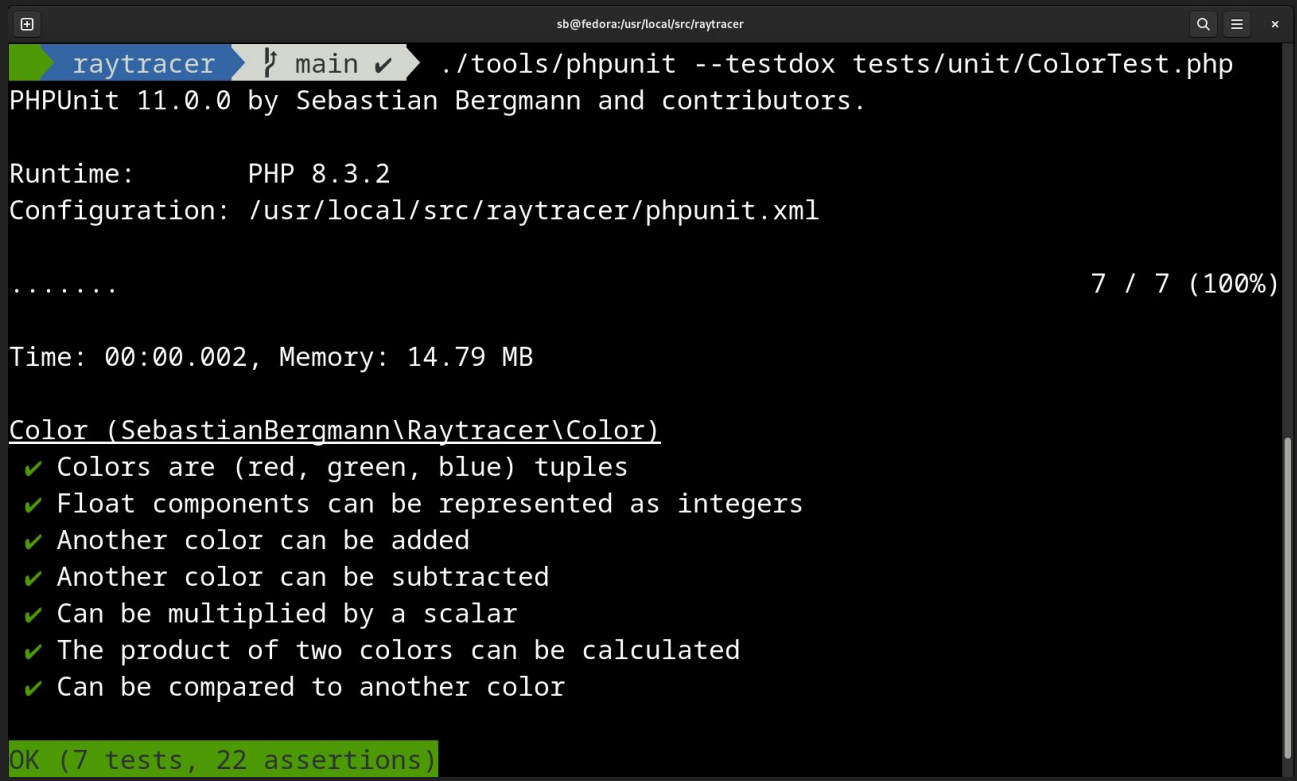
Next-Gen PHP

Aula 03

Testes para maior assertividade

Pra vocês quais as vantagens de se ter testes?



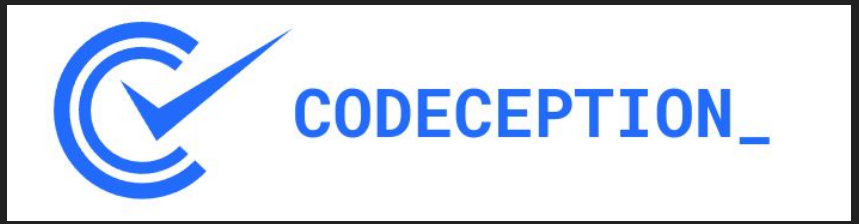
As dores

1. Não saber como funciona o código.
2. Não garantir o que já funciona.
3. Pensar em futuros problemas!
4. Complexidade em refatorar.
5. Problemas de Design!
6. Insegurança no deploy.

