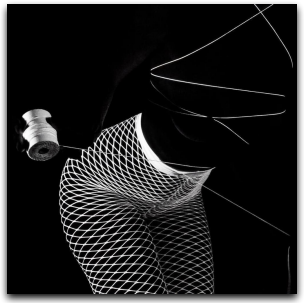


Tônus - carne doce (2018)

Live de Python #75

Testes de unidade (unitários)

Ajude a Live de Python

apoia.se/livedepython

picPay: @livedepython

Roteiro

- Testes, pra que testes?
- Ciclo de feedback
- Quando escrever testes
- Quando rodar os testes
- xUnit-style
- Unittest

Testes, pra que testes?

A ideia principal dos testes é garantir que a coisa funciona de fato.

Tá, isso é muito vago.

Imagine que alguém use seu código (Você, seu amigo, a equipe de QA, o cliente que te paga por isso) então tem alguma coisa ou alguém dependendo de que isso funcione.

Então, para desenvolver um bom software ele precisa funcionar como esperado. E dependendo de como os testes são executados, mais rápido você vai ter um *feedback* de que ele está como deveria.

Ciclo de feedbacks

Então vamos imaginar que você escreva uma linha de código, pare de programar, execute o programa e veja se ele retorna o que precisa.

Sim, isso é um teste. Mas isso demora. Imagina toda linha de código escrita, uma média de 5 minutos pra subir e tudo e rodar. E quando a aplicação é mais complexa? Demora mais tempo.

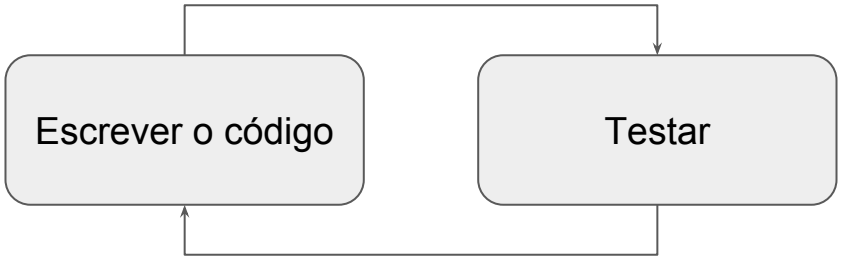
Ciclo de feedbacks

Temos outra situação,

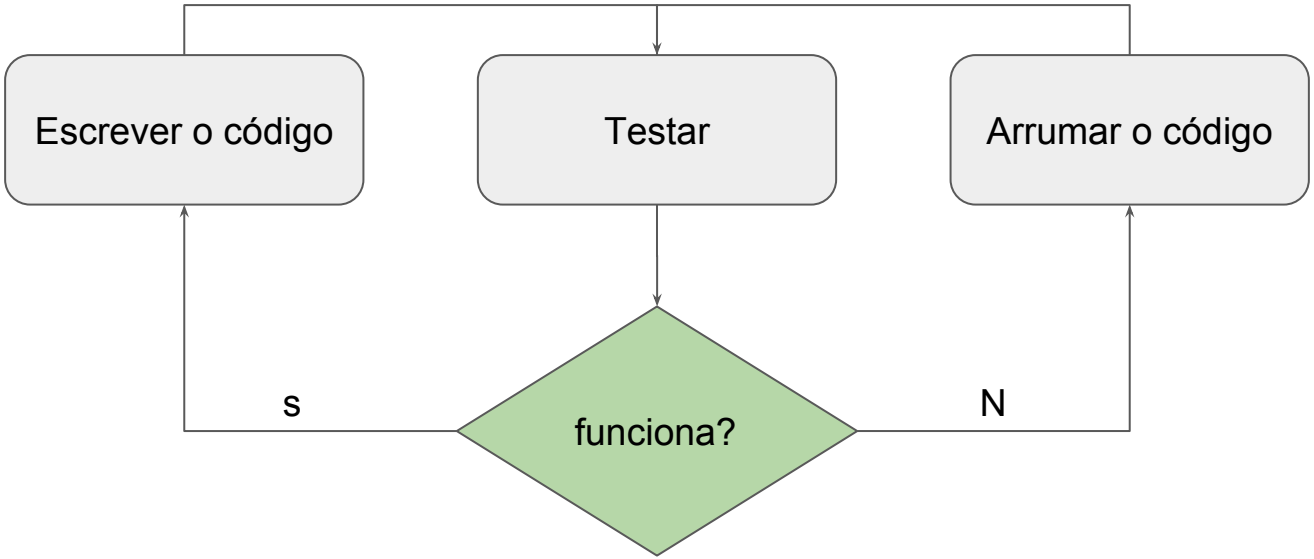
vamos imaginar que você tem um colega de trabalho que testa as coisas de todo mundo. Então você faz a modificação e passa pra ele. Só que ele tem outros testes para fazer. Logo você vai ter que testar algumas vezes antes dele testar de fato, pq você precisa ver se aqui pode ser mandado pra ele.

