TDD가 하고 싶어요

김지훈

여러분은 개발을 어떻게 하고 계신가요?

여러분은 개발을 어떻게 하고 계신가요?

- 회의 갔다 와서
- 기능명세 보고
- 열심히 코딩
- 다되면 전달

여러분은 개발을 어떻게 하고 계신가요?

- 회의 갔다 와서
- 기능명세 보고
- 열심히 코딩
- **?? 테스트 ??**
- 다되면 전달

여러분은 개발을 어떻게 하고 계신가요?

- 회의 갔다 와서
- 기능명세 보고
- 열심히 코딩
- **?? 테스트 ??**
  - 이게 TDD 아냐?
- 다되면 전달

우린 아마 SDD를 하고 있을겁니다.

우린 아마 SDD를 하고 있을겁니다.

Self Driven Development

Self Driven Development

- 생각하고, 만들고, 테스트 하고
- 평소에 일하시면서...
- 아마 과제하면서...

Do you know TDD?

TDD

- Test Driven Development

테스트 주도 개발

- “결정과 피드백 사이의 갭에 대한 인식"
  간극을 조절하기 위한 테크닉
- 결정 - ‘이렇게 프로그래밍 해야지'
- 피드백 - '이게 되네?' 혹은 '안되네?' 와 같은 피드백

저는 테스트 코드를 적는데요?

- 테스트 코드를 적는다 != TDD

테스트 코드를 적는다

자동화 테스트

테스트 코드를 적는다

자동화 테스트

내가 안하니까 자동임

TDD는 Test 하는 방법이나 종류를 다루지는 않습니다

TDD는 Test 하는 방법이나 종류를 다루지는 않습니다

다만, 테스트가 불가결 하기 때문에, 테스트를 하기에 적합한 방법을 배울거에요

TDD 지향점

"clean code that works"

"clean code that works"

