Por que usar TDD?

Quem eu sou

- Felipe Martins, fefas
- Paulista
- Bacharel em Física
- Desenvolvedor na UPX Technologies
- +5 anos na área
- PHP (também trabalhei com C++, Python, Perl, VB6 e NodeJS)
- Hambúrguer, música (rock) e idiomas

@fefas (ou @eufefas)

~~Por que usar TDD?~~

Introdução ao TDD

O que me leva a estudar TDD e o abordar em uma palestra?

Nós, desenvolvedores, somos orgulhosos

Nós, desenvolvedores, somos orgulhosos

Nossos sistemas devem:

- ser eficiente
- ser genérico
- responder bem ao negócio
- usar os mais novos *tech-hacks*
- ser o melhor

Nós, desenvolvedores, somos orgulhosos

Nossos sistemas devem:

- ser eficiente
- ser genérico
- responder bem ao negócio
- usar os mais novos *tech-hacks*
- ser o melhor

Desenvolvedores acabam gastando tempo:

- imaginando como as coisas serão
- imaginando todos os casos do uso
- definindo design do banco
- prevendo como todas as classes e módulos irão se relacionar

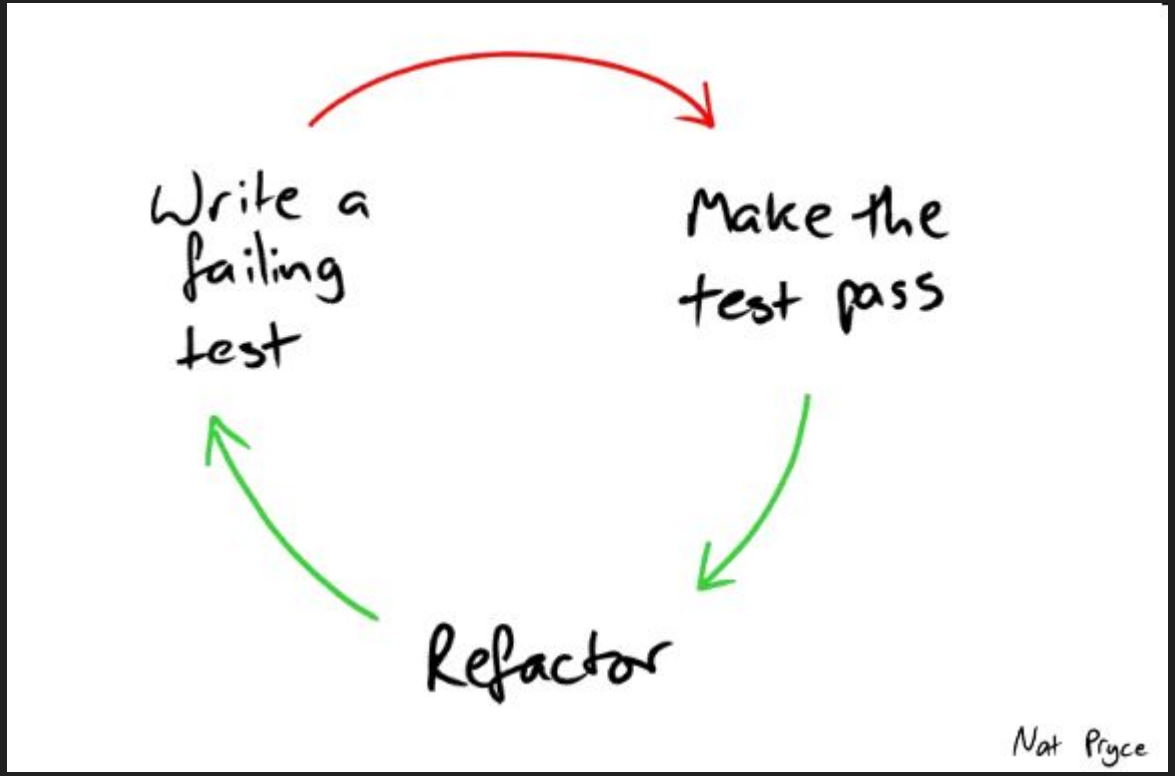
Mas e aí?
Será que está é a melhor forma?

Acho que estou fazendo algo errado...

TDD bateu na minha cara!

TDD é boa parte das respostas que buscamos

Afinal, o que é TDD?



