TDD

26 ~ 29 장

26. 빨간 막대 패턴

한 단계 테스트

테스트 목록 중 골라야 하는 것은?

- 우리에게 새로운 무언가를 가르쳐 줄 수 있으며, 구현할 수 있다는 확신이 드는 테스트를 고를 것

- 우선 정답은 없다.

➔ 환경이 프로그램에 영향을 주고 프로그램이 환경에 다시 영향을 주며 성장하는 것이 TDD의 프로세스라고 볼 수 있다

만약 방향성을 가질 필요가 있다면?

➔ 상/하향식보다는 아는 것에서 모르는 것으로가 유용하다

26. 빨간 막대 패턴

시작 테스트

어떤 테스트부터 시작하는 게 좋을까?

우리는 '빨강/초록/리펙토링'의 고리가 짧게 짧게 반복되기를 원함

➔ 오퍼레이션이 아무 일도 하지 않는 경우를 먼저 테스트 할 것 (출력과 입력이 같은 경우)

➔ 입력이 가장 적은 것을 테스트 할 것

➔ 그 다음 테스트는 나에게 뭔가를 가르쳐 줄 수 있으며 빠르게 구현할 수 있는 테스트를 선택

설명 테스트

팀에 TDD를 전파하는 방법

설명을 요청/요구 시 테스트를 수단으로 사용하라

EX) 시퀀스 다이어그램의 모든 요소를 테스트 케이스로 보여주는 방식으로 설명

26. 빨간 막대 패턴

학습 테스트

- 외부 패키지의 새로운 기능을 처음으로 사용해보기 전에 테스트를 작성해볼 수 있다.

- 새로운 API가 우리 예상대로 실행됨을 확인 할 수 있는 테스트를 사용한다.

+ 버전이 변경되면서 다르게 동작하는 경우 확인 가능

또 다른 테스트

- 주제와 무관한 아이디어가 떠오르면 이에 관한 테스트를 할 일 목록에 적고 다시 주제로 돌아오기

- 대화를 엄격하게 한 주제로 묶는 것은 좋은 아이디어를 억압하는 방법이다.

- 때때로 프로그래밍은 뭔가를 훌쩍 뛰어 넘는 기회에 의존하므로, 새 아이디어를 존중하고 맞이하되 내 주의를 흩뜨리지 않게 한다

26. 빨간 막대 패턴

회귀 테스트

- 시스템 장애로 인해 실패하는 테스트, 통과하는 경우엔 장애가 수정되었다고 볼 수 있는 테스트

- 처음 코딩할 때 작성했어야 하는 테스트

- 애플리케이션 차원의 회귀 테스트는 시스템의 사용자들이 무엇을 기대했고 무엇이 잘못되었는지 말할 기회를 줌

EX) 오버플로우 테스트

다시 하기

- 길을 잃은 느낌이 들 때는 코드를 다 지워버리고 처음부터 다시 해보자

싸구려 책상 좋은 의자

- 허리가 아프면 프로그램을 잘 짤 수가 없다

27장. 테스팅 패턴

상세한 테스트 작성법

자식 테스트

지나치게 큰 테스트를 어떻게 돌아가도록 할 수 있을까?

➔ 원래 테스트 케이스의 깨지는 부분에 해당하는 작은 테스트를 작성하고 그 작은 테스트를 실행되도록 하라 -> 하나하나 해결해서 결국 큰 테스트가 해결되도록 하라

➔ 즉, 큰 테스트를 작은 테스트로 나눠라

➔ WHY? 빨강/초록/리펙토링 리듬을 잃어버리지 않기 위해 + 빨간 막대가 주는 압박감이 있기 때문

27장. 테스팅 패턴

상세한 테스트 작성법

모의 객체

- 비용이 많이 들거나 복잡한 리소스에 의존하는 객체를 테스트 하려면 이를 본 딴 모의 객체를 만들어서 테스팅해라

- 모의 객체의 가치는 성능과 견고함 이외에도 가독성이 있다.

- "모의 객체가 진짜 객체와 동일하게 동작하지 않으면?"에 대한 걱정은 모의 객체용 테스트를 진짜 객체에 적용해서 확인

SELF SHUNT – 자기가 보낸 것이 다시 자신에게 돌아오는지 확인하는 루프백 테스트

- 테스트 케이스가 구현할 인터페이스를 얻기 위해 인터페이스 추출을 해야한다.

➔ 인터페이스를 추출하는 것과 존재하는 클래스를 블랙 박스로 테스트하는 것이 더 쉬운지를 결정해야한다

27장. 테스팅 패턴

상세한 테스트 작성법

로그 문자열

- 메시지의 호출 순서가 올바른지 검사할 때 사용
- 메시지가 호출될 때 마다 로그 문자열에 추가

크래시 테스트 더미

- 잘 호출되지 않을 것 같은 에러를 테스트 하는 방법은, 그냥 예외를 발생시키는 특수한 객체를 만들어서 이를 호출한다.

