# PHPunit 101

## Introdução aos testes

Olá! Vamos falar um pouco sobre testes?!

Muito têm se falado sobre Qualidade de Software, sobre Aplicações desacopladas e diversas assuntos relacionados a Design Orientado a Objetos.

Os testes se encaixam nesse panorama, eles visam garantir a qualidade do código que se está querendo entregar, com os testes você consegue mapear os bugs de sua aplicação e corrigí-los, documentado-os através da escrita de testes na sua linguagem escolhida.

Existem vários tipos de teste, vamos abordar aqui os Unitários através do PHPUnit e os funcionais, através de uma extensão do Symfony e que também usa o PHPUnit para tal.

## TDD

TDD ou Test Driven Development (Desenvolvimento Guiado Por Testes) é uma metodologia onde você escreve primeiro o teste e depois a implementação.

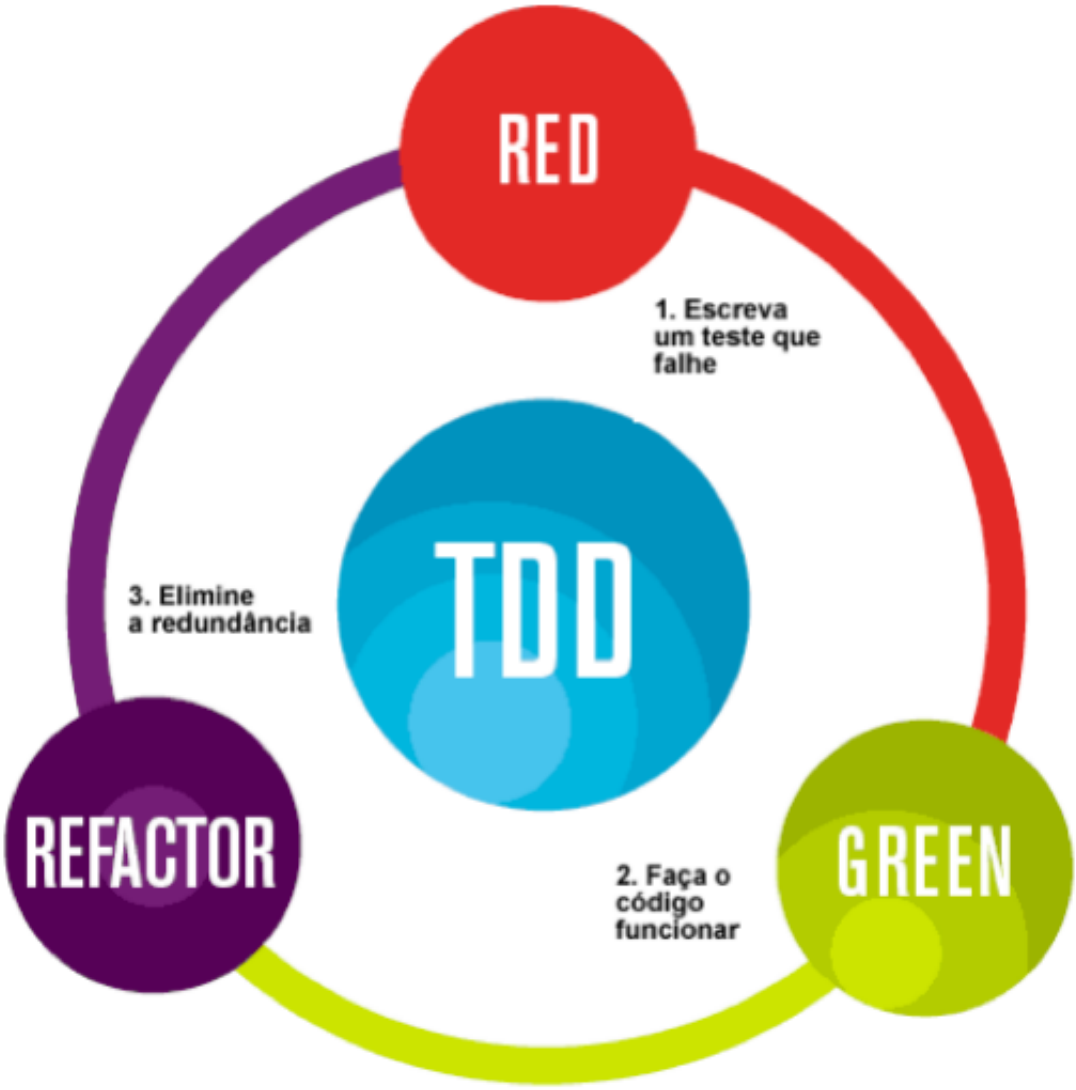
Explicando melhor:

A filosofia TDD, visa o planejamento antes para depois implementar. O planejamento engloba a escrita do teste ou seja, você pensa antes em como sua API vai funcionar, quando me refiro a API aqui! Falo da interface dos seus objetos, para depois implementar o comportamento de fato.

Isso te leva a um ciclo simples de TDD, que são:

- Escreva o teste (Ele vai falhar | Fica Vermelho);
- Crie o minimo, possivel, de implementação para o teste funcionar (Ele passa | Fica Verde);
- Refatore sua implementação com o foco em melhorar seu código, tendo sempre em mente que a refatoração têm que refletir sempre o green nos testes.

Vide imagem abaixo:




# O PHPUnit!

O PHPUnit é um Framework para escrita de testes unitários em PHP, criado por Sebastian Bergmann. Ele facilita muito a escrita de testes, através de N assertions disponíveis em seu core.

O PHPUnit ainda disponibilza uma interface cli, que nos permite executar os testes criados, de forma a facilitar nosso debug durante a execução e saber o status dos mesmos.

## Conhecendo as configurações

Vamos entender agora o nosso arquivo de configurações, comentando em noss último capítulo.

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit
        backupGlobals="false"
        backupStaticAttributes="false"
        bootstrap="./vendor/autoload.php"
        cacheTokens="true"
        colors="true"
        convertErrorsToExceptions="true"
        convertNoticesToExceptions="true"
        convertWarningsToExceptions="true"
        processIsolation="false"
        stopOnFailure="false"
        verbose="false">
    <testsuites>
        <testsuite name="GetStarted Tests">
            <directory>./tests/</directory>
        </testsuite>
    </testsuites>
</phpunit>
```

Da linha 3 a 13 temos as configurações setadas para utilização na execução de nossa suíte de testes. Explicando cada linha, temos o seguinte:

**backupGlobals="false"**

Realiza o backup e a restauração de varáveis globais durante a execução dos testes, caso `true`

**backupStaticAttributes="false"**

Copia todas os valores de atributos estáticos e os restaura posteriormente pro caso de está setado como `true`

**bootstrap="./vendor/autoload.php"**

Define o bootstrap de nossa aplicação, onde colocamos nosso `autoload` dentre outras configs a nivel de aplicação, como set de Timezone por exemplo.