Mesmo que você não teste, esse ciclo vai demorar.

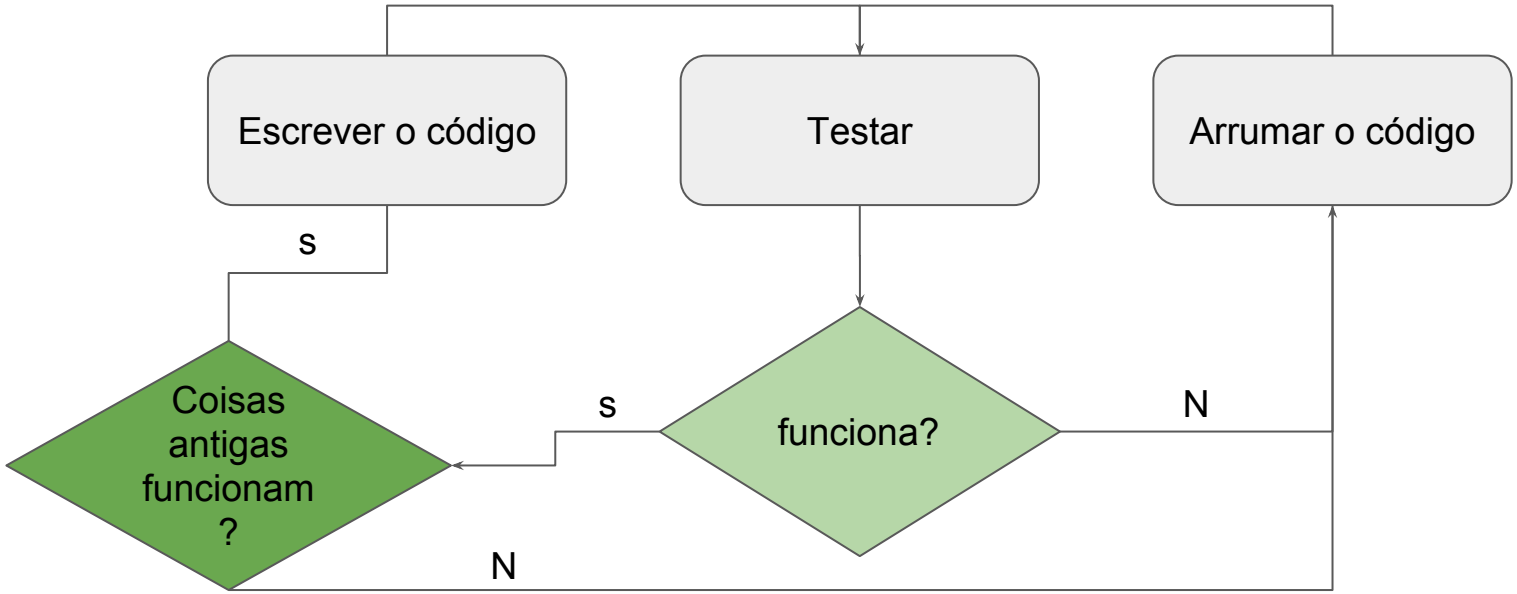
Ciclo de feedbacks



Ciclo de feedbacks



Ciclo de feedbacks



Ciclo de feedbacks

Vamos pensar que cada vez que você para tudo e testa você gasta 1 minuto pra fazer cada teste. Se você tem 30 unidades (funções, classes, etc..) Cada vez que você mudar algo vai levar 30m pra testar. Se você mudar bastante coisa, vai levar mais de um dia...

