



Testes e Qualidade de Software

CteSP Desenvolvimento de Software

Testes

- Testar é uma fase fundamental do desenvolvimento de software
- De forma formal ou não, o programador está sempre a testar: usando instruções *printf* ou *echo*, ou introduzindo o *input* e verificando o *output* e correto funcionamento do software
- No entanto, este processo pode ser moroso e pouco fiável: sempre que introduzimos uma alteração numa parte do software, nem sempre testamos que uma outra parte deixou de funcionar da forma pretendida



Testes

- Para facilitar o trabalho do programador, aumentar a produtividade e melhorar a fiabilidade do software produzido, é preciso automatizar o processo de testes
- Test-Driven Development (TDD) e Behavior-Driven Development (BDD) são duas abordagens de desenvolvimento de software onde primeiro é descrito qual o comportamento esperado, e só depois é feita a implementação



Testes

- Este tipo de desenvolvimento procede-se da seguinte forma:
 - Criar o teste que descreve o comportamento esperado
 - Correr o teste e verificar que o mesmo falha (pois ainda não foi implementado)
 - Implementar o software que permite passar o teste com sucesso
 - Correr o teste e verificar que o mesmo passou com sucesso



Testes

- O processo anterior poderá ser demasiado moroso e complexo, pelo que nem sempre é a abordagem a seguir. Então quando testar?
 - O software é grande e complexo
 - Os requisitos são complexos e crescentes
 - O projeto é para ser usado a longo prazo
 - Os custos de uma possível falha são demasiado grandes



Testes

- Quando é que a implementação de testes é demasiado custosa?
 - O projeto é simples e não irá ficar mais complexo
 - É um projeto para ser usado apenas uma vez (utilização limitada no tempo)