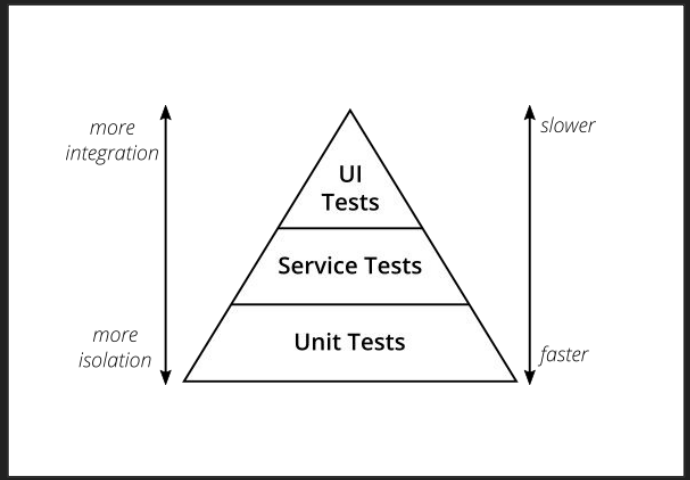
Minha escola de aprendizado

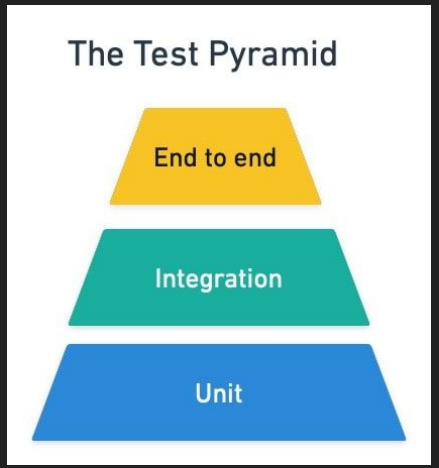
Kent Beck - Test-driven Development
Alistair Cockburn - Arquitetura Hexagonal
Eric Evans - Domain-driven Design
Craig Larman - LeSS e UML
Martin Fowler - DSL e Padrões mais uma série de temas
James Grenning - Manifesto Ágil e TDD com C
Ron Jeffries - Um dos criadores do XP e Agile
Ken Schwaber - Um dos criadores do Scrum
Lisa Crispin - Manifesto Ágil e Agile testing
Jim Highsmith - Auto do Adaptive Software Development
Robert C. Martin (Uncle Bob) - Clean Code/Architecture e mais

Ferramentas para testes no PHP



A pirâmide de testes





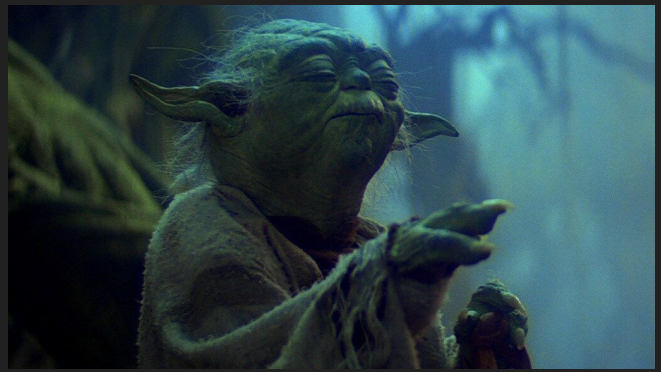
Artigo: https://martinfowler.com/articles/practical-test-pyramid.html

O que Testar?

TUDO, menos o código dos outros (vendors).

