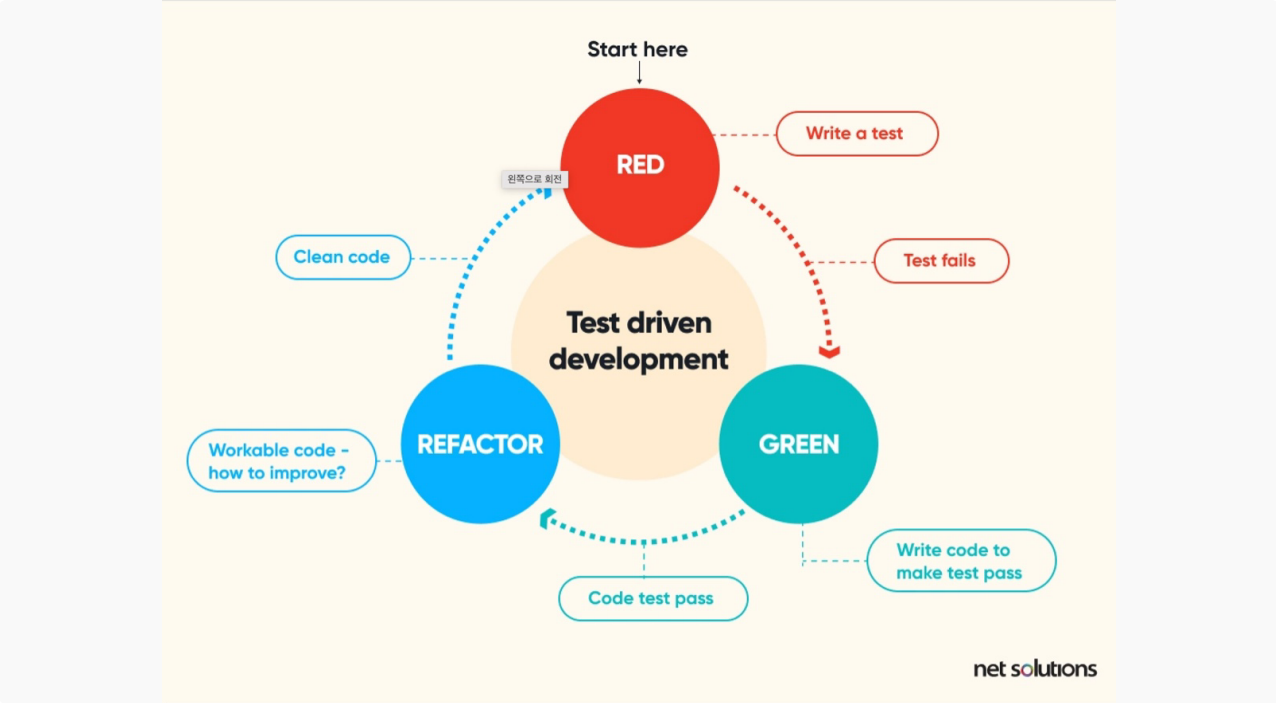
TDD(Test Driven Development)

테스트 주도 개발(Test-driven development, TDD)은 테스트 우선 개발 방법론을 사용하는 소프트웨어 개발 방식입니다. 이 방식에서는 소프트웨어를 개발하기 전에 소프트웨어 요구 사항을 유닛 테스트 케이스(특정 기능 또는 기능을 검증하는 일련의 동작)로 변환합니다. TDD는 개발이 시작되기 전에 테스트 케이스를 미리 정의하기 때문에 종종 테스트 주도 설계(Test-driven design)라고도합니다.

