# Introdução

## A vida sem testes automatizados

* Quebrar em produção Prejuizo financeiro
  + custo de mídia ou propagandas
  + custo de oportunidade
* Qualidade de vida
  + Estresse
  + Trabalhar até tarde/Final de semana
* Limitações no processo de desenvolvimento
  + Deploys em dias ou horários específicos
  + Pessoas validando tudo

## Testes Manuais

* Mais lentos
* Não exploram muitos cenários (fluxos)
* Talvez não tão confiáveis
* Adiciona uma etapa ao processo (Um responsável)
* Caros (Envolve o tempo de uma pessoa)

## Vantagens de se fazer testes automatizados

* Mais rápidos (Muito mais fácil de testar vários cenários)
* Aplicação muito mais estável (Cobre cenários mais críticos)
* Aumenta confiança do time
* Melhora o levantamento de requisitos (Com testes é possível pensar em mais cenários possíveis)
* Documenta as regras de negócio
* Menor custo

# Criando nosso próprio framework de testes

## Entendendo a dinâmica de um teste

Entender o que eu quero testar

O que vai variar no meu cenário

Como seria a criação desses teste de maneira ilustrativa (pseudo Código):

| SerchTest // Classe de teste  shouldShowBioTest{ // Método dos nossos testes (Cenários)  $google = **new** Google(); // Instância da classe que a gente quer testar  $page = $google->serch("Obama"); // Métdo do teste a ser tomado  assertTrue($page->hasBio()); //   // asserção é a parte do teste que determina se o teste passou ou não    }  shouldShowCalculateTest{  $google = **new** Google();  $page = $google->serch("1+1");  assertTrue($page->hasCalculator());  // asserção é a parte do teste que determina se o teste passou ou não   }   shouldShowNotFoundTest{  $google = **new** Google();  $page = $google->serch("Obama");  assertTrue($page->hasNotFoundMessage());  // asserção é a parte do teste que determina se o teste passou ou não   } |
| --- |

## Material

[Link do material no GitHub](https://github.com/viniciuswebdev/curso-php-phpunit)

## Testando uma calculadora de descontos

## Criando nossa própria ferramenta de testes

## Encontrando cenários e estruturando os testes

Question 1

Quais são as partes que geralmente estruturam um teste?

* Preparação, execução e asserção

Exato! Preparação (Instanciar a classe alvo de teste com seu cenário), execução (Chamar o método alvo) e asserção (Conferir se o resultado veio como esperado)

Question 2

Quantos cenários seriam possíveis testar no seguinte problema ?

| <?php   **public** **function** **categorizeInvestor**($variableIncomePercentage) {    **if** ($variableIncomePercentage > 90) {  **return** "AGGRESSIVE";  }    **if** ($variableIncomePercentage > 30) {  **return** "MODERATE";  }    **return** "CONSERVATIVE";   } |
| --- |

* 3 cenários

Exato! Maior que 90, maior que 30 e menor que 30.

Question 3

Quantos cenários seriam possíveis testar no seguinte problema?

| <?php   **public** **function** **isPreferential**($customer) {    **if** ($customer->isPregnant() || $customer->isAged()) {  **return** **true**;  }    **return** **false**;   } |
| --- |

* 3 cenários

O correto seria 3, pregnant, aged e nenhum dos dois.

# Diferenciando os tipos de testes

## Testes de integração

* Testa a aplicação de forma integrada (interface visual, endpoit de api, query de banco de dados)
  + broad integration tests
    - testa a integração por completo
  + narrow integration tests
    - testa partes da integração isoladas, quebradas, query de banco de dados, módulos com outros, classe com outros.
* Cobre mais códigos nos primeiros testes
* Bons para começar
* Mais lentos e configuração do ambiente mais trabalhosa
* Mais difícil de encontrar o motivo da falha

## Testes unitários

Definição -> Quebra o sistema em vários pedacinhos e submete cada pedacinho a vários testes, inicialmente podemos falar que são esses pedacinhos são classes mas não é descartada a ideia de que podem ser também funções:

No paradigma de orientção a objetos é visto como classes.

No paradigma de orientação funcional, são visto como como funções

* Testa de forma contextual
* Mais rápidos
* Mais fácil de encontrar a falha (Por ser totalmente local)
* Mais testes para chegar numa boa corbertura
* Setup mais simples (Precisa somente da linguagem)

## Diferenciando testes unitários de testes de integração

Questão 1

Este teste é um teste de integração ou um teste unitário?

| **class** **PostControllerTest** **extends** **WebTestCase** {  **public** **function** **testShowPost**()  {  $client = **static**::createClient();    $client->request('GET', '/post/hello-world');    **$this**->assertEquals(200, $client->getResponse()->getStatusCode());  } } |
| --- |

Exato! Pois está testando um endpoint de uma API, ou seja, vai executar a aplicação inteira

Questão 2

Este teste é um teste de integraação ou um teste unitário?

| **class** **CalculatorTest** **extends** **TestCase** {  **public** **function** **testAdd**()  {  $calculator = **new** Calculator();    $result = $calculator->add(30, 12);    **$this**->assertEquals(42, $result);  } } |
| --- |

Exato! Pois a classe está isolada das demais funcionalidades do sistema.

# Praticando com um aplicação de mercado

## 

## Apresentação do projeto

## Organizando nossos arquivos de teste

## Implementando Data Providers

## Escrevendo bons nomes para os cenários

| shouldBeValidWhenValueIsANumber WhenValueIsANumberShouldBeValid |
| --- |

Nome do cenário deve conter:

* O que está sendo testado?
* Valid
* Quais as circunstâncias ?
* IsANumber
* Qual o resultado esperado?
* Be Valid
* shouldBeValid

Outras possibilidades:

should\_BeValid\_When\_ValueIsANumber

IsValid\_ValueIsANumber\_true

IsValid\_True\_ValueIsANumber

BeValidIfValueIsANumber

## Devemos testar entidades?