Ou

“Testar de terceiros códigos, você deve nunca!” - Yoda, mestre…

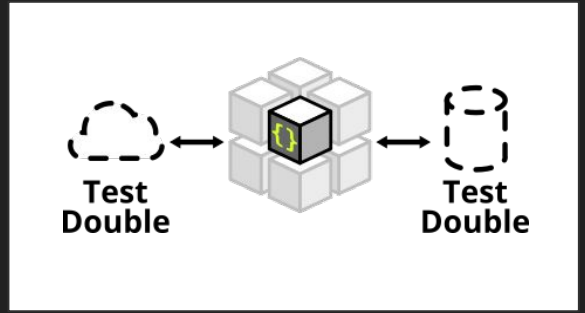


Testes unitários

Classes ou conjunto de classes isoladas, nenhum teste de mecanismo externo deve ser usado aqui, NENHUM, NADA! evitar(ou nunca criar) conexões socket, stream, http e etc…

Uma unidade só se importa com o código que você está desenvolvendo, aqui também é comum, criar dublês para evitar se preocupar com implementação ou código de outra unidade (dependência indireta).

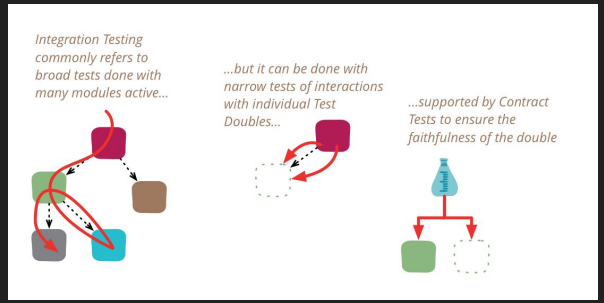
Você pode separar por funcionalidade ou por um conjunto de funcionalidades, se o teste da unidade precisar de configurações distintas, é indicado que se crie mais de uma classe de testes.



Testes de integração

Estes do contrário que muitos pensam são testes que visam a integração entre as unidades, testa-se as unidades separadas e em seguida fazemos as junções e testamos o comportamento completo, novamente aqui não é recomendado nenhuma comunicação externa, NÃO é uma boa prática se conectar com dependências externas, você não está na fase de testar comunicações de terceiros.

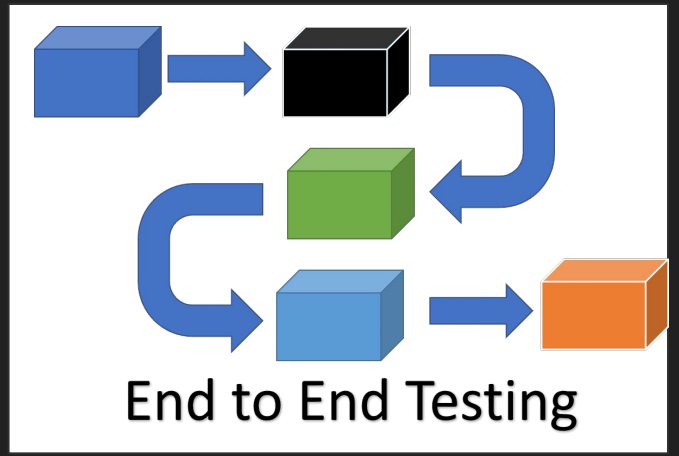
Em códigos de boas práticas aqui é onde trabalhamos a injeção das dependências entre os componente ou classes testada, onde fazemos dublês de comunicações externas visando validar se o que será passado está correto.