**cacheTokens="true"**

Quando `true`, faz o cache de tokens php através do pacote PHP-Token-Stream. Esse pacote é um wrapper da extensão PHP Tokenizer, mais info aqui.

**colors="true"**

Utiliza cores para os outputs da interface cli.

**convertErrorsToExceptions="true"**

Converte os erros do PHP, para Exceptions. Segue a lista de Constant Errors que ele converte:

- E_WARNING
- E_NOTICE
- E_USER_ERROR
- E_USER_WARNING
- E_USER_NOTICE
- E_STRICT
- E_RECOVERABLE_ERROR
- E_DEPRECATED
- E_USER_DEPRECATED

Mais sobre as constant errors no PHP, veja aqui.

**convertNoticesToExceptions="true"**

Quando setado para false, não converte o E_NOTICE, E_USER_NOTICE e E_STRICT para exception.

**convertWarningsToExceptions="true"**

Quando setado para false, não converte o E_WARNING e E_USER_WARNING para exception.

**processIsolation="false"**

Se setado como `true`, executa cada teste em um processo php diferente.

**stopOnFailure="false"**

Se setado como `true`, para a execução dos testes no primeiro erro ou falha.

**verbose="false"**

Na execução dos testes, se setado como `true`, exibe informações mais detalhadas na cli.

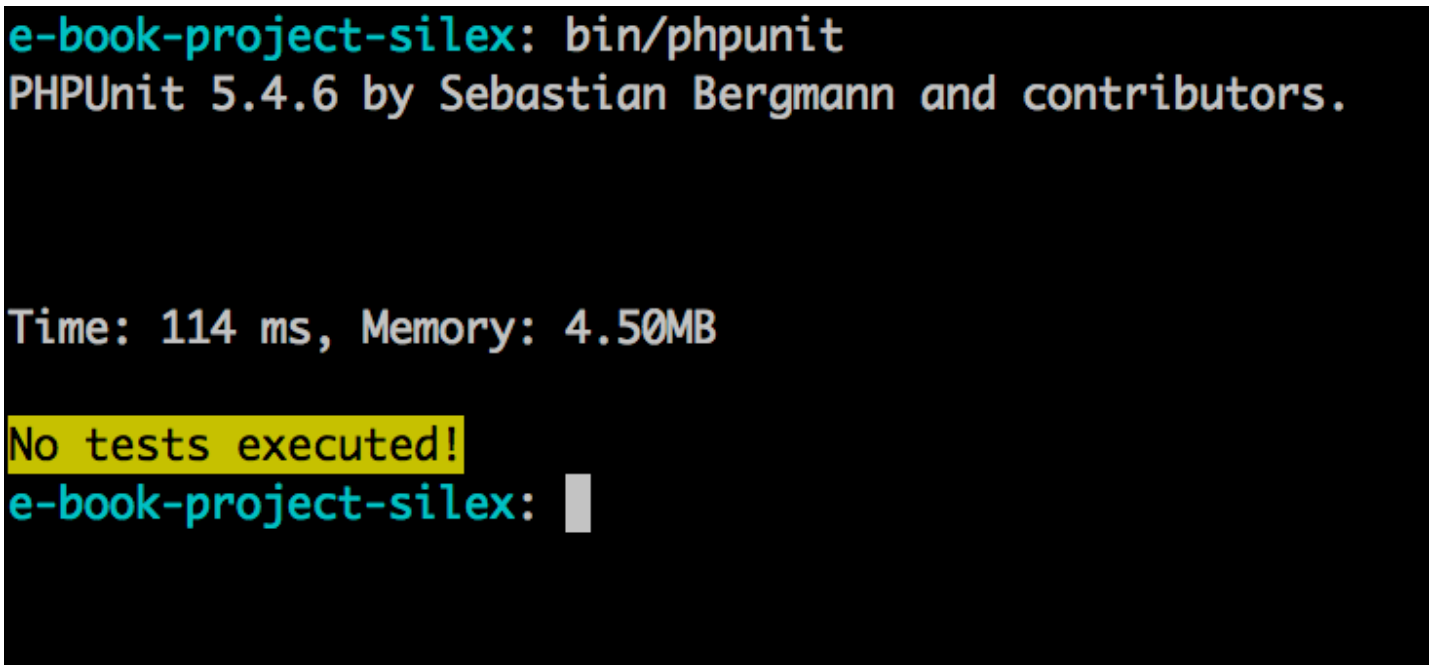
**Linhas 14 a 18**

Definimos o folder onde o PHPUnit irá buscar nossas classes de teste. Que devem possuir o sufixo `Test` nos nomes dos arquivos e classes php, e cada método dessas classes devem possuir o prefixo `test` para que o PHPUnit indentifique os testes a serem executados de cada classe! Ufa!

Veremos mais a fundo isso na prática!

## Executando nosso primeiro teste

Com nosso arquivo `phpunit.xml.dist` na raiz do nosso projeto e com as dependências devidamente instaladas. Podemos executar um `bin/phpunit` na raiz do projeto. Por hora teremos o resultado da imagem abaixo:




Por hora não temos nenhum teste criado, veremos isso na prática em nossos próximos capítulos.

## Conclusão

Essa introdução a testes nos deu uma base para nossas próximas atividades, atividades relacionadas a criação de nossa API de Eventos. Nos aprofundaremos ainda mais nesse assunto durante a criação de cada parte integrante do nosso projeto proposto.

Nos vemos nos próximos capítulos! Até lá!