깨진 테스트

혼자서 프로그래밍 할 때는 마지막 테스트가 깨진 상태로 끝마치는 것이 좋다

➔ 다음 작업 시작 시 할 일을 바로 알 수 있다

➔ 일종의 책갈피 역할을 하는 것

27장. 테스팅 패턴

상세한 테스트 작성법

깨끗한 체크인

팀 프로그래밍을 할 때는 모든 테스트가 성공한 상태로 끝마치는 것이 좋다

- 자신이 마지막으로 코딩한 이후 지금까지 무슨 일이 있었는지 자세하게 모르기 때문에 안심되고 확신이 있는 상태에서 작업을 시작할 필요가 있다

➔ 체크인 전에 항상 모든 코드가 돌아가는 상태로 만들어놔야 한다

➔ 테스트 슈트가 실패하면 작업한 코드를 전부 날려버려라 - 우리가 만들어낸 프로그램을 우리가 완전히 이해하지 못한 상황이므로

28장. 초록 막대 패턴

깨진 테스트 고치기

가짜로 구현하기(진짜로 만들기 전까지만)

- 실패하는 테스트를 만들었다면, 상수를 반환하게 만들어라
- 테스트가 통과하면 단계적으로 상수를 변수로 변경(리펙토링)하라

심리학적 효과 : 확신을 갖고 거기부터 리펙토링 할 수 있다

범위 조절 효과 : 현재의 문제에만 집중하도록 해준다

삼각측량

- 예가 2개 이상일 때만 삼각측량법을 이용해서 추상화하라
- 어떻게 해야 올바르게 추상화 할 수 있을지 모르겠을 때 사용하는 방식이다
- 그 외의 경우라면 명백한 구현이나 가짜로 구현하기에 의존하자

28장. 초록 막대 패턴

깨진 테스트 고치기

명백한 구현

- 뭘 해야 할 지 알고, 그걸 재빨리 할 수 있다면 그냥 해버려라
- '제대로 동작하는'과 '깨끗한 코드'를 해결하는 것을 한번에 하기 힘들다면, '제대로 동작하는'부터 해결하고 '깨끗한 코드'를 해결하라
- 손가락이 머리를 따라오지 못하면 앞의 방식으로 해결하라

29장. xUnit패턴

단언(Assertion)

- 판단의 결과는 Boolean 값이어야 한다
- 단언을 작성할 때, 특정 값을 명시 할 것 (예외 값을 피하는 형태의 코드는 기피)
- 즉, 단언은 구체적이어야 한다

픽스처

- 객체들을 세팅하는 코드가 여러 테스트에 걸쳐 동일한 경우가 있다 (이러한 객체를 테스트 픽스처라고 한다) - 이런 중복은 장단점 존재

장점 - 테스트 코드를 그냥 위에서 아래로 읽어내려갈 수 있다

단점 - 인터페이스를 수동으로 변경할 필요가 있을 경우, 여러 테스트를 고쳐줘야 한다

- 작성하는데 시간이 많이 소요된다

29장. xUnit패턴

외부 픽스처

테스트의 목적 중 하나는 테스트 실행 전/후의 외부 세계가 동일하게 유지되도록 만드는 것이다

- 픽스처 중 외부 자원이 있을 경우 tearDown() 메서드를 재정의 해서 자원을 해체하자
- xUnit은 setUp()이 정상적으로 실행됐다면, 테스트가 끝난 후에 tearDown()을 수행한다

29장. xUnit패턴

테스트 메서드

- 관습에 의해 테스트 메서드의 이름은 test로 실행한다
- 테스트 메서드 이름의 나머지 부분은 아무것도 모르는 사람이 보더라도 이해할 수 있을 만큼 구체적이고 명확해야 한다
- 테스트 메서드는 의미가 드러나는 코드로 읽기 쉬워야 한다

⇨ 가능한 짧게 작성하며, 두 세줄 정도의 아웃라인을 작성하는 것도 좋은 방법이다

- 픽스처를 사용하기 위해 클래스를 사용한다면, 테스트 메서드가 각각의 테스트가 된다

29장. xUnit패턴

예외 테스트

예외가 발생하는 것이 정상인 경우에 대한 테스트

- 예상되는 예외에 대해서만 catch한다
- 그 외의 경우는 fail()을 호출 하도록한다 (다른 예외 혹은 예외가 발생하지 않은 경우 실패)

전체 테스트

모든 테스트를 한번에 실행하려면?

- 모든 테스트 슈트에 대한 모음을 작성한다
- 패키지에 대해 테스트 슈트를 모으고, 각 패키지 별로 해당 모음을 다시 모아 애플리케이션 레벨의 테스트 슈트를 만든다