Espera... Computadores não são bons em executar tarefas trabalhosas e repetitivas?

Quando escrever testes?

Acho que a pergunta foi respondida, não é mesmo? O tempo todo...

O legal de escrever testes é que diferente das tarefas manuais completamente chatas... Eles vão durar pra sempre.

Quando rodar seus testes?

Acho que também já foi respondido. Quanto mais rodar, mais feedback.

xUnit-Style

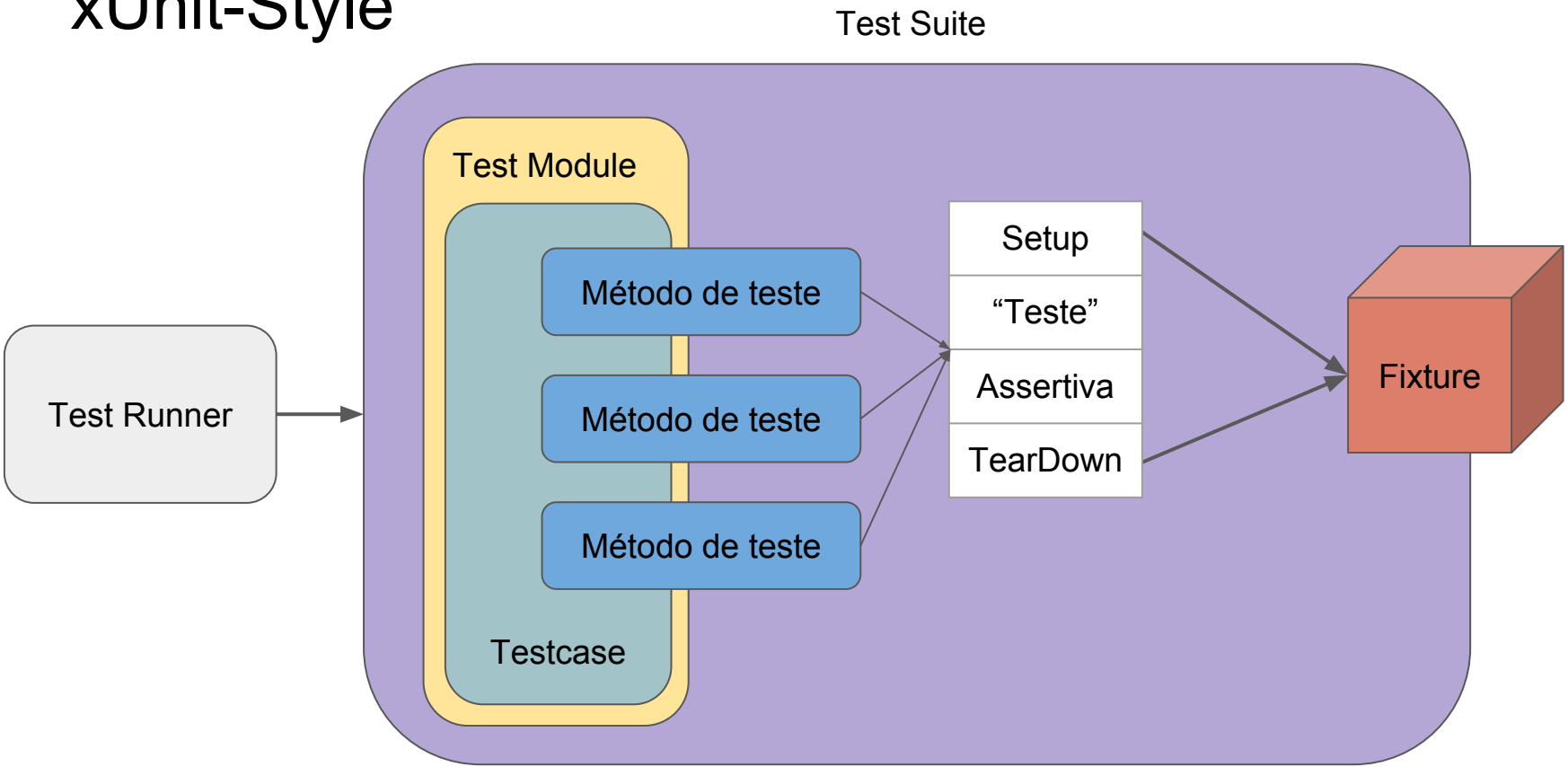
xUnit é uma filosofia de frameworks de testes. Ele nasceu no Smalltalk e lá se chamava SUnit.

