



www.devmedia.com.br

[versão para impressão]

Link original: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=33568>

PHP TDD: Desenvolvimento Guiado a Testes

PHP TDD, veja nesse artigo como desenvolver aplicações guiadas a testes utilizando TDD e PHP. Confira!

Você já se perguntou o porquê de testar um software. Se não testarmos, o software funciona? Por que testar? Dentre algumas metodologias ágeis, o **TDD** tem conseguido espaço em projetos, principalmente os de grande porte, visando trazer qualidade ao software.

Mas, por que TDD? O **TDD** é mais do que testar código: é uma mudança cultural que propõe uma nova cultura no padrão de desenvolvimento de software. Entenda alguns pontos fortes da metodologia:

- Software escalável e flexível;
- Prevenção de erros desde o início;
- Qualidade agregada ao produto final;
- Confiança do código;
- Facilita a documentação.

Mas se testar é importante, por que nem sempre testamos? Porque testar tem um custo. Muitas vezes achamos que o esforço de contratar um profissional com conhecimentos em testes de software é um custo desnecessário. Por outro lado, o custo de contratação de um novo profissional no orçamento de uma empresa geralmente é menor do que o custo de reparar possíveis bugs causados por falhas de software. Quando testamos, descobrimos erros precocemente. Essa é uma das vantagens de usar uma metodologia de testes.

Vamos analisar melhor uma aplicação muito comum. Suponhamos um software para uma instituição de ensino, e que dentro do nosso projeto de software existe uma classe `aluno.php` que serve como base para criação de nossos alunos. A **Listagem 1** mostra o modelo da nossa classe de maneira resumida.

Listagem 1. Classe Aluno.

```
<?php
class Aluno {
    private $nome;
    private $turma;
    private $idade;
    private $notaGeral;
    private $matricula;
```

```

        //Construtor da classe
        public function __construct($nome, $turma, $idade, $matricula) {
            $this->nome = $nome;
            $this->turma = $turma;
            $this->idade = $idade;
            $this->matricula = $matricula;
        }

        public function atribuirNota($n1,$n2,$n3) {
            $notas = array();
            array_push($notas, $n1,$n2,$n3);
            $this->notaGeral = array_sum($notas);
        }

        public function getTurma() {
            return $this->turma;
        }

        public function getNotaGeral() {
            return $this->notaGeral;
        }

        public function getMatricula() {
            return $this->matricula;
        }
    }
    ?>

```

Quando nos referimos a testes em um projeto Orientado a objetos, a função do teste de unidade é testar uma pequena parte do projeto, ou seja, a menor unidade que pode ser testada é a classe. Então deverá existir um teste de unidade para cada classe existente.

Para nosso arquivo aluno.php vamos criar um arquivo de teste com a classe AlunoTest. A base da nossa classe teste ficaria conforme a **Listagem 2**.

Listagem 2. Classe de teste AlunoTest.

```

<?php

require_once '../src/Application/Exemplo/aluno.php';
class AlunoTest extends Aluno{
    public function testarNumeroMatricula() { }
    public function testarTurma() { }
    public function testarNota() { }
}
?>

```

Cada método que necessita ser implementado é responsável por testar uma função da classe Aluno em aluno.php. O trabalho é exaustivo, pois primeiro teríamos que instanciar objetos da classe Aluno em nossa classe filha Alunotest, e só assim, testar as possibilidades de implementação, testando possíveis dados corretos e dados que causariam falha ao código.

O PHPUnit poupa bastante trabalho quando se trata de testes, devido à uma grande quantidade de métodos já implementados que podem ser usados, poupando boa parte da implementação.

Vejamos como pode ser feita a aplicação do TDD em um projeto de PHP. Cada funcionalidade é iniciada com a criação de um teste, ou seja, em um projeto TDD devemos criar a classe de teste antes da própria classe que será testada.

Foi mostrado no código anterior como podemos criar e definir uma classe de teste funcional, responsável por uma pequena parte do código. Você pode pensar que é fácil e rápido. Pode até ser feito de forma rápida em um pequeno projeto, com poucas classes implementadas. Mas imagine um grande projeto com centenas ou até milhares de classes. É inviável criar testes manualmente. É importante que estes sejam automatizados, pois a automação dos processos do TDD gera códigos de testes de maior qualidade e menor probabilidade de erros.

Os processos de teste automatizados são semelhantes aos manuais. Precisamos ter uma situação para executá-los e recebermos uma validação.

Ambiente ->Execução->Validação

Já que falamos de automação, vamos então repensar nossa maneira de testar nossas classes, especificamente usando a linguagem PHP.