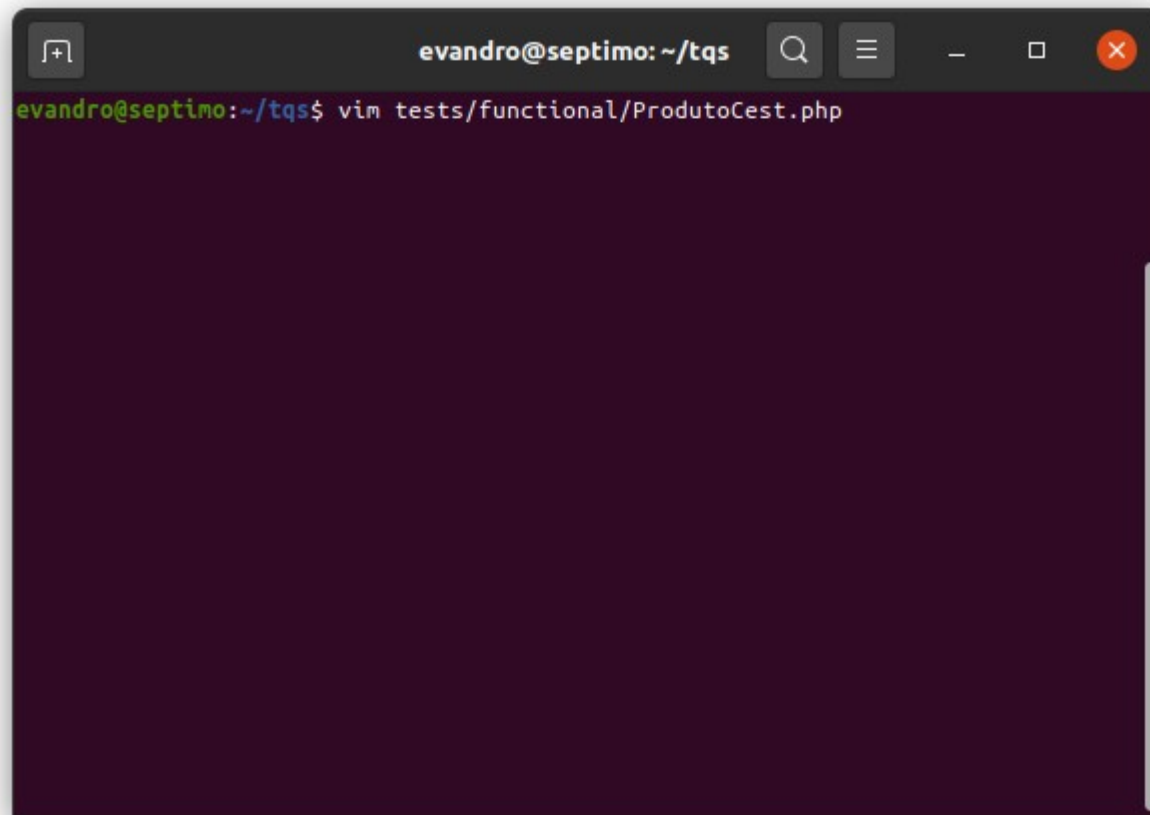


Testes

- Que tipo de testes é possível definir:
 - Testes unitários
 - Testes que validam uma unidade elementar de código
 - Testes funcionais
 - Testes que validam um cenário do ponto de vista do utilizador
 - Testes de comportamento
 - Testes idênticos aos funcionais mas realizados num ambiente real

Testes

- Criar um teste funcional



A terminal window with a dark background and light text. The window title bar shows 'evandro@septimo: ~/tqs' and standard window controls. The command prompt shows 'evandro@septimo:~/tqs\$' followed by the command 'vim tests/functional/ProdutoCest.php'.

```
evandro@septimo: ~/tqs
evandro@septimo:~/tqs$ vim tests/functional/ProdutoCest.php
```


Testes

- Criar um teste funcional

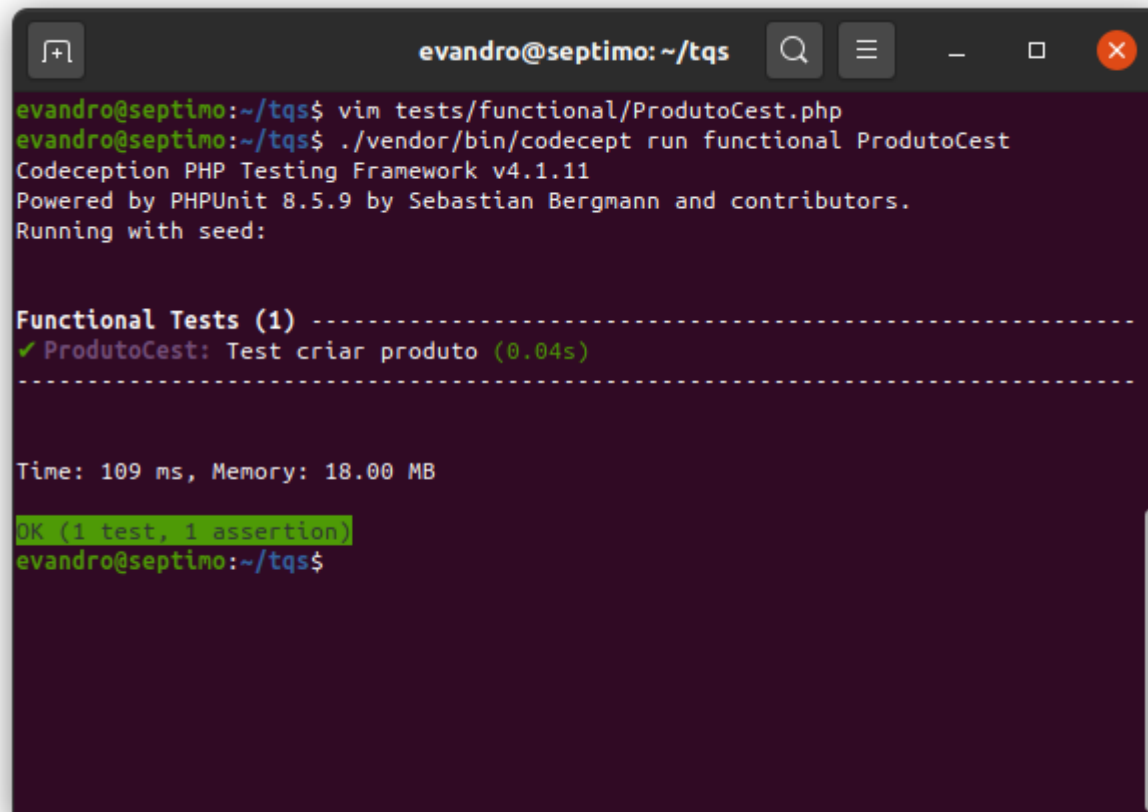
```
evandro@septimo: ~/tqs
<?php
use app\models\Produto;

class ProdutoCest
{
    public function testCriarProduto(\FunctionalTester $I)
    {
        $I->amOnRoute('/produto/create');
        $I->submitForm('form', [
            'Produto[nome]' => 'Produto Bom',
            'Produto[preco]' => '10',
        ]);
        $I->canSee('Produto Bom');
    }
}

~
~
~
~
~
~
"tests/functional/ProdutoCest.php" 17L, 332C 1,1 Tudo
```