O fato de ser baseado no smalltalk torna tudo MUITO burocrático. Mas isso é detalhe.

xUnit-Style - Glossário

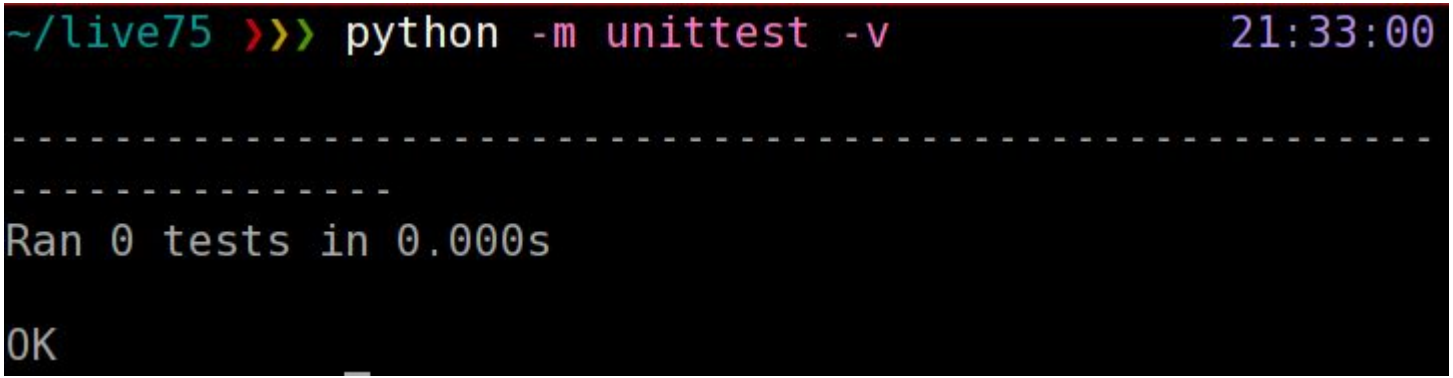
- Test case class
  - Classe base para todas as classes de teste
- Test fixtures
  - Funções ou métodos que são executados antes e depois da execução dos blocos do código de teste.
- Assertions
  - Funções ou métodos são usados para verificar o comportamento do componente que está sendo testado
- Test runner
  - Programa ou bloco de código que executa o conjunto de testes.

xUnit-Style


Test Suite
Test Module
Testcase
Método de teste
Método de teste
Método de teste
Setup
“Teste”
Assertiva
TearDown
Fixture
Test Runner