군더더기 없는 코드

(Short Version) TDD

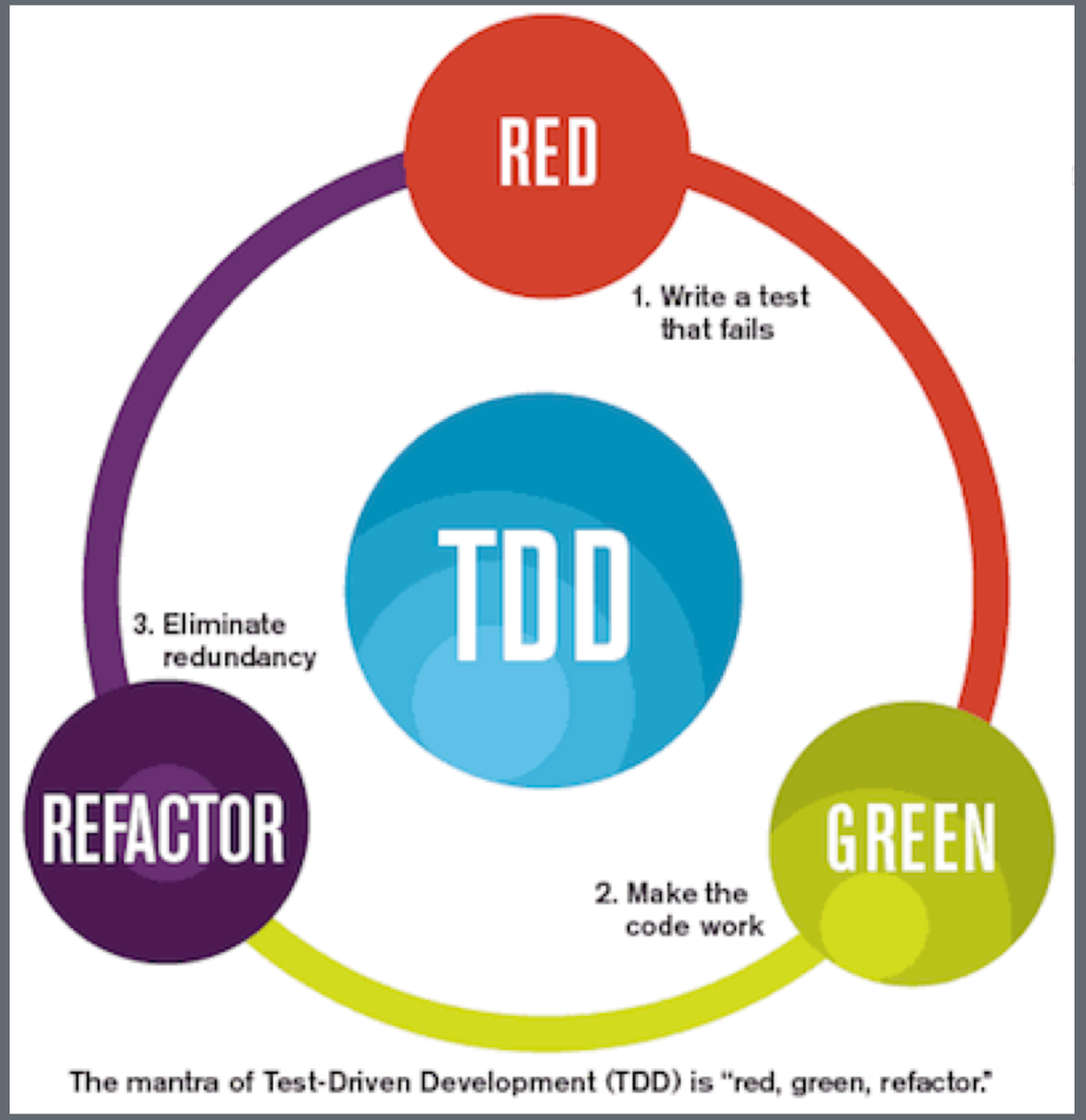
- red, green, blue

(Short Version) TDD

- fail, success, refactor

(Short Version) TDD

- Red(fail)
  - 해결하려는 문제에 대한 테스트를 만들고,
    실행시켜서 Fail을 얻어낸다
- Green(success)
  - 가장 적은 비용으로 테스트를 성공시킨다
- Blue(refactor)



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Test Frist Development + Refactor

TDD 지향점 +

TDD 지향점 +

- TDD를 통해 군더더기 없고, 좋은 구조를 만들게 됨
  - 군더더기?
  - 좋은 구조

TDD 지향점 +

- 군더더기?
  - Test 외의 구현것들
  - e.g. 이런거 만들어 놓으면 쓰지 않을까?
- 좋은 구조
  - 명확한 책임 분리
  - 관심사의 집중(SRP, OCP)

자 TDD 를 해봅시다

자 TDD 를 해봅시다

뭐부터 해야 하죠?

TDD

1. 우선 풀고자 하는 문제를 명확히 한다
2. 문제 중 하나를 테스트로 적는다
    - TDD 에서의 테스트는 단위테스트임
3. 테스트를 최소한의 리소스로 성공시킨다
4. 테스트를 부시지 않고 제대로 구현한다

TDD 를 위한 지침

TDD 를 위한 지침

테스트 코드를 짤 때 이걸 기억하면서 짜야함

TDD 를 위한 지침

- 무엇을 테스트 할 것인가? (테스트 목적)
- 가능한 작게 작게 테스트 해야함
- 하나하나가 atomic 해야함

TDD 를 위한 지침

- transactional 해야함
- 순서가 없어야 함

TDD를 전파 하기 위한 지침

TDD를 전파 하기 위한 지침

혼자하긴 힘듬, 좋은건 같이하자

TDD를 전파 하기 위한 지침

- 나랑 맘이 맞는 사람을 꼭 섭외하자
- CI에 테스트 task를 넣고 공유하자
- 코드 퀄리티에 따른 활동을 공유하자

다루지 않은 내용

다루지 않은 내용

- BDD (Behavior Driven Development)
- DDD (Domain Driven Design)

스터디 하면서 다룰 내용

- 테스트 도구들

아마도... JUnit

- 4? 5?
- 차이점
  - https://www.baeldung.com/junit-5-migration
  - https://howtodoinjava.com/junit5/junit-5-vs-junit-4/
  - https://howtodoinjava.com/junit5/junit-5-vs-junit-4/

끗

끗 하기 전에

TDD 단점 타임

TDD 가 항상 옳은가?

- 사용하려는 언어와 프레임 워크에 익숙해지는게 먼저
- TDD가 만병통치약이 아니다
- TDD는 오버헤드가 있다.

