡪 개발 🡪 테스트 🡪 배포

이러한 방식은 소프트웨어 개발을 느리게 할 수 있음.

1. 소비자의 요구사항이 처음부터 명확하지 않을 수 있음.

2. 그래서 처음부터 완벽한 설계는 어려움.

3. 자체 버그 검출 능력이나 소스코드의 품질이 저하될 수 있음. (작은 수정에도 모든 부분을 테스트해야하기 떄문) + 이래서 잘 고치려 하지 않음

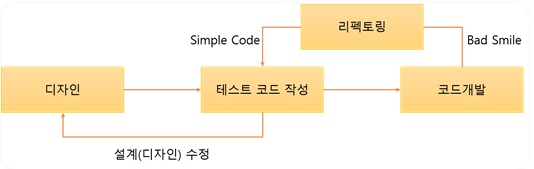
4. 자체 테스트 비용이 증가할 수 있음.

1~4 모두 어떤 프로젝트든 초기 설계가 완벽하다고 확신할 수 없기 때문이다.

고객의 요구사항, 디자인에서의 오류와 같은 많은 내부 혹은 외부 조건에 의해 재설계하여 점진적으로 완벽한 설계로 나아감. 이러한 과정 속에서 개발자는 코드를 삽입, 수정, 삭제하는 과정을 거치는데 이때 불필요한 코드가 남거나 중복될 가능성이 크다.

🡪결론적으로 이러한 코드들은 재사용이 어렵고 관리가 어려워서 유지보수를 어렵게 만든다.

**-TDD 개발 방식**



가장 큰 차이점은 테스트 코드를 작성한 뒤에 실제 코드를 작성한다는 것.

디자인(설계)단계: 프로그래밍의 목적을 반드시 미리 정해야 함. (테스트 케이스)

이중 예외 사항 발생 🡪 테스트 케이스에 추가하고 설계를 개선

테스트 통과된 코드만을 코드 개발 단계에 가 실제 코드로 작성해줌.

* 코드의 버그가 줄어들고 소스코드는 간결해짐.

**TDD 개발 방식의 단점**

생산성 저하

처음부터 2개의 코드를 짜야하고 중간중간 테스트를 하면서 고쳐나가야하기 때문

TDD 방식은 일반적인 방식에 비해 10~30%정도 개발 시간이 늘어난다.

SI프로젝트에선 소프트웨어의 품질보다는 납기일 준수가 훨씬 중요해 TDD방식을 사용하지 않는다. (SI: 시스템을 구축하는 업무)

참고: https://hanamon.kr/tdd%EB%9E%80-%ED%85%8C%EC%8A%A4%ED%8A%B8-%EC%A3%BC%EB%8F%84-%EA%B0%9C%EB%B0%9C/