Live de Python #76

Testes de unidade (unitários) p.II
Dublês de teste

Ajude a Live de Python

apoia.se/livedepython

picPay: @livedepython

Roteiro

- Definindo unidades
- SUT
- Fixtures
- Acoplamento
- Dublês de teste

Definindo unidades

Falamos tanto de testes de unidades na live passada(75), porém não definimos uma unidade.

Quando se fala em unidades pode-se entender muita coisa dependendo do “estilo” de código que se faz. E essa definição é um tanto quanto complexa.

Em POO, a menor unidade é um objeto. Em funcional, a menor é uma função.

MAAAAASSSSSS....

Definindo unidades

Python é uma linguagem mista (multi paradigma). Então vou me referir a tudo como módulos[1].

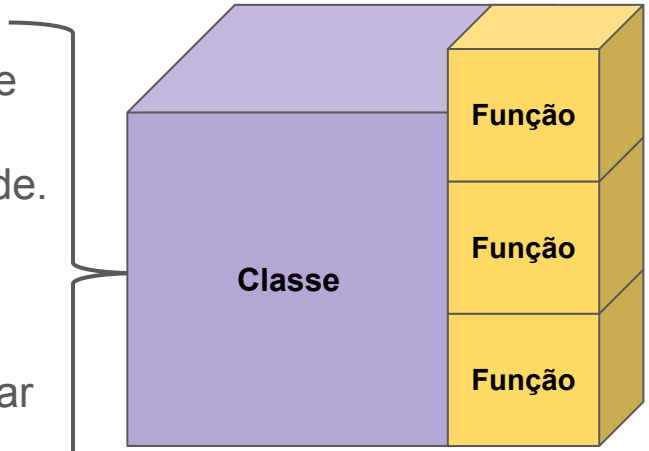
Função -> Módulo
Objeto -> Módulo

São coisas que interagem entre si.

Definindo unidades

Em alguns casos, pode-se pensar em classes “maiores” que uma unidade.

Mas tudo que existe na classe está “acoplado” na mesma. É impossível isolar parte de classe



Definindo Unidades

Testes de unidade devem prezar por[1]:

- Isolado:
  - O teste de unidade não pode conter dependências externas (bancos de dados, apis e etc)
- Stateless:
  - Não se pode guardar estados, ou seja, a cada teste todos os recursos que foram utilizados (instâncias, mocks e tudo mais) devem ser destruídos completamente e novos devem ser criados
- Unitário:
  - É um pouco redundante dizer isso, mas um teste de unidade deve apenas testar **uma** unidade, ou seja, se você começar a instanciar outras unidades já não é mais um teste unitário.

SUT (System Under Test)

Geralmente, “o que está sendo testado” na bibliografia aparece como SUT, mas pode aparecer com nomes diferentes, como[2]:

- SUT: Sistema em teste (A coisa em si)
- CUT: Classe em teste
- MUT: método em testes
- AUT: Tudo em testes (All)
- DOC: Componente de quem o SUT depende

- FUT: Função em teste
- M2UT: Módulo em teste

SUT (System Under Test)

Geralmente, “o que está sendo testado” na bibliografia aparece como SUT, mas pode aparecer com nomes diferentes, como[2]:

- **SUT**: Sistema em teste (A coisa em si)
- CUT: Classe em teste
- MUT: método em testes
- AUT: Tudo em testes (All)
- **DOC**: Componente de quem o SUT depende

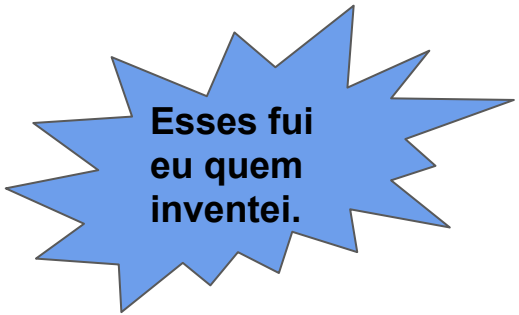
- FUT: Função em teste
- M2UT: Módulo em teste

SUT (System Under Test)

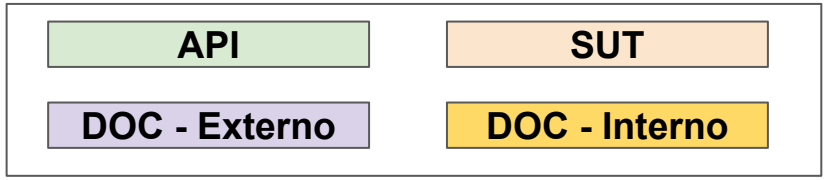
Geralmente, “o que está sendo testado” na bibliografia aparece como SUT, mas pode aparecer com nomes diferentes, como[2]:

- SUT: Sistema em teste (A coisa em si)
- CUT: Classe em teste
- MUT: método em testes
- AUT: Tudo em testes (All)
- DOC: Componente de quem o SUT depende

- FUT: Função em teste
- M2UT: Módulo em teste

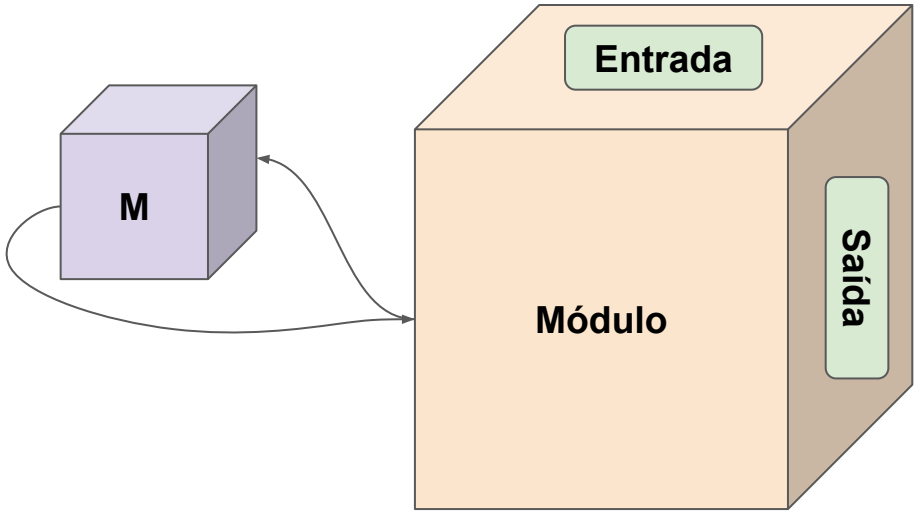


Acoplamento

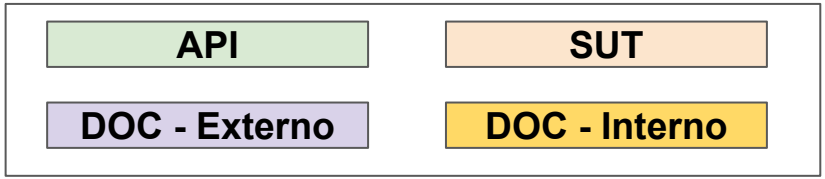


Não é tão complicado entender o sentido de acoplamento.

O quanto uma coisa depende de outra. Por exemplo o módulo A faz uma chamada para B que retorna para A que retorna pra quem chamou.