Testes

- Correr um teste funcional



```
evandro@septimo: ~/tqs
evandro@septimo:~/tqs$ vim tests/functional/ProdutoCest.php
evandro@septimo:~/tqs$ ./vendor/bin/codecept run functional ProdutoCest
Codeception PHP Testing Framework v4.1.11
Powered by PHPUnit 8.5.9 by Sebastian Bergmann and contributors.
Running with seed:

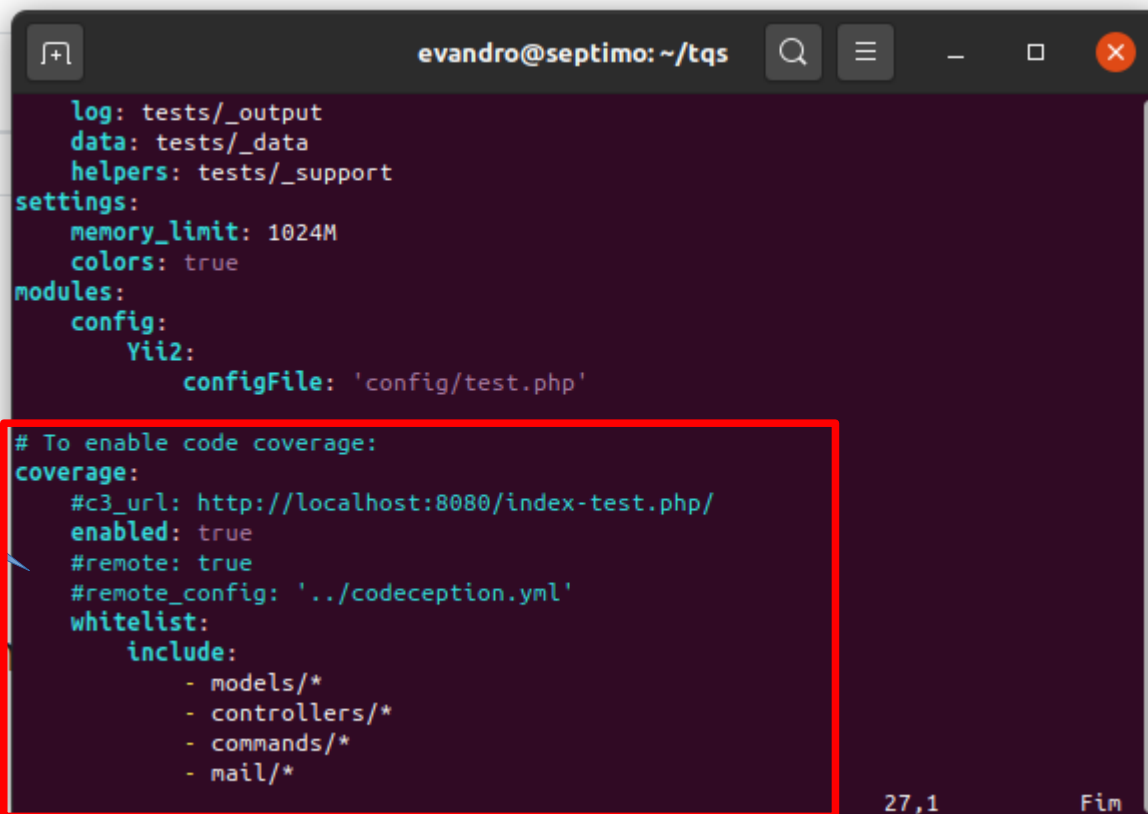
Functional Tests (1) -----
✓ ProdutoCest: Test criar produto (0.04s)
-----

Time: 109 ms, Memory: 18.00 MB
OK (1 test, 1 assertion)
evandro@septimo:~/tqs$
```

