Desenvolvimento orientado a testes (TDD)

Prof. Eric Sales

Por que testar?

- Garantir que atendemos as necessidades de nossos clientes;

- Ter segurança e autoconfiança com o que fizemos.

Como testar?

- Manualmente
- Automaticamente

Por que automatizar?

- Permitir repetir constantemente os testes (regressão);
- Ganhar velocidade de desenvolvimento;
- Aumentar a confiabilidade do SW e autoconfiança do DEV.

Por que automatizar? Refatorar!

- Fazer melhorias no código;
- Com automação é possível modificar o código e **garantir que tudo funciona**

"If you have a test suite that you trust so much that you are willing to deploy the system based solely on those tests passing; and if that test suite can be executed in seconds, or minutes, then you can quickly and easily clean the code without fear."
~Robert C. Martin

O que é melhor?

- Fazer os testes depois do código produzido?
- Fazer os testes antes do código produzido?
- Ambos?

Por que testar primeiro?

- Será que você irá lembrar de testar depois?
- Quando fazemos testes depois podemos ser influenciados pelo código;
- Refletimos sobre o código e consequentemente aumentamos a garantia de fazer corretamente.

TDD

Desenvolvimento orientado a testes

Fazemos o(s) teste(s) para um código que ainda não existe!

TDD não é...

... apenas escrever testes antes do código.

...apenas focar nos testes.

TDD é...

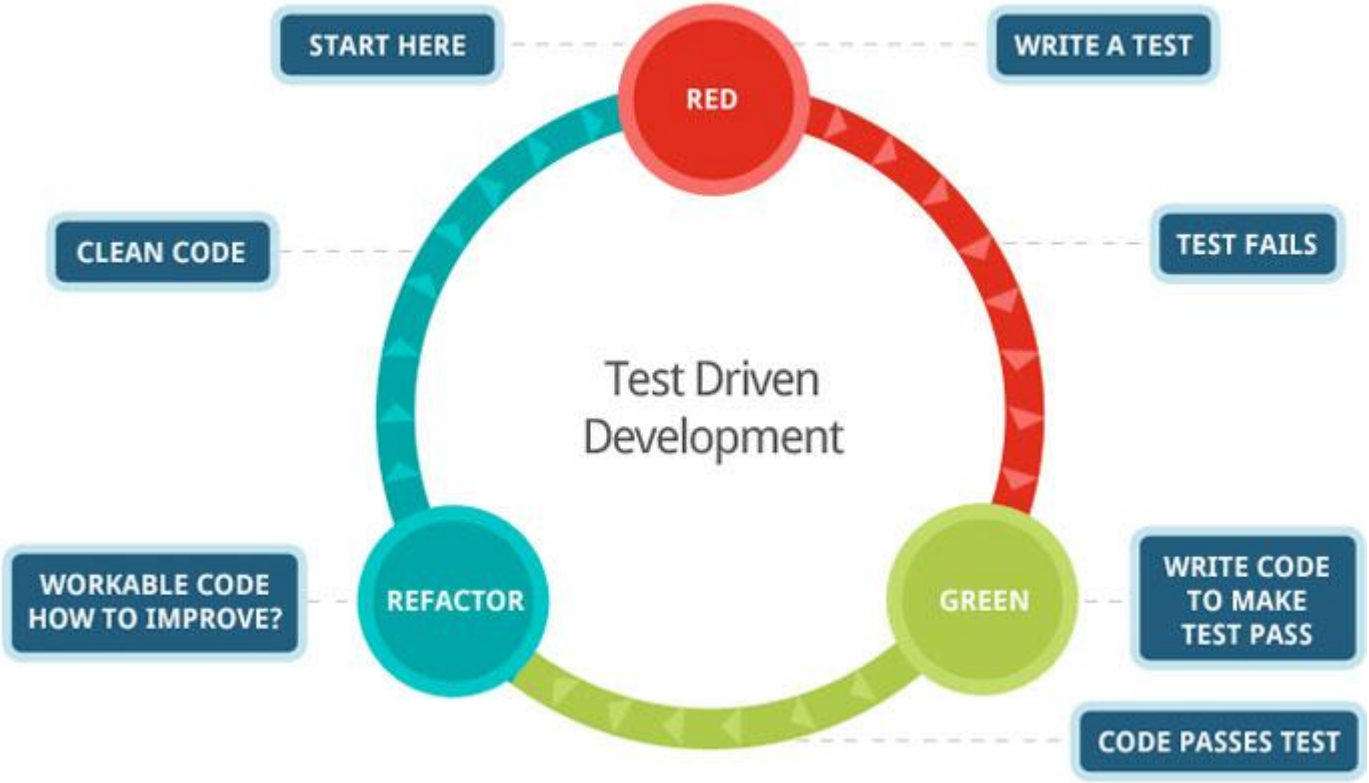
... escrever testes antes dos códigos.