Testes de End to End (E2E)

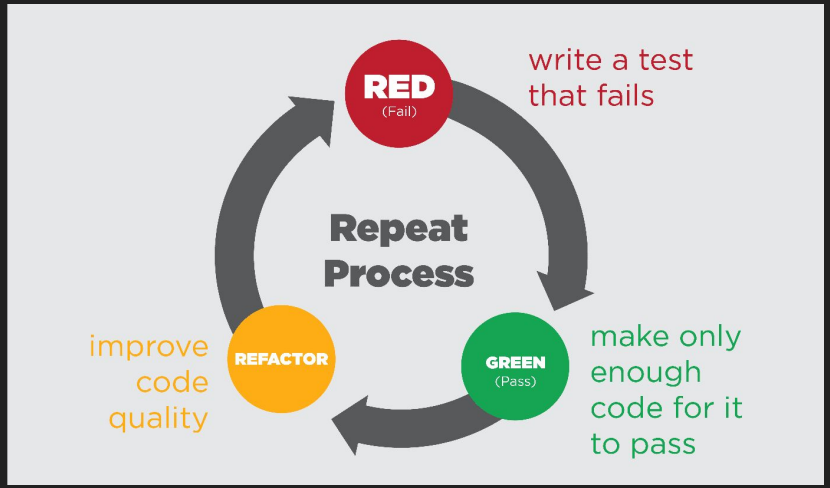
São os testes mais lentos e que visam testar ações reais, esses testes sim precisam de comunicação externas, eles podem ser em CLI usando a aplicação e gerando uma versão real da aplicação ou podem ser testes de interfaces que visam realmente usar um navegador para emular operações reais, desde o front até o backend.

São mais o foco da área de QA(Quality Assurance), são testes frágeis que precisam muito de manutenção e adaptação e tomam muito tempo, numa equipe sem QA esses testes raramente existem ou fazem sentido.



O TDD

O TDD é uma técnica de programação guiada por testes, há um engano muito comum de pensar que esta metodologia se trata de escrever os testes primeiro e depois implementar num geral, mas a realidade é que se trata de escrever o testes pensando na funcionalidade que será desenvolvida, com isso é refletido muito na usabilidade e em seguida, como testar, e assim como implementar a solução.



Testes em PHPUnit

Instalando

```
$ composer install phpunit/phpunit --dev
```

Gerando o arquivo de configuração

```
$ vendor/bin/phpunit --generate-configuration
```

Leia a Documentação: https://docs.phpunit.de/

Como testar em PHPUnit

Criar uma classe de teste

```
ONomeDaSuaClasseTest.php
```

Criar um método de teste

```
public function testDescrevaOQueSeuTesteEstaValidando
```

Exemplo de nome de método

```
public function testComponentShouldCheckAnAction() { }
```

Assertions

Os asserts são funções usadas para garantir as verificações dos testes, testamos os resultados esperados com elas, é também possível verificar a ocorrência de erros.

```
$this->assertEquals(1, 1); // mesmo que 1 == 1
```

```
$this->assert*
```

Stubs vs Mocks vs Spies

Stubs são dublês que simplesmente retornam o esperado, não fazem nada além de simular o resultado de uma comunicação com um componente.

Mocks são dublês com mais responsabilidades, como validar o que é passado na comunicação, quantas vezes o método deve ser chamado e até comportamentos distintos.

Spies são dublês que vão olhar para uma sequência de chamadas, garantindo que os passos esperados aconteceram, eles podem conter as mesma validações dos mocks porém são focados em múltiplos métodos.

Stubs

```php
$dateStub = $this->createStub(\DateTimeImmutable::class);
$dateStub->method( constraint: 'format')
         ->willReturn( value: '20/03/2023')
;

// 20/03/2023
$dataFalsa = $dateStub->format('d/m/Y');

// 20/03/2023
$dataFalsa = $dateStub->format('');
```

Mocks

```php
$dateStub = $this->createMock( originalClassName: \DateTimeImmutable::class);
$dateStub->expects($this->once())
        ->method( constraint: 'format')
        ->with($this->equalTo( value: 'd/m/Y'))
        ->willReturn( value: '20/03/2023')
;

// Erro
$dataFalsa = $dateStub->format('Y-m-d');
```