TDD의 원칙

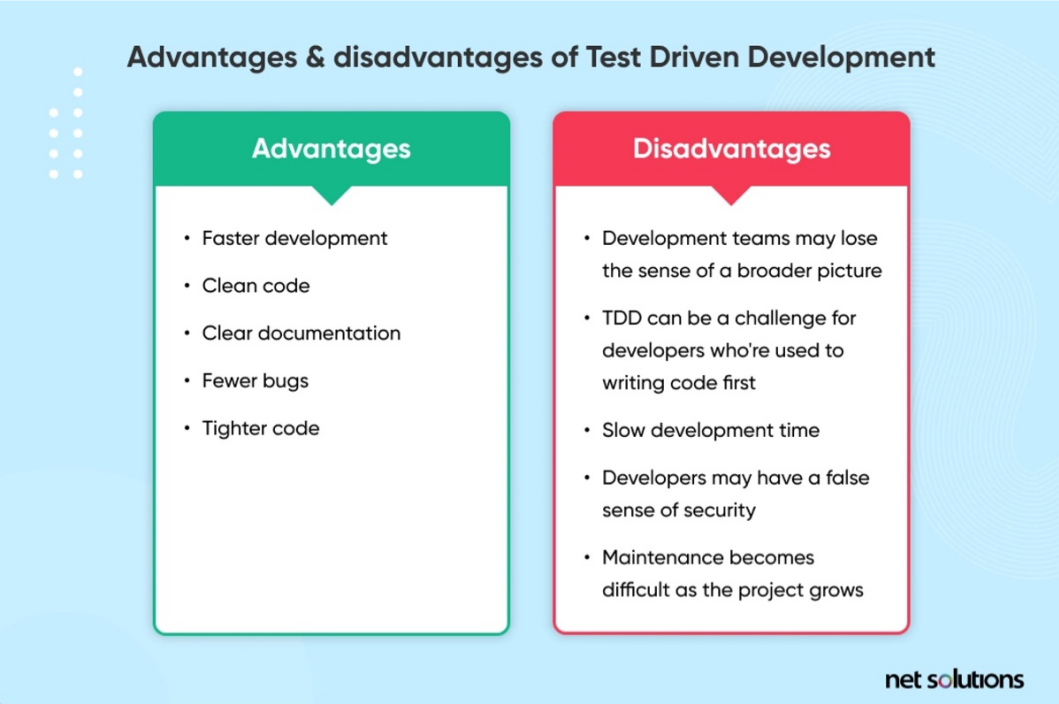
1. 제품 코드를 작성하기 전에 실패하는 테스트를 작성해야 합니다.
2. 실패하거나 컴파일되지 않을 만큼 충분하지 않은 테스트를 작성해서는 안 됩니다.
3. 현재 실패하는 테스트를 통과하기에 충분한 만큼만 제품 코드를 작성해야 합니다.



TDD 개발 방법이 어떻게 동작하는지 더 자세히 살펴보자면 다음과 같다:

- **1단계**: 특정 기능이 요구하는 사양을 충족시키는 경우에만 통과하는 유닛 테스트를 작성한다.
- **2단계**: 테스트를 실행하여 테스트가 통과하기 위해 코드가 실제로 필요한지 확인한다. 이 경우, 테스트가 잘못된 것이 아니라는 것을 확인하기 위해 실패한다.
- **3단계**: 가장 간단한 코드를 작성하여 유닛 테스트를 통과한다.
- **4단계**: 유닛 테스트에 대한 테스트를 진행하고, 테스트가 통과할 때까지 이 코드 작성 및 테스트 주기를 반복한다.
- **5단계**: 비기능적인 속성을 개선하거나 단순화하기 위해 동작을 변경하지 않고 코드를 리팩토링한다. 코드 리팩토링의 목표는 기능의 동작을 변경하지 않으면서 소프트웨어의 디자인, 구조 또는 구현을 개선하거나 성능을 향상시키는 것이다.
- **6단계**: 기능이 동일하며 버그가 도입되지 않도록 하는 것을 확인하기 위해 이러한 단계를 반복한다.

TDD의 장단점



🚀 장점

- 빠른 개발 - 단위 수준에서 테스트가 지정되기 때문에, 개발자는 무엇을 하는지 정확히 알고 있어 개발 속도를 높일 수 있습니다.
- 깔끔한 코드 - 명확성의 결과로, 과도한 엔지니어링 없이 잘 디자인되고 깨끗하며 간단한 코드를 작성할 수 있습니다.
- 명확한 문서화 - 단위 테스트의 작성으로 소프트웨어 개발의 모든 단계를 문서화할 수 있습니다.
- 적은 버그 - TDD는 각 단위를 지속적으로 테스트하므로, 소프트웨어는 회귀 결함과 버그가 적습니다.
- 더 탄탄한 코드 - 효율성을 개선하기 위한 코드의 지속적인 검토(리팩토링)는 코드 중복을 줄이고 코드 구성을 개선하여 더 탄탄한 코드를 만들어줍니다.

💔 단점

- TDD로 개발자가 세부 사항에만 집중하기 때문에, 팀 전체의 더 넓은 비즈니스 목표를 잃어버릴 수 있습니다.
- 코드를 먼저 작성하는 개발자들에게 TDD 접근법은 도전이 될 수 있으며, 이에 적응하는 데 시간이 걸려 개발 시간이 늦어질 수 있습니다.
- 테스트를 작성하는 것은 복잡한 프로젝트에 코드를 작성하는 것보다 시간이 더 걸릴 수 있으며, 제품 개발 지연 및 마감 기한 놓침으로 이어질 수 있습니다.