Como exemplo, será utilizado o framework de testes PHPUnit. E para instalá-lo vamos precisar de alguns requisitos (saiba que existe mais de uma maneira de instalação, mas no nosso projeto instalaremos através do Composer. Mais detalhes poderão ser conferidos na documentação oficial do PHPUnit (seção **Links**)):

- Composer (Gerenciamento de dependências);
- PHP (Instalar a última versão do **PHP** é altamente recomendável);
- Suporte a extensões .json, que normalmente estão habilitadas por padrão;

Como exemplo, a IDE usada será a Sublime Text 3.

Instalando o PHPUnit

Existe mais de uma forma de instalar o PHPUnit. No site oficial tem informações de como instalar usando o Composer ou através do download do arquivo PHAR (PHP Archive).

Faremos a instalação através do Composer, que é o gerenciador de dependências do PHP. Antes de instalar o PHPUnit, a primeira tarefa que deve ser feita é criar um diretório no qual vamos trabalhar. Dentro deste diretório deverá conter o arquivo "composer.json", onde deverão estar declaradas as bibliotecas utilizadas pelo PHPUnit. O arquivo funcionará basicamente como auxiliar e gerenciará todas as dependências do projeto. Tudo que estiver no repositório do Composer será gerenciado conforme as especificações do arquivo.

Dentro deste arquivo, obrigatoriamente, deverá conter o conteúdo de acordo com a **Listagem 3**.

Listagem 3. Arquivo composer.json

```
{
  "require-dev" : {
    "phpunit/phpunit" : "*"
  }
}
```

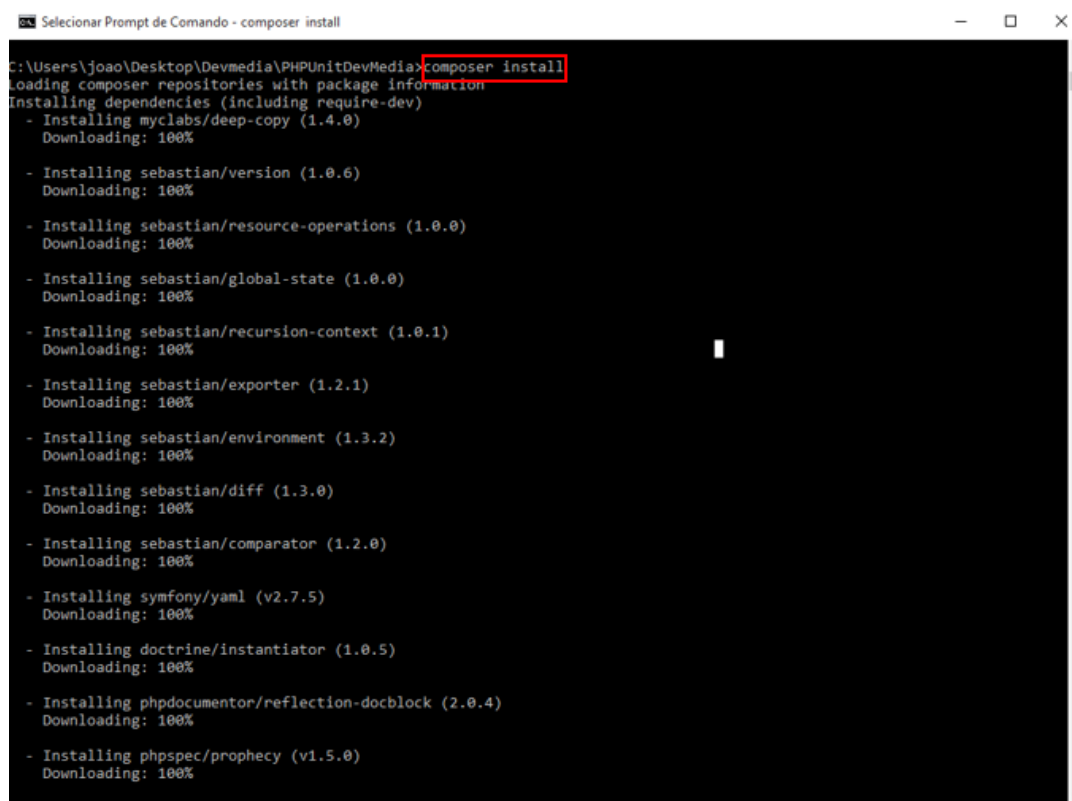
Veja que estamos importando o PHPUnit na sua versão mais atual. Caso queira utilizar alguma versão específica, basta acrescentar o número da versão antes de "*", dentro das aspas.

Assim a declaração das dependências que serão usadas no projeto está pronta.