xUnit-Style

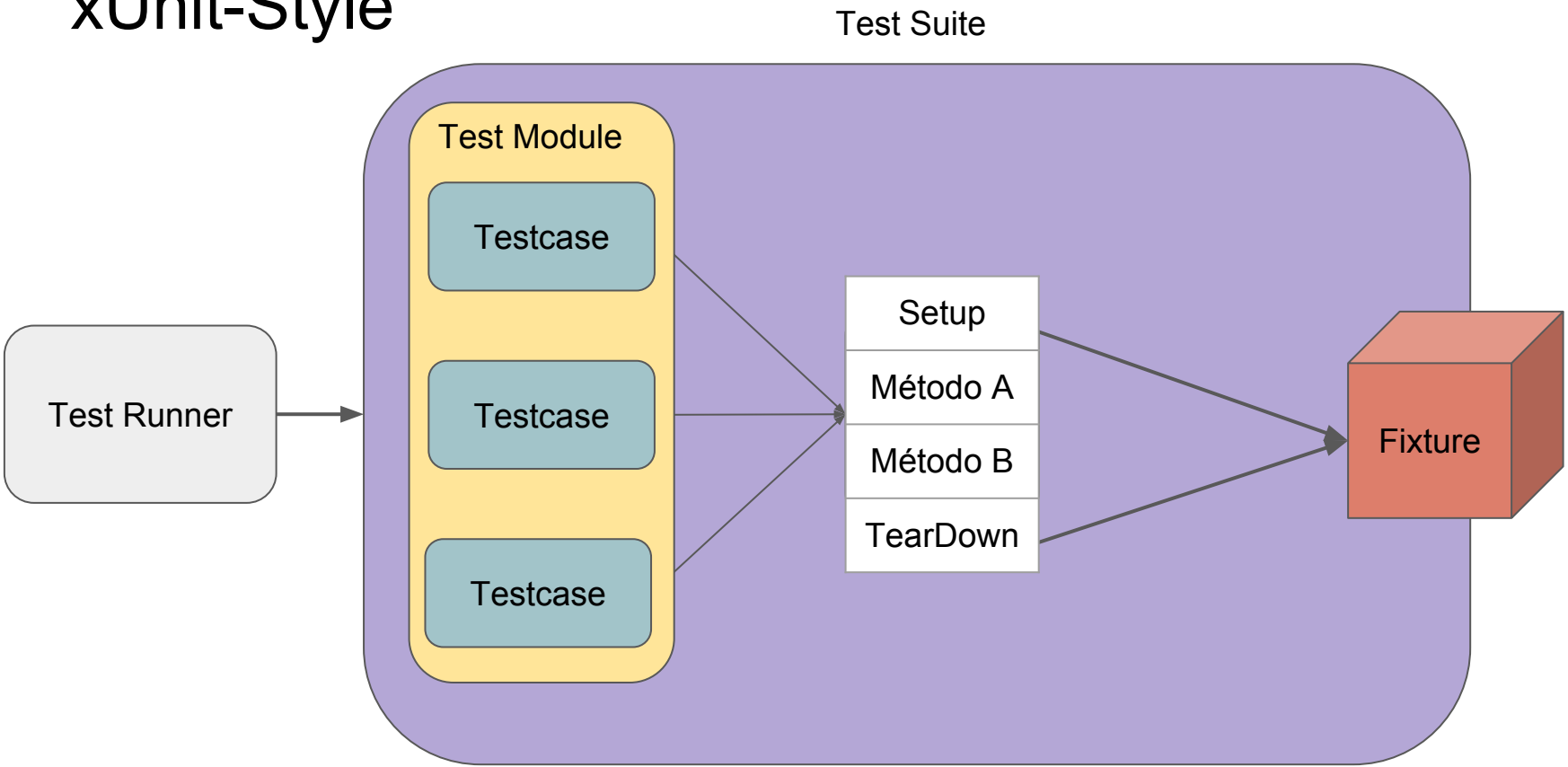
Test Runner



xUnit-Style


class TestCase(unittest.TestCase):
def setUp(self):
print("\nNo setUp()...")
def tearDown(self):
print("No tearDown()...")
def test_case01(self):
print("No test_case01()")
self.assertEqual('a', 'a', msg='a é igual a')
Testcase
Fixtures
Método de teste
Assertiva