1. Escrever teste que falha
2. Fazer teste passar
3. Refatorar

Ciclo fundamental de *feedback* do TDD

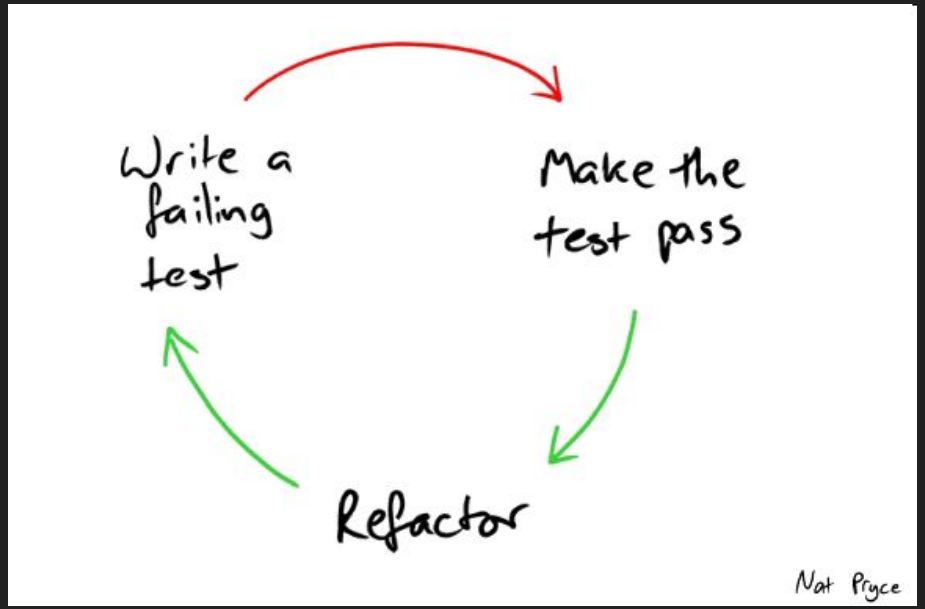
Certo... e aí?

Como tirar proveito destes passos?

Test Driven Development

Desenvolvimento guiado por testes

Vamos tentar perguntar:
qual passo seria o mais importante?



- Pense no próximo passo
- Faça o mínimo (trivial)
- Melhore a qualidade

Beleza, vamos guardar isso..

Quais os níveis de testes e seus casos de uso?

Níveis de teste

1. Aceitação (ou funcional):

   O sistema funciona por completo?

   ...

Níveis de teste

1. Aceitação (ou funcional):

   O sistema funciona por completo?

2. Integração:

   Como o nosso código se comporta contra código que não podemos mudar?

   ...

Níveis de teste

1. Aceitação (ou funcional):

   O sistema funciona por completo?

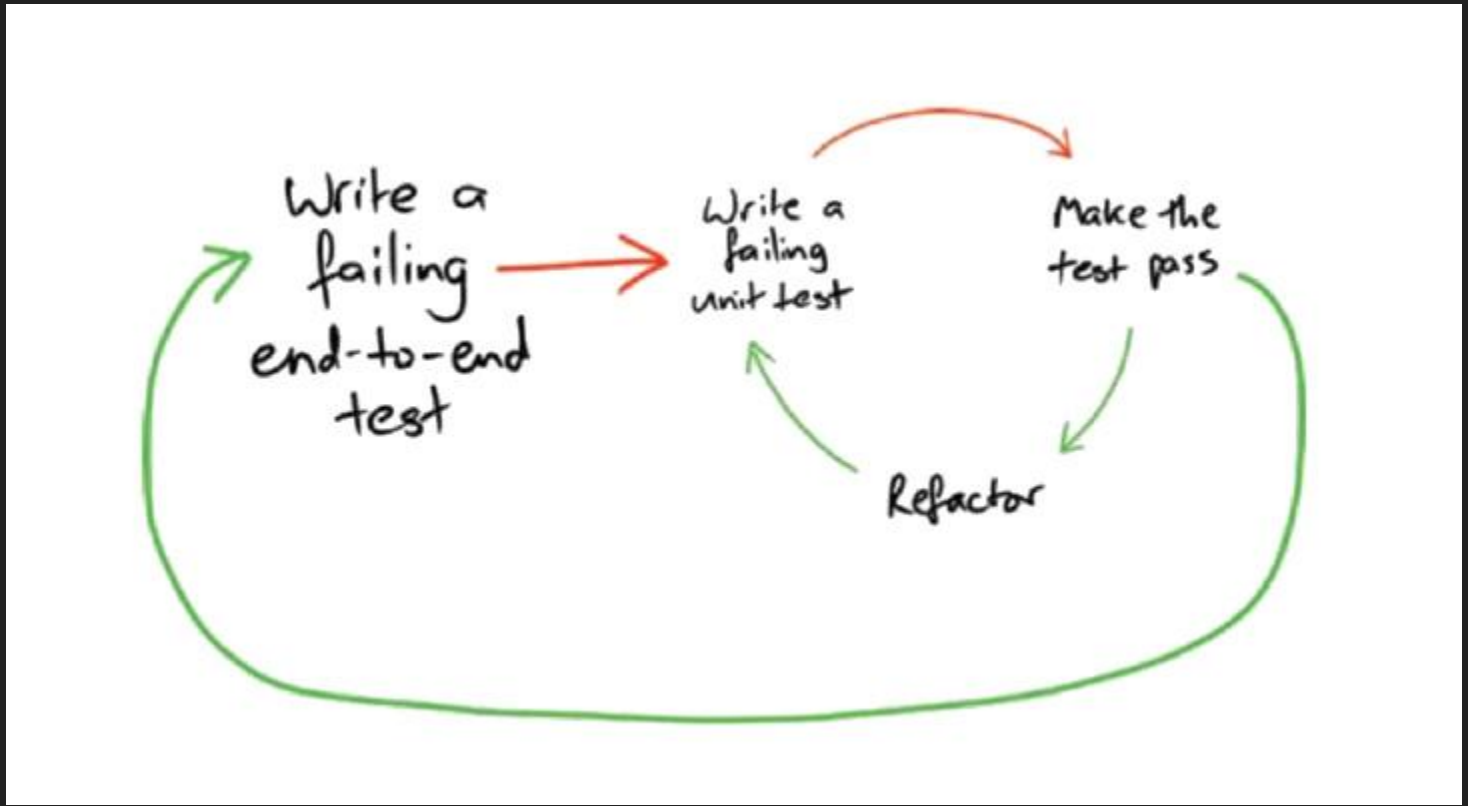
2. Integração:

   Como o nosso código se comporta contra código que não podemos mudar?

3. Unitários:

   Os nossos objetos fazem a coisa certa? É conveniente de trabalhar com eles?

Se formos começar uma API agora, qual seria o ponto de partida?



Ciclo global de *feedback* do TDD

Bem, vamos tentar..

Precisamos validar códigos de usuários no FORM de cadastro:

1. Pode ser alfanumérico
2. Não pode haver conflito

Vamos usar Behat e PHPUnit.

```
Request:

POST /check-username

{
    "username": "fefas"
}

--

Responses:

> 200 Ok
> 422 Unprocessable Entity
> 409 Conflict
```

Escrevendo o primeiro teste

Vamos começar implementando o caminho de sucesso:

```
Feature: Check username format and availability
  In order to check if I can use an username
  As an user
  I want to request POST /check-username with the username in the body

  Scenario: The username has a valid format and is available
    When I request POST "/check-username" with the following body:
      """
      {
        "username": "fefas"
      }
      """
    Then the response status should be 200
    And the response body should be empty
```

Vamos rodar o teste

```
Feature: Check username format and availability
  In order to check if I can use an username
  As an user
  I want to request POST /check-username with the username in the body

  Scenario: The username has a valid format and is available       # tests/features/che
ck-username.feature:6
    When I request POST "/check-username" with the following body: # HttpClientContext:
:iRequestPostWithTheFollowingBody()
      """
      {
        "username": "fefas"
      }
      """
      cURL error 6: Could not resolve host: nginx (see http://curl.haxx.se/libcurl/c/li
bcurl-errors.html) (GuzzleHttp\Exception\ConnectException)
    Then the response status should be 200                         # HttpClientContext:
:theResponseStatusShouldBe()
    And the response body should be empty                          # HttpClientContext:
:theResponseBodyShouldBeEmpty()

--- Failed scenarios:

    tests/features/check-username.feature:6

1 scenario (1 failed)
3 steps (1 failed, 2 skipped)
0m0.11s (10.04Mb)
```

Vamos atacar o primeiro erro

1. Configurar *webserver*
2. Configurar phpfpm
3. Criar um index vazio

```
Feature: Check username format and availability
  In order to check if I can use an username
  As an user
  I want to request POST /check-username with the username in the body

  Scenario: The username has a valid format and is available       # tests/features/che
ck-username.feature:6
    When I request POST "/check-username" with the following body: # HttpClientContext:
:iRequestPostWithTheFollowingBody()
      """
      {
        "username": "fefas"
      }
      """
      cURL error 6: Could not resolve host: nginx (see http://curl.haxx.se/libcurl/c/li
bcurl-errors.html) (GuzzleHttp\Exception\ConnectException)
    Then the response status should be 200                         # HttpClientContext:
:theResponseStatusShouldBe()
    And the response body should be empty                          # HttpClientContext:
:theResponseBodyShouldBeEmpty()

--- Failed scenarios:

    tests/features/check-username.feature:6

1 scenario (1 failed)
3 steps (1 failed, 2 skipped)
0m0.11s (10.04Mb)
```