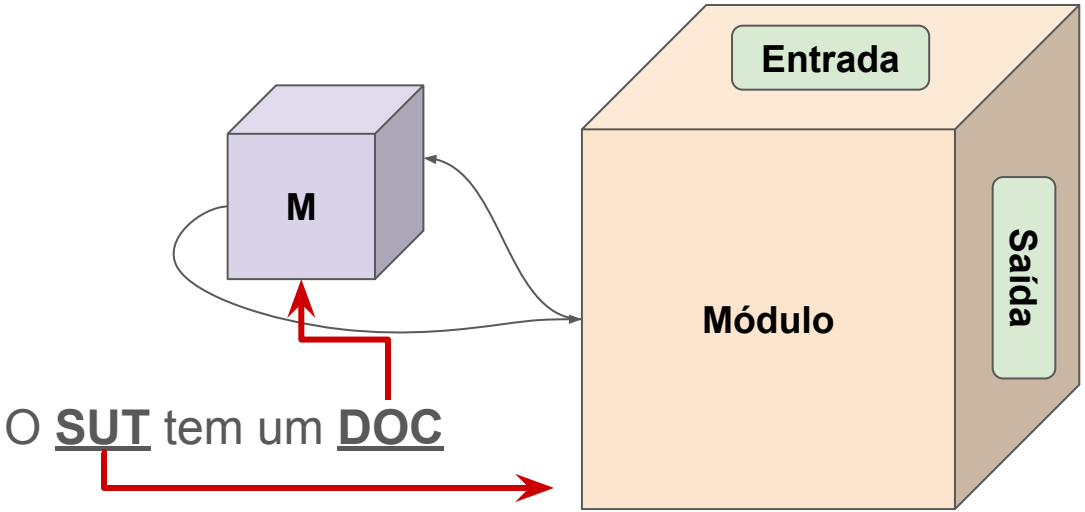


Acoplamento

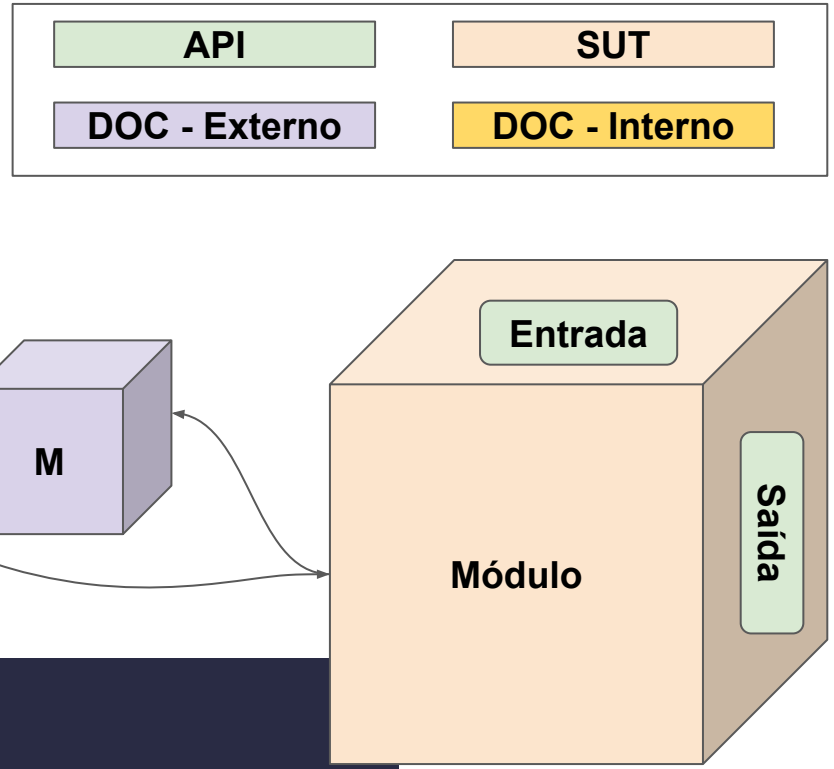


Não é tão complicado entender o sentido de acoplamento.

O quanto uma coisa depende de outra. Por exemplo o módulo A faz uma chamada para B que retorna para A que retorna pra quem chamou.