O próximo passo é instalar o Composer no diretório do projeto. Para isso, utilizando a linha de comando, navegue até o diretório onde se encontra o arquivo composer.json e utilize o comando a seguir:

```
Composer install
```

Veja na **Figura 1** o processo de instalação.



```
Selecionar Prompt de Comando - composer install
C:\Users\joao\Desktop\Devmedia\PHPUnitDevMedia>composer install
Loading composer repositories with package information
Installing dependencies (including require-dev)
- Installing myclabs/deep-copy (1.4.0)
  Downloading: 100%
- Installing sebastian/version (1.0.6)
  Downloading: 100%
- Installing sebastian/resource-operations (1.0.0)
  Downloading: 100%
- Installing sebastian/global-state (1.0.0)
  Downloading: 100%
- Installing sebastian/recursion-context (1.0.1)
  Downloading: 100%
- Installing sebastian/exporter (1.2.1)
  Downloading: 100%
- Installing sebastian/environment (1.3.2)
  Downloading: 100%
- Installing sebastian/diff (1.3.0)
  Downloading: 100%
- Installing sebastian/comparator (1.2.0)
  Downloading: 100%
- Installing symfony/yaml (v2.7.5)
  Downloading: 100%
- Installing doctrine/instantiator (1.0.5)
  Downloading: 100%
- Installing phpdocumentor/reflection-docblock (2.0.4)
  Downloading: 100%
- Installing phpspec/prophecy (v1.5.0)
  Downloading: 100%
```

Figura 1. Instalação do composer via Prompt de comando

Você perceberá que o Composer iniciará a instalação, inclusive do PHPUnit, conforme mostra a **Figura 2**.

```

Prompt de Comando

- Installing phpunit/php-token-stream (1.4.8)
  Downloading: 100%

- Installing phpunit/php-file-iterator (1.4.1)
  Downloading: 100%

- Installing phpunit/php-code-coverage (3.0.0)
  Downloading: 100%

- Installing phpunit/phpunit (5.0.3)
  Downloading: 100%

sebastian/global-state suggests installing ext-uopz (*)
phpdocumentor/reflection-docblock suggests installing dflydev/markdown (~1.0)
phpdocumentor/reflection-docblock suggests installing erusev/parsedown (~1.0)
phpunit/php-code-coverage suggests installing ext-xdebug (>=2.2.1)
phpunit/phpunit suggests installing phpunit/php-invoker (~1.1)
Writing lock file
Generating autoload files

C:\Users\joao\Desktop\Devmedia\PHPUnitDevMedia>
C:\Users\joao\Desktop\Devmedia\PHPUnitDevMedia>
C:\Users\joao\Desktop\Devmedia\PHPUnitDevMedia>
C:\Users\joao\Desktop\Devmedia\PHPUnitDevMedia>

```

Figura 2. Instalação do PHPUnit com composer

Perceba que foi feito o download do framework conforme a especificação do arquivo composer.json.

Após o término da instalação serão criadas algumas pastas no nosso diretório. A estrutura de pastas de instalação ficará conforme a **Figura 3**.

| Nome | Data de modificaç... | Tipo | Tamanho |
|---------------|----------------------|-------------------|---------|
| vendor | 07/10/2015 05:26 | Pasta de arquivos | |
| composer.json | 07/10/2015 04:07 | Arquivo JSON | 1 KB |
| composer.lock | 07/10/2015 05:26 | Arquivo LOCK | 36 KB |