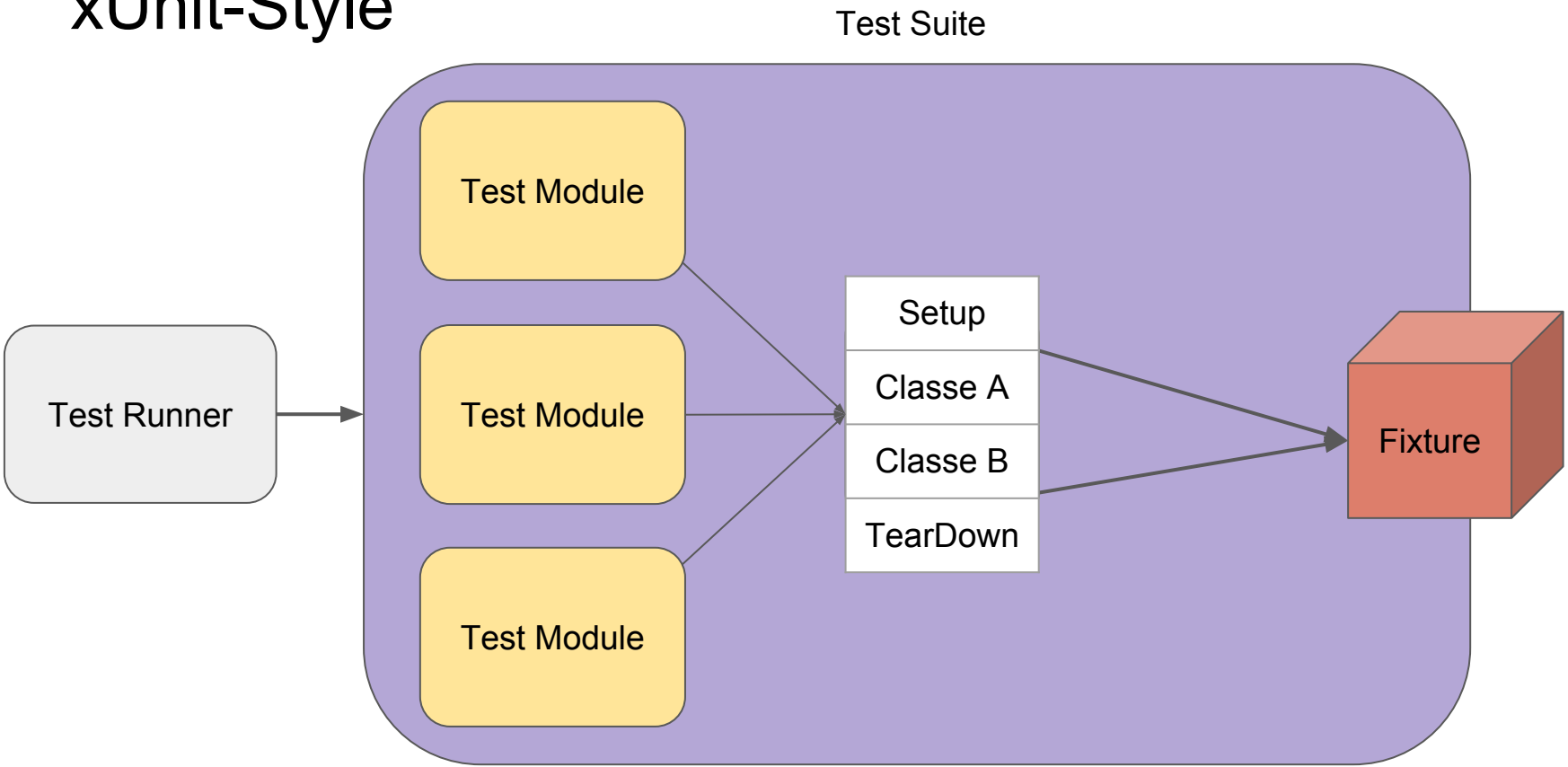
xUnit-Style


Test Suite
Test Module
Testcase
Testcase
Testcase
Test Runner
Setup
Método A
Método B
TearDown
Fixture