O primeiro teste passou! \o/

w/index.php

```
1 <?php
```

```
Feature: Check username format and availability
  In order to check if I can use an username
  As an user
  I want to request POST /check-username with the username in the body

  Scenario: The username has a valid format and is available       # tes
    When I request POST "/check-username" with the following body: # Htt
      """
      {
        "username": "fefas"
      }
      """
    Then the response status code should be 200                   # Htt
    And the response body should be empty                         # Htt

1 scenario (1 passed)
3 steps (3 passed)
0m0.04s (9.23Mb)
```

Viu só?

Já temos uma API.. mostra pro chefe!

Agora temos com o que trabalhar

O Esqueleto Andante

Agora sim!

O que garantiu o funcionamento foi o teste

w/index.php

```php
<?php

use Psr\Http\Message\ServerRequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;
use Slim\App;

require __DIR__.'/../vendor/autoload.php';

$app = new Slim\App();

$app->post('/check-username', function (Request $request, Response $response) {
    return $response->withStatus(200);
});

$app->run();
```

Próximo caso: *username* não enviado

```
Scenario: The username is not provided
  When I request POST "/check-username" with the following body:
    """
    {
    }
    """
  Then the response status code should be 422
  And the response body should be:
    """
    {
      "message": "The field 'username' is missing"
    }
    """
```

```php
<?php

use Psr\Http\Message\ServerRequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;
use Slim\App;

require __DIR__.'/../vendor/autoload.php';

$app = new Slim\App();

$app->post('/check-username', function (Request $request, Response $response) {
    $parsedBody = json_decode($request->getBody()->getContents(), true);
    $username = $parsedBody['username'] ?? null;

    if (null === $username) {
        return $response
            ->withStatus(422)
            ->withJson(['message' => 'The field \'username\' is missing']);
    }

    return $response->withStatus(200);
});

$app->run();
```

Hum.. dá pra melhorar..

```php
<?php

use Slim\App;
use TalkWhyTdd\Infrastructure\Middlewares\RequestBodyParserMiddleware;
use TalkWhyTdd\Infrastructure\Controllers\CheckUsernameController;

require __DIR__.'/../vendor/autoload.php';

$app = new Slim\App();

$app->add(RequestBodyParserMiddleware::class);

$app->post('/check-username', CheckUsernameController::class);

$app->run();
```

NORMAL web/index.php php 86% 13/15☰: 75

```php
<?php

namespace TalkWhyTdd\Infrastructure\Middlewares;

use Psr\Http\Message\ServerRequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;

class RequestBodyParserMiddleware
{
    public function __invoke(Request $request, Response $response, $next)
    {
        $requestRawBody = $request->getBody()->getContents();

        $parseToArray = true;
        $parsedBody = json_decode($requestRawBody, $parseToArray);

        $request = $request->withParsedBody($parsedBody);

        return $next($request, $response);
    }
}
```

<frastructure/Middlewares/RequestBodyParserMiddleware.php php 9% 2/21☰: 1

```php
<?php

namespace TalkWhyTdd\Infrastructure\Controllers;

use Psr\Http\Message\ServerRequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;

class CheckUsernameController
{
    public function __invoke(Request $request, Response $response)
    {
        $requestParsedBody = $request->getParsedBody();
        $username = $requestParsedBody['username'] ?? null;

        if (null === $username) {
            return $response
                ->withStatus(422)
                ->withJson(['message' => 'The field \'username\' is missing']);
        }

        return $response->withStatus(200);
    }
}
```