Testes

- Configurar relatório de cobertura
 - Editar o ficheiro *codeception.yml*

Descomentar

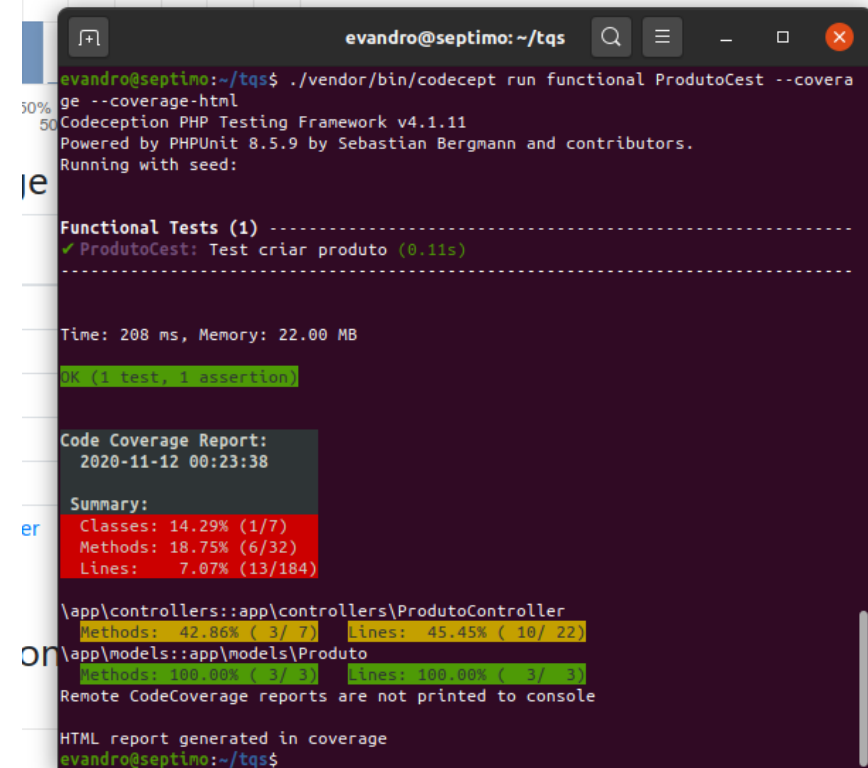


```
evandro@septimo: ~/tqs
log: tests/_output
data: tests/_data
helpers: tests/_support
settings:
  memory_limit: 1024M
  colors: true
modules:
  config:
    Yii2:
      configFile: 'config/test.php'

# To enable code coverage:
coverage:
  #c3_url: http://localhost:8080/index-test.php/
  enabled: true
  #remote: true
  #remote_config: '../codeception.yml'
  whitelist:
    include:
      - models/*
      - controllers/*
      - commands/*
      - mail/*
```