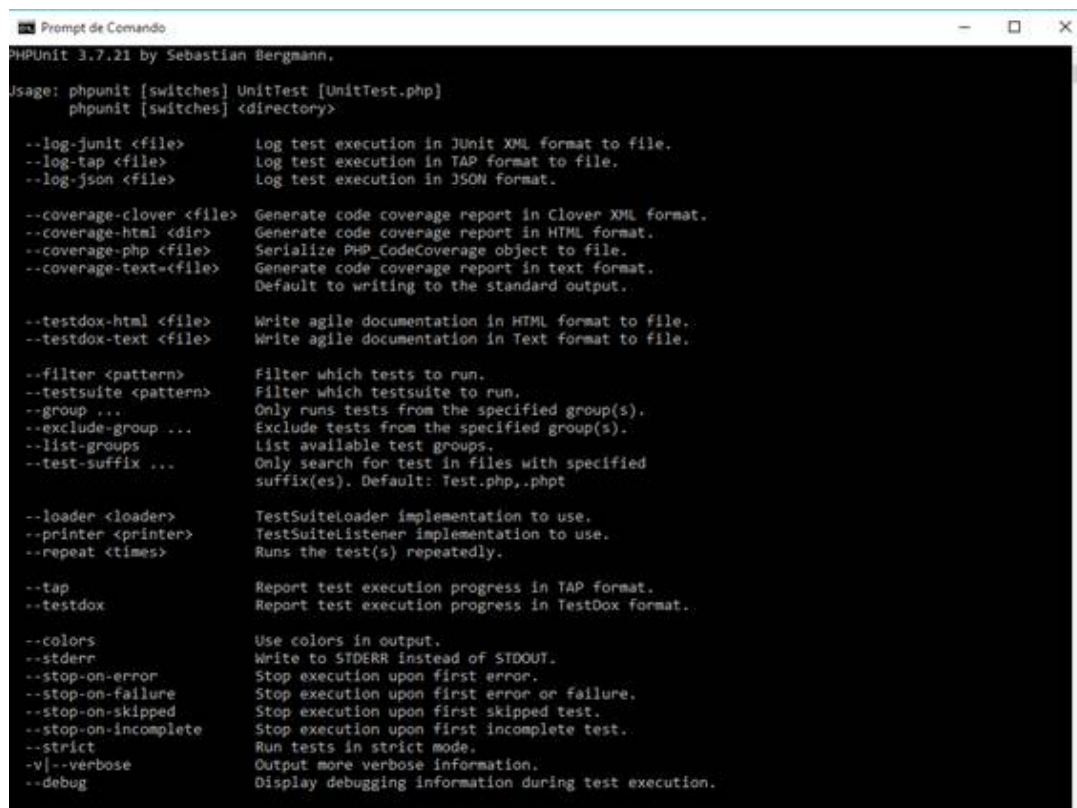
| Nome | Data de modificaç... | Tipo | Tamanho |
|---------------|----------------------|-------------------|---------|
| bin | 07/10/2015 05:26 | Pasta de arquivos | |
| composer | 07/10/2015 05:26 | Pasta de arquivos | |
| doctrine | 07/10/2015 04:09 | Pasta de arquivos | |
| myclabs | 07/10/2015 04:08 | Pasta de arquivos | |
| phpdocumentor | 07/10/2015 04:09 | Pasta de arquivos | |
| phpspec | 07/10/2015 04:09 | Pasta de arquivos | |
| phpunit | 07/10/2015 05:25 | Pasta de arquivos | |
| sebastian | 07/10/2015 04:09 | Pasta de arquivos | |
| symfony | 07/10/2015 04:09 | Pasta de arquivos | |
| autoload.php | 07/10/2015 05:26 | Arquivo PHP | 1 KB |

Figura 3. Diretório de instalação do composer

Na pasta vendor é onde estão as bibliotecas de terceiros (incluindo o PHPUnit) que poderão ser usadas no projeto. Além disso, na pasta raiz aparecerá um arquivo, o composer.lock, que armazena as versões exatas das bibliotecas assim como quaisquer alterações feitas.

Conhecendo o PHPUnit

Após a instalação, vamos testar se tudo está conforme o esperado. Na linha de comando digite "phpunit". Se a instalação foi bem-sucedida, o resultado será uma espécie de helper com comandos válidos de acordo com a **Figura 4**.



```
Prompt de Comando
PHPUnit 3.7.21 by Sebastian Bergmann.

Usage: phpunit [switches] UnitTest [UnitTest.php]
       phpunit [switches] <directory>

--log-junit <file>          Log test execution in JUnit XML format to file.
--log-tap <file>            Log test execution in TAP format to file.
--log-json <file>           Log test execution in JSON format.

--coverage-clover <file>     Generate code coverage report in Clover XML format.
--coverage-html <dir>        Generate code coverage report in HTML format.
--coverage-php <file>        Serialize PHP_CodeCoverage object to file.
--coverage-text <file>       Generate code coverage report in text format.
                             Default to writing to the standard output.

--testdox-html <file>        Write agile documentation in HTML format to file.
--testdox-text <file>        Write agile documentation in Text format to file.

--filter <pattern>           Filter which tests to run.
--testsuite <pattern>        Filter which testsuite to run.
--group ...                  Only runs tests from the specified group(s).
--exclude-group ...          Exclude tests from the specified group(s).
--list-groups                List available test groups.
--test-suffix ...            Only search for test in files with specified
                             suffix(es). Default: Test.php,.php*

--loader <loader>            TestSuiteLoader implementation to use.
--printer <printer>          TestSuiteListener implementation to use.
--repeat <times>             Runs the test(s) repeatedly.

--tap                        Report test execution progress in TAP format.
--testdox                    Report test execution progress in TestDox format.

--colors                     Use colors in output.
--stderr                     Write to STDERR instead of STDOUT.
--stop-on-error              Stop execution upon first error.
--stop-on-failure            Stop execution upon first error or failure.
--stop-on-skipped            Stop execution upon first skipped test.
--stop-on-incomplete        Stop execution upon first incomplete test.
--strict                     Run tests in strict mode.
-v|-v|--verbose              Output more verbose information.
--debug                     Display debugging information during test execution.
```

Figura 4. Helper PHPUnit

Todos esses comandos serão válidos no nosso console sempre que utilizarmos o phpunit.