<c/Infrastructure/Controllers/CheckUsernameController.php php 8% 2/23☰: 1

Próximo caso: *username* com formato inválido

```
+ 29
+ 30  Scenario: The username has an invalid format
+ 31    When I request POST "/check-username" with the following body:
+ 32      """
+ 33      {
+ 34        "username": "-fefas"
+ 35      }
+ 36      """
+ 37    Then the response status code should be 422
+ 38    And the response body should be:
+ 39      """
+ 40      {
+ 41        "message": "The 'username' is not properly formatted"
+ 42      }
+ 43      """
```

```
10  public function __invoke(Request $request, Resp
11  {
12      $requestParsedBody = $request->getParsedBod
13      $username = $requestParsedBody['username']
14
15      if (null === $username) {
16          return $response
17              ->withStatus(422)
18              ->withJson(['message' => 'The field \'username\' is missing']);
19      }
20
+ 21      if (false === ctype_alnum($username)) {
+ 22          return $response
+ 23              ->withStatus(422)
+ 24              ->withJson(['message' => 'The \'username\' is not properly formatted']);
+ 25      }
+ 26
27      return $response->withStatus(200);
28  }
29 }
```

Primeiro teste unitário

```php
<?php

namespace TalkWhyTdd\User\Model;

use InvalidArgumentException;

class Username
{
    public function __construct(string $username)
    {
        if (false === $this->isFormatValid($username)) {
            throw new InvalidArgumentException('Invalid username provided');
        }
    }

    private function isFormatValid(string $username): bool
    {
        return ctype_alnum($username);
    }
}
```

NORMAL src/User/Model/Username.php php 5% 1/20 : 1

```php
<?php

namespace TalkWhyTdd\User\Model;

use PHPUnit\Framework\TestCase;

class UsernameTestTest extends TestCase
{
    /**
     * @test
     * @expectedException \InvalidArgumentException
     * @expectedExceptionMessage Invalid username provided
     * @dataProvider invalidUsernames
     */
    public function exceptionOccursIfFormatIsInvalid($invalidUsername)
    {
        new Username($invalidUsername);
    }

    public function invalidUsernames()
    {
        return [
            ['-fefas'],
            [' fefas'],
            ['fef@s'],
            ['14f&fas'],
            ['14fe_fas'],
        ];
    }
}
```

tests/unit/User/Model/UsernameTest.php php 3% 1/30

Voltando...

```php
<?php

namespace TalkWhyTdd\Infrastructure\Controllers;

use InvalidArgumentException;
use Psr\Http\Message\ServerRequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;
use TalkWhyTdd\User\Model\Username;

class CheckUsernameController
{
    public function __invoke(Request $request, Response $response)
    {
        $requestParsedBody = $request->getParsedBody();
        $username = $requestParsedBody['username'] ?? null;

        if (null === $username) {
            return $response
                ->withStatus(422)
                ->withJson(['message' => 'The field \'username\' is missing']);
        }

        try {
            new Username($username);
        } catch (InvalidArgumentException $e) {
            return $response
                ->withStatus(422)
                ->withJson(['message' => 'The \'username\' is not properly formatted']);
        }

        return $response->withStatus(200);
    }
}
```

E assim por diante...

TDD é direto ao ponto

Vantagens

- Ganho de objetividade
- Resultados mais simples
- Documentação como código
- Aumento de qualidade:
  - Código mais expressivo
  - Diminuição de acoplamento
  - Incentivo a injeção de dependências
- CI e CD mais seguros (como fazer sem?)

Acrônimo AAA

Arrange - garante estado necessário para exercício do teste

Act - exercitar o código a ser testado; (chamada do método)

Assert - verifica se o comportamento foi como esperado

```
Scenario: Retrieve a list of available areas and their codes
  Given the following areas were registered:
    | Area Code | Area Name |
    | 011       | São Paulo |
    | 017       | Mirassol  |
    | 019       | Campinas  |
  When I request "GET" "/areas"
  Then I should receive a response with the status code 200
  And the following JSON body:
    """
    [
      {
        "code": "011",
        "area": "São Paulo"
      },
      {
        "code": "017",
        "area": "Mirassol"
      },
      {
        "code": "019",
        "area": "Campinas"
      }
    ]
    """
```

Acrônimo FIRST

Fast	- quanto mais devagar, maior o risco
Isolated	- foco em pequena parte do código
Repeatable	- sempre o mesmo resultado; controle do entorno
Self-validating	- válidos por si só; sem interação manual
Timely	- não postergue; depois é nunca

Testes é um tema extenso

.. espero ter dado um gostinho

Referências

- Jeff Langr, 2015.
  Pragmatic Unit Testing with JUnit in Java 8

- Nat Pryce; Steve Freeman, 2009.
  Growing Object Oriented Software, Guided by Tests

Alguma dúvida?

Código: https://github.com/fefas/talk-2017-06-20-why-tdd

Felipe Martins
@fefas (ou @eufefas)
me@fefas.net
https://blog.fefas.net