Testes

- Correr teste com geração do teste de cobertura com os parâmetros:
 - --coverage
 - --coverage-html



```
evandro@septimo: ~/tqs
evandro@septimo:~/tqs$ ./vendor/bin/codecept run functional ProdutoCest --coverage --coverage-html
Codeception PHP Testing Framework v4.1.11
Powered by PHPUnit 8.5.9 by Sebastian Bergmann and contributors.
Running with seed:

Functional Tests (1) -----
✓ ProdutoCest: Test criar produto (0.11s)
-----

Time: 208 ms, Memory: 22.00 MB
OK (1 test, 1 assertion)

Code Coverage Report:
 2020-11-12 00:23:38

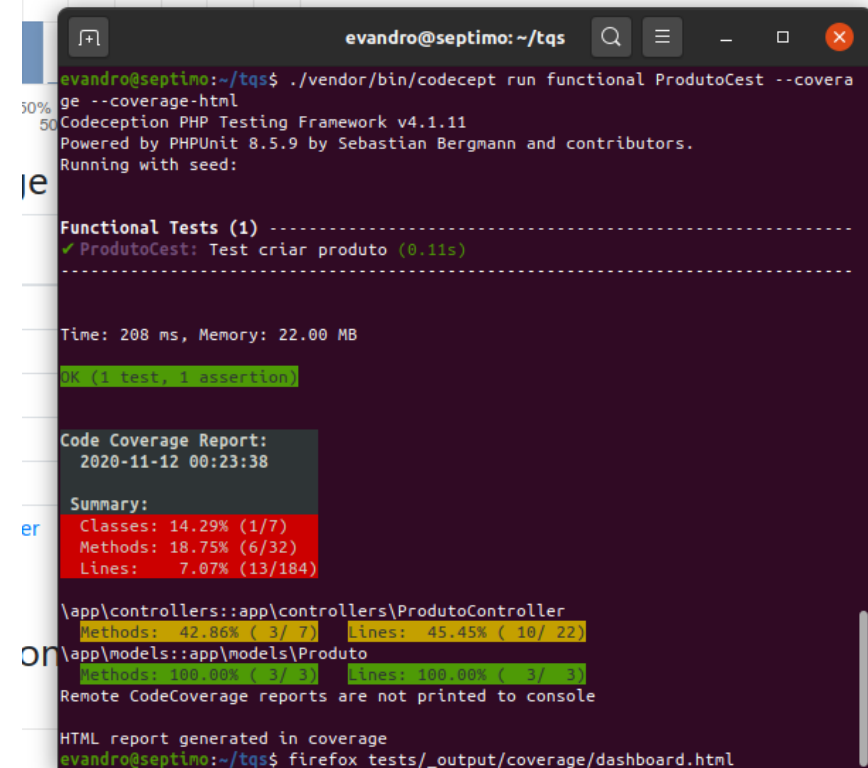
Summary:
Classes: 14.29% (1/7)
Methods: 18.75% (6/32)
Lines: 7.07% (13/184)

/app\controllers::app\controllers\ProdutoController
Methods: 42.86% ( 3/ 7) Lines: 45.45% ( 10/ 22)
/app\models::app\models\Produto
Methods: 100.00% ( 3/ 3) Lines: 100.00% ( 3/ 3)
Remote CodeCoverage reports are not printed to console

HTML report generated in coverage
evandro@septimo:~/tqs$
```

Testes

- Abrir o relatório de cobertura HTML em
 - tests/_output/coverage/dashboard.html