Além dos testes convencionais, é possível testar interações com bancos de dados. Para isso existe o DbUnit, que pode ser instalado via composer adicionando no arquivo composer.json a dependência:

```
"phpunit/dbunit": ">=1.2"
```

Asserções no PHPUnit3

Asserções são funções do PHPUnit já implementadas que servem para validações. Elas asseguram que uma saída terá o valor esperado. Existem asserções para testar tipos primitivos ou até mesmo objetos e comunicações com bancos de dados.

Normalmente o método de asserção usado com bastante frequência é o assertEquals().

Iniciando com PHPUnit

Com o PHPUnit instalado crie os subdiretórios do projeto de modo que sejam iguais. Isso é importante para facilitar e separar a responsabilidade de cada diretório.

A **Figura 5** mostra como exemplo uma estrutura de diretórios, diferenciando a pasta de código de produção e de testes, onde na pasta "src" temos nosso código em produção, e na pasta "tests" temos nossos arquivos de testes unitários.

| Nome | Data de modificaç... | Tipo | Tamanho |
|---------------|----------------------|-------------------|---------|
| vendor | 07/10/2015 05:26 | Pasta de arquivos | |
| composer.json | 07/10/2015 04:07 | Arquivo JSON | 1 KB |
| composer.lock | 07/10/2015 05:26 | Arquivo LOCK | 36 KB |

| Nome | Data de modificaç... | Tipo | Tamanho |
|---------------|----------------------|-------------------|---------|
| bin | 07/10/2015 05:26 | Pasta de arquivos | |
| composer | 07/10/2015 05:26 | Pasta de arquivos | |
| doctrine | 07/10/2015 04:09 | Pasta de arquivos | |
| myclabs | 07/10/2015 04:08 | Pasta de arquivos | |
| phpdocumentor | 07/10/2015 04:09 | Pasta de arquivos | |
| phpspec | 07/10/2015 04:09 | Pasta de arquivos | |
| phpunit | 07/10/2015 05:25 | Pasta de arquivos | |
| sebastian | 07/10/2015 04:09 | Pasta de arquivos | |
| symfony | 07/10/2015 04:09 | Pasta de arquivos | |
| autoload.php | 07/10/2015 05:26 | Arquivo PHP | 1 KB |

Figura 5. Exemplo de diretório

Agora, para iniciarmos o trabalho com o PHPUnit é interessante trabalharmos com autoloader, além de ser um padrão do Framework Interop Group (FIG).

Então adicionaremos algumas linhas no arquivo composer.json, que ficará com a seguinte estrutura de acordo com a **Figura 6**.



Figura 6. Arquivo composer.json

Agora vamos praticar os nossos testes e o ciclo do TDD.

O seu funcionamento é dividido em três estados: Red, Green, refactor, como de acordo com a **Figura 7**.

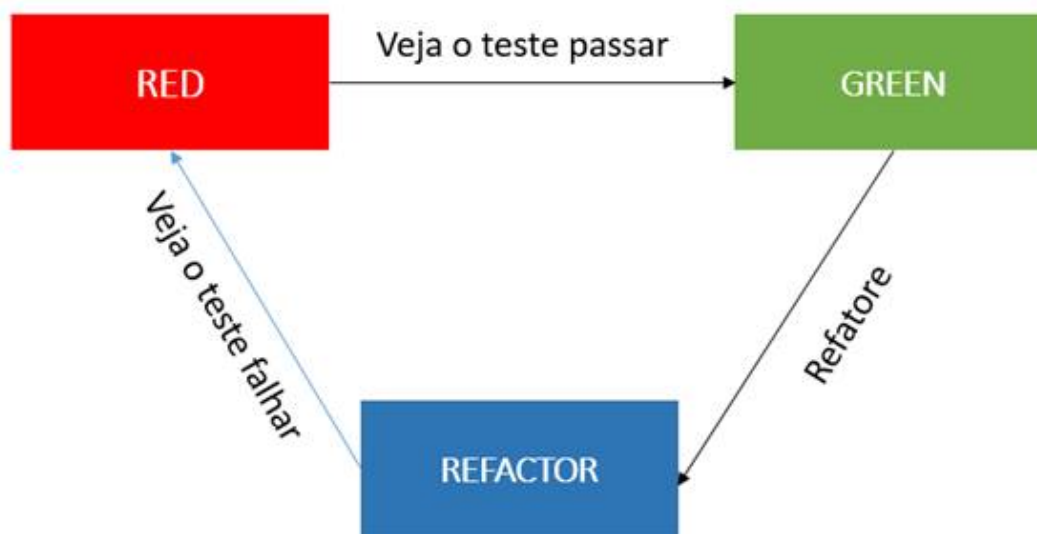


Figura 7. Ciclo do TDD

Primeiro escreva um teste que falhe. Depois escreva um que funcione da maneira mais simples e, por fim, refatore o código. Geralmente esse ciclo é feito uma vez para cada teste unitário.