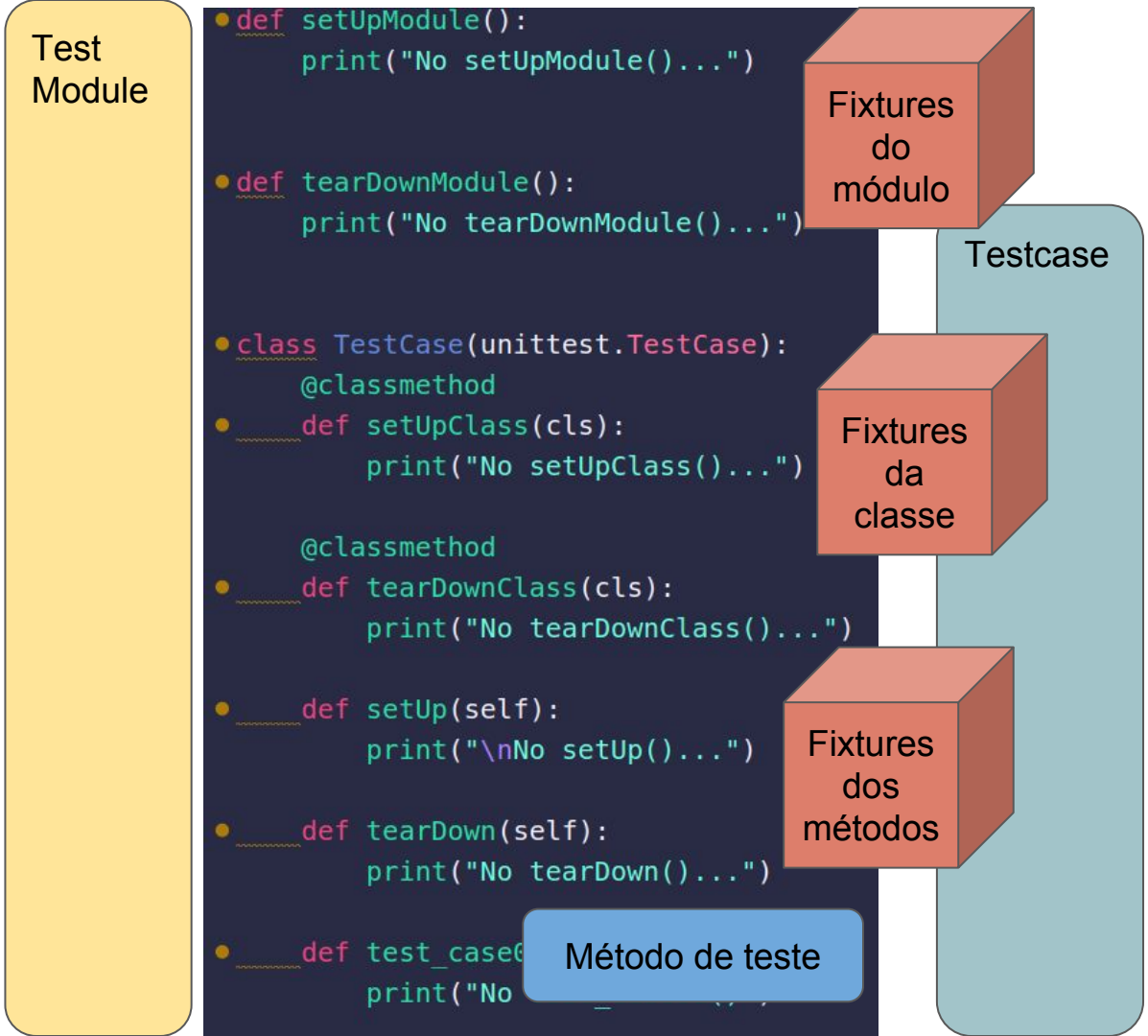
xUnit-Style


Testcase
class TestCase(unittest.TestCase):
@classmethod
def setUpClass(cls):
print("No setUpClass()...")
@classmethod
def tearDownClass(cls):
print("No tearDownClass()...")
def setUp(self):
print("\nNo setUp()...")
def tearDown(self):
print("No tearDown()...")
def test_case01(self):
print("No test_case01()")
Fixtures da classe
Fixtures dos métodos
Método de teste

xUnit-Style


Test Suite
Test Module
Test Module
Test Module
Test Runner
Setup
Classe A
Classe B
TearDown
Fixture

xUnit-Style


Test Module
def setUpModule():
print("No setUpModule()...")
def tearDownModule():
print("No tearDownModule()...")
Fixtures do módulo
Testcase
class TestCase(unittest.TestCase):
@classmethod
def setUpClass(cls):
print("No setUpClass()...")
Fixtures da classe
@classmethod
def tearDownClass(cls):
print("No tearDownClass()...")
def setUp(self):
print("\nNo setUp()...")
Fixtures dos métodos
def tearDown(self):
print("No tearDown()...")
def test_case
print("No
Método de teste

Referências

1. unittest - Unit testing framework
2. Simple Smalltalk Testing: With Patterns
3. Como ser um programador melhor - Pete Goodliffe
4. Guia do mochileiro Python - Reitz & Schlusser