A terminal window showing the execution of a test suite and the generation of a code coverage report. The terminal output includes the command to run the tests, the test results for 'ProdutoCest', and a detailed code coverage report for the 'ProdutoController' and 'Produto' models.

```
evandro@septimo: ~/tqs
evandro@septimo:~/tqs$ ./vendor/bin/codecept run functional ProdutoCest --coverage --coverage-html
Codeception PHP Testing Framework v4.1.11
Powered by PHPUnit 8.5.9 by Sebastian Bergmann and contributors.
Running with seed:

Functional Tests (1) -----
✓ ProdutoCest: Test criar produto (0.11s)
-----

Time: 208 ms, Memory: 22.00 MB
OK (1 test, 1 assertion)

Code Coverage Report:
 2020-11-12 00:23:38

Summary:
Classes: 14.29% (1/7)
Methods: 18.75% (6/32)
Lines: 7.07% (13/184)

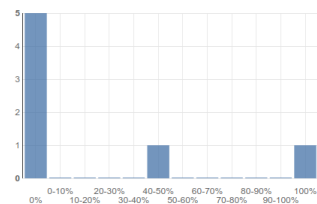
/app\controllers::app\controllers\ProdutoController
Methods: 42.86% ( 3/ 7) Lines: 45.45% ( 10/ 22)
/app\models::app\models\Produto
Methods: 100.00% ( 3/ 3) Lines: 100.00% ( 3/ 3)
Remote CodeCoverage reports are not printed to console

HTML report generated in coverage
evandro@septimo:~/tqs$ firefox tests/_output/coverage/dashboard.html
```

Testes

Classes

Coverage Distribution

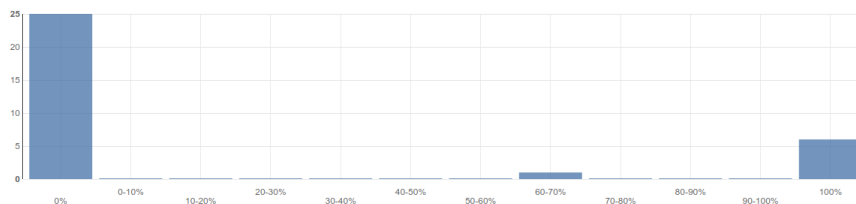


Insufficient Coverage

Class	Coverage
app\commands\HelloController	0%
app\controllers\SiteController	0%
app\models\ContactForm	0%
app\models\LoginForm	0%
app\models\User	0%
app\controllers\ProdutoController	45%

Methods

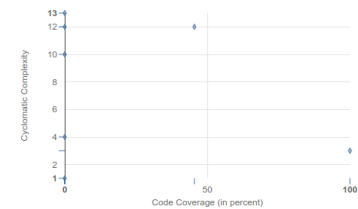
Coverage Distribution



Insufficient Coverage

Method	Coverage
actionIndex	0%
contact	0%

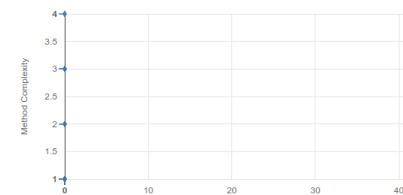
Complexity



Project Risks

Class
app\controllers\SiteController
app\models\User
app\models\LoginForm
app\controllers\ProdutoController
app\models\ContactForm

Complexity



Project Risks

Method
validatePassword
actionLogin

Testes

- Da análise de cobertura podemos obter os seguintes indicadores:
 - percentagem de cobertura do código fonte
 - quais as classes e métodos que representam um grau de incerteza maior
 - qual o nível de risco do projeto
 - ao nível das classes e métodos usando o indicador Change Risk Anti-Patterns (CRAP)



Testes

- O indicador CRAP dá uma avaliação de risco do projeto cruzando a complexidade do projeto com a percentagem de cobertura dos testes
- A complexidade do código é feita analisando a quantidade de pontos de decisão existentes, isto é, o número de caminhos possíveis que a execução do código pode ter, e tem o nome de *cyclomatic complexity*

Testes