Vamos ao nosso exemplo aluno.php, mas desta vez, vamos testar nossa classe usando o PHPUnit. A nossa classe de teste foi modelada conforme a **Listagem 4**.

Listagem 4. Classe alunotest a ser modificada


```

require_once '../src/Application/Exemplo/aluno.php';
class AlunoTest extends Aluno{
    public function testarNumeroMatricula() { }
    public function testarTurma() { }
    public function testarNota() { }
}
?>

```

Essa classe merece ser reescrita de acordo com o uso do PHPUnit. Vamos reescrevê-la conforme a **Listagem 5**.

Listagem 5. Classe alunoTest remodelada

```

<?php

use PHPUnit_Framework_TestCase as PHPUnit;

require_once '../src/Application/Exemplo/aluno.php';

class AlunoTest extends PHPUnit{

    //Função de teste de tipo - Compara tipo da variável Matricula
    public function testType() {
        $aluno = new Aluno("Jose", "B", 12, 001);
        $this->assertInternalType('int', $aluno->getMatricula());
    }

    //Função de teste de número de matricula - Verifica numero de alunos cadastrados
    public function testNumeroMatricula() {
        $aluno = new Aluno("Jose", "B", 12, 001);
        $this->assertEquals(001, $aluno->getMatricula());
    }

    //Função de teste que verifica turma do aluno
    public function testTurma() {
        $aluno = new Aluno("Maria", "C", 10, 002);
        $this->assertEquals("C", $aluno->getTurma());
    }

    //Função que compara nota do aluno atual com nota esperada
    public function testNota() {
        $aluno = new Aluno("Joao", "C", 10, 002);
        $aluno->atribuirNota(10,7,9);
        $this->assertEquals(25, $aluno->getNotaGeral());
    }
}
?>

```

Perceba no código apresentado que foram usadas algumas convenções do framework. O nome da classe está como AlunoTest, que é uma convenção do PHPUnit, onde uma classe de teste, para ser reconhecida como classe uma, deve possuir o sufixo "Test". Além disso, está herdando da classe PHPUnit_Framework_TestCase. Isso é necessário para que se possa trabalhar com os métodos já implementados pelo framework. No caso dos métodos, deve ser usada a palavra "test" como prefixo, ou seja, antes do nome do método.

Perceba também que na maioria dos nossos métodos foi utilizado a função assertEquals(), com dois parâmetros:

```
assertEquals(x,y);
```

Onde x é o resultado esperado para a função e y é o resultado corrente.

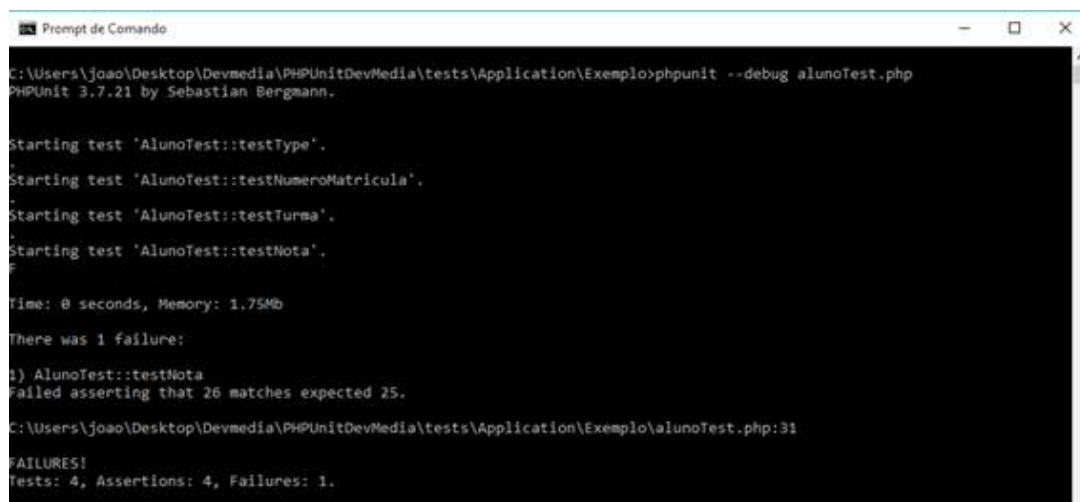
Visualizando resultados

Como a instalação foi feita via composer, vamos executar os testes através do prompt de comando.