TDD is dead. Long live testing.

TDD 뭬짐.

진짜?

DHH vs Kent Beck (feat, Martin Fowler)

정리된 글은 요기에..

바쁜 사람들을 위한 논점 요약

DHH의 주장 요점정리

- **많은 개발자들이 TDD를 사용하지 않는 코드는 더티한 코드라고 생각하도록 몰아넣고 있다.** - 사실상 DHH가 이 논쟁을 시작하도록 한 직접적인 원인이 아닌가 싶다. 누군가 DHH에게 "니 코드는 구려. 유닛 테스트가 없잖아?" 이렇게 말한것이 아닐까?
- **유닛테스트 주도의 설계는 좋은 생각이 아니다.** - TDD 근본주의자들은 제대로 된 설계는 테스트하기가 쉽다고 말한다. 필자는 이것을 부정하진 않는다. 하지만 '테스트하기 좋은 설계' = '좋은 설계'가 성립하는것은 아니다.
- **TDD의 개념인 "테스트는 빨라야 한다"는 근시안적인 생각이다.** - 아무래도 테스트의 속도는 개발자 환경에선 부담이 될 수밖에 없다. 다행이도 젠킨스와 같은 자동화CI 툴이 도입된 이후로는 이러한 부담이 많이 줄었다. 자동화 테스트를 돌릴 수 있는 환경만 만들어 놓는다면 개개인은 자신의 코드에 대한 풀테스트 결과는 CI서버 상에서 확인 가능하게 되며 개인레벨에서 코드 커밋 이전에 수행해야만 하는 테스트는 코드에 의해 새로 작성된 테스트 케이스의 결과 확인만으로 충분할 것이다.
- **TDD에 대한 믿음은 시스템 테스트를 완전히 잊어버리도록 만든다.** - DHH의 이러한 생각은 시스템 테스트나 시나리오 테스트를 자동화 하도록 권장하는 BDD의 영향을 받은듯 싶다. 실제로 Ruby에서는 BDD툴인 RSpec 이 큰 인기를 누리고 있다.
- **유닛테스트에 포커싱을 하거나 유닛테스트만을 행하는것은 큰 규모의 시스템을 만드는데에 도움이 되지 않는다.** - 큰 규모의 시스템을 만들기 위해서는 반드시 여러 레이어를 횡단하는 테스트의 도움이 필요하다.
- **100% 커버리지는 웃기는 이야기다** - 100% 커버리지 달성에 목매는 개발현장을 심심찮게 본다. 이런 현장에는 십중팔구 생각하기 싫어 하는 PM이 존재한다. 100% 커버리지는 품질에 대한 그 어떤 보장도 되어주질 않는다.
- **프로그래머는 소프트웨어가 과학이 되길 원한다. 하지만 소프트웨어는 과학이 아니다. 그것은 좀 더 문학 창작에 가깝다.** - 얼마전에 해커와 화가를 읽으면서 공감했던 부분.
- **좋은 소프트웨어는 엔지니어링과는 다르다.**
- **그것은 마치 글쓰기와 같다. 명확하고 간결한 글이 난해한 글보다 낫다.**
- **명확함은 좋은것이다. 따라서 테스트 커버리지나 테스트 스피드가 아니라 명확함이야 말로 추구하는 목표중 하나가 되어야 한다.**
- **좋은 개발자가 되는것은 좋은 작가가 되는것만큼 어렵다.** - 아무리 그래도 좋은 작가가 되기 보다는 쉽지 않을까? 아무리 생각해 봐도 좋은 개발자의 수가 좋은 작가의 수보다는 많아 보인다.
- **글쓰기와 마찬가지로 좋은 프로그래머가 되기위한 가장 명확한 방법은 많은 소프트웨어를 만들어 보고, 또 좋은 소프트웨어의 코드를 읽어보는것이다.** - 100%공감한다.

한 번 더 요약

- TDD는 오버헤드가 너무 크다
- TDD가 설계를 망친다
- TDD가 테스트를 망친다

conclusion

- 오버헤드가 있는 개발방법론이지만,
- 로직을 더 간결하게 짜려고 노력하게 되고,
- 꼼꼼하게 코딩할 수 있
- 코드를 격리 시킬 수 있음
- 100% TDD 를 하지 않아도 된다 (사견)