Mocks

```
Expectation failed for method name is "format" when invoked 1 time
Parameter 0 for invocation DateTimeImmutable::format('Y-m-d'): string does not match expected value.
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'d/m/Y'
+'Y-m-d'
```

Spy

```php
$spy = \Mockery::spy( ...args: MyDependency::class);
$sut = new MyClass($spy);

// act
$sut->callFoo();

// assert
$spy->shouldHaveReceived()
    ->foo()
    ->with('bar');
```

Fixtures

Fixtures visam preparar ou desarmar seus testes, tem em vista alguma ação necessária de preparação ou destruição, por exemplo, podemos criar um arquivo que precisa ser escrito e já existir para um teste.

Ou sempre prepara um objeto de teste mais complexo.

```php
public static function setUpBeforeClass(): void { }

protected function setUp(): void { }
public function testSeuTesteAqui() { }
protected function tearDown(): void { }

public static function tearDownAfterClass(): void { }
```

Coverage

O coverage gera um relatório da situação da cobertura dos testes, lembre-se que cobertura só informa os locais onde o teste não passa, é importante para nos dar objetivos e métricas do quanto estamos cobrindo, porém é importante lembrar que cobertura é diferente de amplitude, ou seja o coverage não garante que seus testes estão amplos no quesito possibilidades.

	Code Coverage							
	Lines			Functions and Methods			Classes and Traits	
Total	24.51%	1172 / 4781		39.16%	242 / 618		23.43%	41 / 175
Actions	11.59%	8 / 69		11.11%	1 / 9		12.50%	1 / 8
Attributes	100.00%	3 / 3		100.00%	2 / 2		100.00%	1 / 1
Console	8.70%	51 / 586		50.00%	24 / 48		0.00%	0 / 24
Exceptions	75.00%	18 / 24		50.00%	1 / 2		0.00%	0 / 1
Helpers	79.31%	23 / 29		25.00%	1 / 4		n/a	0 / 0
Http	26.24%	398 / 1517		52.38%	110 / 210		53.70%	29 / 54
Interfaces	n/a	0 / 0		n/a	0 / 0		n/a	0 / 0
Jobs	0.00%	0 / 412		0.00%	0 / 20		0.00%	0 / 7
Libraries	0.00%	0 / 60		0.00%	0 / 6		0.00%	0 / 2
Listeners	100.00%	9 / 9		100.00%	2 / 2		100.00%	1 / 1
Logging	100.00%	21 / 21		100.00%	4 / 4		100.00%	2 / 2
Models	66.28%	57 / 86		60.87%	28 / 46		8.33%	1 / 12
Modules	5.30%	69 / 1301		4.96%	7 / 141		2.70%	1 / 37
Providers	65.75%	96 / 146		50.00%	12 / 24		30.77%	4 / 13

Coverage

```
$ vendor/bin/phpunit --coverage-html [dir]
```

Existem vários formatos que podem se adequar com seu CI/CD como texto, veja exemplos de parametros:

```
--coverage-clover <file>    Generate code coverage report in Clover XML format
--coverage-cobertura <file> Generate code coverage report in Cobertura XML format
--coverage-crap4j <file>    Generate code coverage report in Crap4J XML format
--coverage-html <dir>       Generate code coverage report in HTML format
--coverage-php <file>       Export PHP_CodeCoverage object to file
--coverage-text=<file>      Generate code coverage report in text format [default: standard output]
--coverage-xml <dir>        Generate code coverage report in PHPUnit XML format
```

Hands-on

Vamos implementar um componente de CEP que faz a validação de entrada de um cep e resgata o dados reais de um cep passado para uma API.

O foco será nos testes unitários por enquanto.

Será postado 5 aulas que vou mostrar uma prática mais complexa de testes com TDD na nossa plataforma.

Desafio

Presente no repositório do nosso curso:

https://github.com/DifferDev/NextGenPHP

Boa noite!

Obrigado pela presença!