No diretório do arquivo alunoTest.php, vamos executar o seguinte comando:

```
Phpunit --debug alunoTest.php
```

Isso fará com que o teste seja executado. Veja a execução na **Figura 8**.



```
Prompt de Comando
C:\Users\joao\Desktop\Devmedia\PHPUnitDevMedia\tests\Application\Exemplo>phpunit --debug alunoTest.php
PHPUnit 3.7.21 by Sebastian Bergmann.

Starting test 'AlunoTest::testType'.
Starting test 'AlunoTest::testNumeroMatricula'.
Starting test 'AlunoTest::testTurma'.
Starting test 'AlunoTest::testNota'.
F

Time: 0 seconds, Memory: 1.75Mb

There was 1 failure:

1) AlunoTest::testNota
Failed asserting that 26 matches expected 25.

C:\Users\joao\Desktop\Devmedia\PHPUnitDevMedia\tests\Application\Exemplo\alunoTest.php:31

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

Figura 8. Execução dos testes através da linha de comando

Veja que logo após "phpunit" existe o comando "debug", que habilita o modo debug para detalhamento das ações que estão sendo tomadas.

Veja também que o teste apresentou uma falha no método testNota.

A explicação da falha foi: Failed asserting that 26 matches expected 25.

Vamos verificar esta parte do código na **Listagem 6**.

Listagem 6. Função testNota

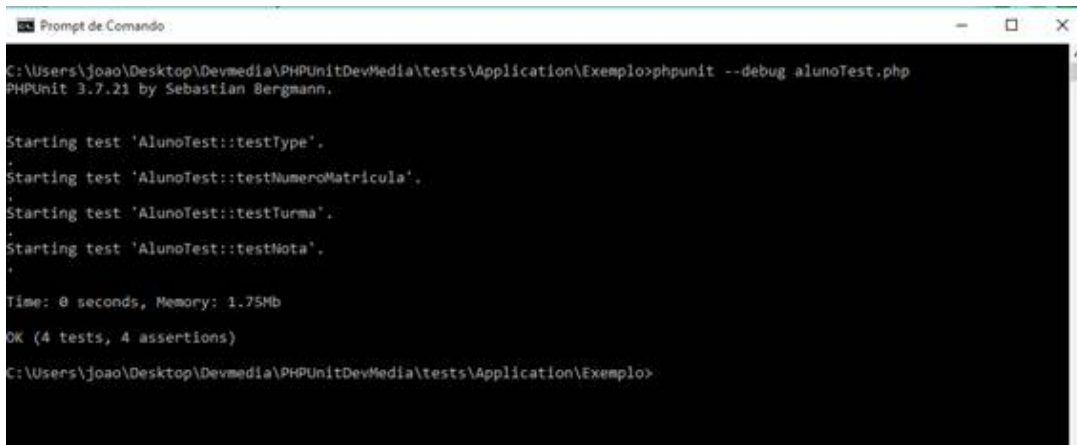
```
public function testNota() {
    $aluno = new Aluno("Joao", "C", 10, 002);
    $aluno->atribuirNota(10,7,9);
    $this->assertEquals(25, $aluno->getNotaGeral());
}
```

A função assertEquals compara dois resultados, neste caso estamos passando como parâmetro o número 25 como esperado para a operação, quando na verdade a soma das notas passadas na chamada do método vai gerar o resultado 26 (10+7+9 = 26).

Veja na **Listagem 7** a correção da função e vamos ver os resultados na **Figura 9**.

Listagem 7. Função testNota corrigida

```
public function testNota() {  
    $aluno = new Aluno("Joao", "C", 10, 002);  
    $aluno->atribuirNota(10,7,9);  
    $this->assertEquals(26, $aluno->getNotaGeral());  
}
```

**Figura 9.** Verificação de resultados do PHPUnit

Note que após a verificação de cada método dentro da classe de teste, o PHPUnit retornou a mensagem "OK (4 tests, 4 assertions)". Isso significa que todos os testes passaram sem erros ou falhas.

Além disso, o PHPUnit nos retornou o tempo de execução e a memória alocada.

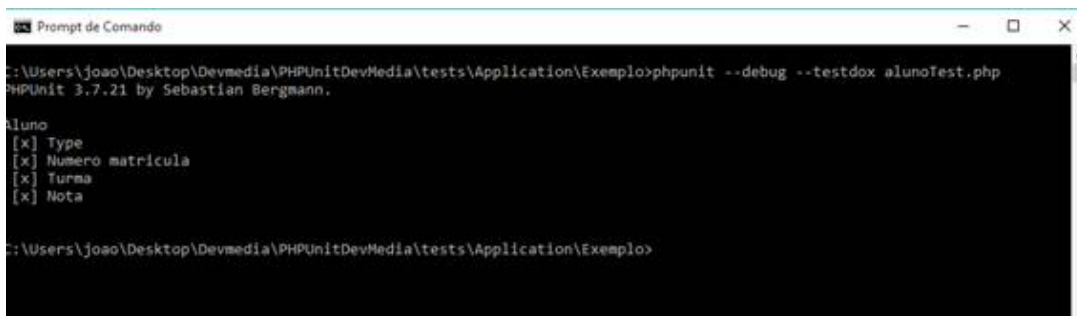
Manipulando dados no prompt de comando

Existem alguns comandos que podem ser usados no console no momento de gerar o output com PHPUnit que podem ser úteis no entendimento dos dados gerados.

Vejamos alguns deles:

```
Phpunit testdox arquivoTest.php
```

O código imprime na tela um checkbox com os métodos implementados na classe de teste. Os testes que passaram recebem um "x" e os que falharam apresentam um campo em branco, como mostra a **Figura 10**.

**Figura 10.** Testdox

```
Phpunit -tap arquivoTest.php
```

O comando `--tap` imprime os métodos e os relacionam com sua classe de forma numerada, como na **Figura 11**.



Figura 11. Comando `-tap` do PHPUnit

Outro comando que também ajuda a visualizar a informação é o seguinte:

```
Phpunit -colors
```

Este último apenas imprime a informação com cores: cor vermelha para testes que falharam e cor verde para os bem-sucedidos.

Executando uma suíte de testes

Além de executar testes um a um na linha de comando, o PHPUnit possui uma funcionalidade chamada de `testsuite`, que é usada para executarmos vários deles ao mesmo tempo.

O `testsuite` pode ser executado de duas maneiras:

1. Essa é a forma mais simples: pela linha de comando é preciso apontar o executor de testes para o diretório que contém os arquivos a serem testados, como mostra o exemplo a seguir:

```
Phpunit --bootstrap vendor/autoload.php tests
```

O comando `--bootstrap` está apontando o executor do PHPUnit para o diretório `tests`, que contém todos os arquivos de teste, assim todos os arquivos serão testados em lote

2. A outra forma é através de um arquivo XML, como mostra o exemplo da **Listagem 8**.

Listagem 8. Testsuites com XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<phpunit>
  <testsuites>
    <testsuite name="Exemplo de suíte de testes">
      <directory> ./tests/</directory>
    </testsuite>
  </testsuites>
</phpunit>
```

O arquivo XML passa como parâmetro o diretório dentro das tags `<directory>`. Isso significa que todos os arquivos que estiverem dentro deste diretório serão testados.

Para executarmos o arquivo XML no terminal, basta digitarmos o seguinte comando:

```
Phpunit -configuration diretorio/arquivo.xml
```

Caso queira executar arquivos específicos dentro da testsuite, basta incluir no arquivo xml a tag `<file>`, passando como parâmetro o diretório completo do arquivo, acompanhado de sua extensão.php, como mostra a **Listagem 9**.

Listagem 9. Testsuites com arquivos específicos

```
<?xml version="1.0" encoding="UTF-8" ?>
<phpunit>
  <testsuites>
    <testsuite name="Exemplo de suite de testes">
      <file> ./tests/Application/Exemplo/alunoTest.php</file>
    </testsuite>
  </testsuites>
</phpunit>
```

Perceba que as tags `<directory>` foram retiradas. Em contrapartida, foram adicionadas as tags `<file>`.

Pulando arquivos da suite de testes

Se por algum motivo quisermos que determinado arquivo dentro de nosso diretório não seja testado, basta usarmos a tag `<exclude>`, passando o diretório completo do arquivo .php, como mostra o exemplo da

Listagem 10.

Listagem 10. Pulando arquivos

```
<?xml version="1.0" encoding="UTF-8" ?>
<phpunit>
<testsuites>
  <testsuite name="Exemplo de suite de testes">
    <directory> ./tests/ </directory>
    <exclude> ./tests/Application/Exemplo/alunoDAOTest.php</exclude>
  </testsuite>
</testsuites>
</phpunit>
```

Com isso, o arquivo passado entre as tags `<exclude>` `</exclude>` será mantido no diretório, mas não será incluído nos testes.

O objetivo deste artigo não foi ensinar PHP, mas apenas dar uma breve introdução ao TDD e ao PHPUnit, que hoje é o framework mais popular para testes em PHP.

A metodologia baseada em testes muda a maneira de como desenvolvemos software. Apesar de uma mudança cultural nos processos do desenvolvimento padrão, testar o código torna o software mais confiável, flexível, escalável e garante a conformidade com os requisitos.

Links

Documentação PHPUnit

https://phpunit.de/manual/current/pt_br/



João Otávio