... focar em melhorar o design do seu código.

Benefícios do TDD

- Melhoria na qualidade do código (clean code e menos acoplamento);
- Defeitos encontrados mais cedo;
- Maior segurança na refatoração;
- Maior produtividade (menos tempo depurando).

Fluxo de trabalho TDD



Fonte: http://techarcis.blogspot.com/2015/09/various-agile-testing-practices-being.html

Fase: RED

- O teste existe!
- O código não existe!
- Então, rodamos o teste e ele deve FALHAR.
- Pense no que você irá desenvolver!

Fase: GREEN

- O teste existe!
- O código foi criado!
- Então, rodamos o teste e ele deve “PASSAR”.
- Pense em como passar no teste.

Fase: Refactor

- Durante a implementação -> Foco na resolução do problema;
- O código é feito com qualidade?
- Neste momento o desenvolvedor pode focar no clean code.
- Pense em como melhorar a implementação.

Como trabalhar com o TDD

1. Pensar em quais requisitos uma determinada funcionalidade deve oferecer

"O cadastro de um usuário deve **indicar o resultado da operação (sucesso ou falha)**, como também **adicioná-lo ao banco de dados** e **enviar notificação por e-mail** para confirmação da conta".

Testes tradicionais x TDD

1. Pensar em quais requisitos uma determinada funcionalidade deve oferecer
2. Escrever um teste que irá falhar

Escrever um teste que irá falhar

```
@Test
public void testarCadastroDeUsuario(){
    Sistema s = new Sistema();
    assertTrue(s.cadastrar(null));
}
```

O método cadastrar ainda não existe na classe sistema!!

Testes tradicionais x TDD

1. Pensar em quais requisitos uma determinada funcionalidade deve oferecer
2. Escrever um teste que irá falhar
3. Escrever um código para passar no teste

Escrever o código para passar no teste

```
@Test
public void testarCadastroDeUsuario(){
    Sistema s = new Sistema();
    assertTrue(s.cadastrar(null));
}
```

```
public boolean cadastrar(Usuario usuario){
        return true;
}
```

Testes tradicionais x TDD

1. Pensar em quais requisitos uma determinada funcionalidade deve oferecer
2. Escrever um teste que irá falhar
3. Escrever um código para passar no teste
4. Refatorar

Refatoramento

Escrever um teste para falhar

```
@Test
public void deveCadastrarUmUsuario(){
	Sistema s = new Sistema();
	assertTrue(s.cadastrar(null));
}

@Test(expected=IllegalArgumentException.class)
public void deveFalharComUsuarioNull(){
		Sistema s = new Sistema();
		s.cadastrar(null);
}
```

Escrever um teste para falhar

```
@Test
public void deveCadastrarUmUsuario(){
	Sistema s = new Sistema();
	Usuario u = new Usuario();
	assertTrue(s.cadastrar(u));
}

@Test(expected=IllegalArgumentException.class)
public void deveFalharComUsuarioNull(){
		Sistema s = new Sistema();
		s.cadastrar(null);
}
```

Refatorar o próprio teste

Escrever o código para passar no teste

```
public boolean cadastrar(Usuario usuario){
        if( usuario == null ){
        throw new IllegalArgumentException();
    }
        return true;
}
```

Escrever o código para passar no teste

```
@Test
public void deveCadastrarUmUsuario(){
        Sistema s = new Sistema();
        Usuario u = new Usuario();
        assertTrue(s.cadastrar(u));
}

@Test(expected=IllegalArgumentException.class)
public void deveFalharComUsuarioNull(){
                Sistema s = new Sistema();
                s.cadastrar(null);
}


public boolean cadastrar(Usuario usuario){
                if( usuario == null ){
                throw new IllegalArgumentException();
        }
                return true;
}
```