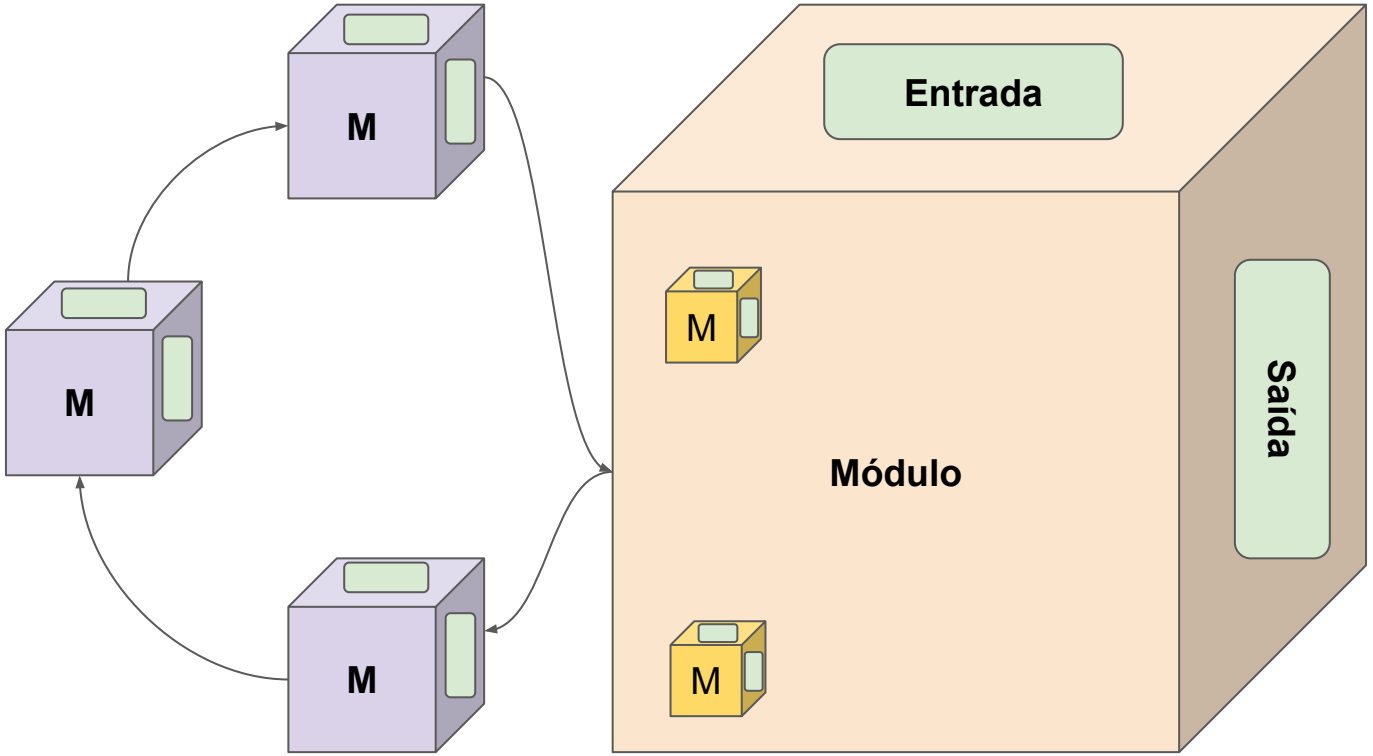


Acoplamento




```python
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

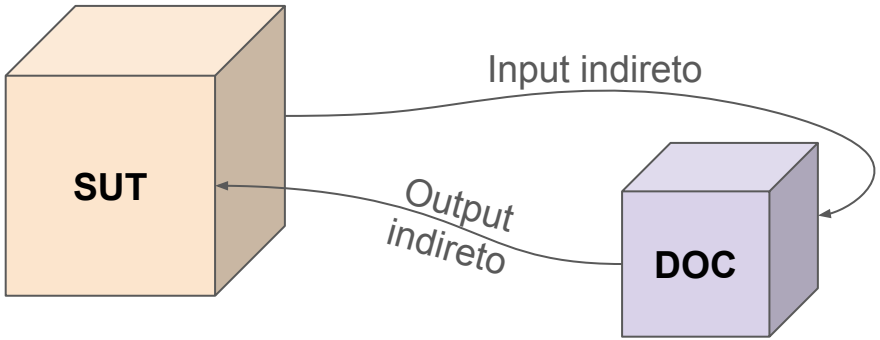
Acoplamento



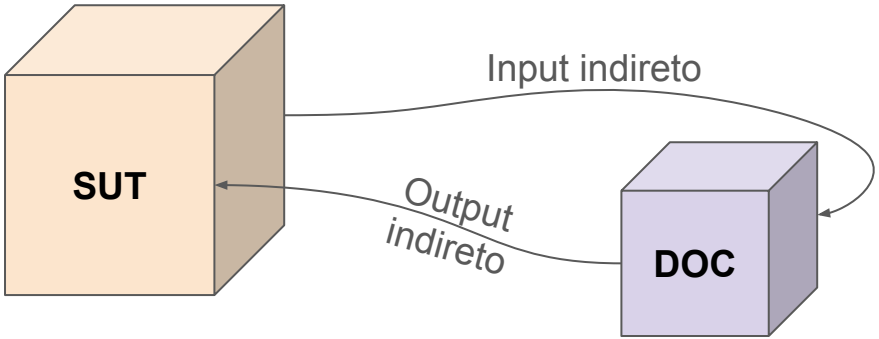
Inputs e outputs **INDIRETOS**

Enquanto chamamos o SUT ele pode conversar com diversos DOCs.

Por exemplo, imagine que possa produzir algum tipo de efeito colateral?



Inputs e outputs **INDIRE**



Muito otimismo pensar que NADA vai dar errado no ‘get’.

```python
def endpoint_up(endpoint_url) -> tuple:
    if not endpoint_url.startswith('http'):
        endpoint_url = f'http://{endpoint_url}'

    request = get(endpoint_url)
    status_code = request.status_code

    # se retornar OK
    if status_code in [200, 201, 202, 302, 304]:
        return True, request.status_code

    # se der erro no endpoint
    elif request.status_code in range(500, 506):
        return False, 'bad request'

    # qualquer coisa não prevista
    else:
        return False, 'Deu ruim'
```

A pergunta de 1M de dólares

Como podemos verificar a lógica de um módulo, independentemente, quando ele depende de outro módulo? (Quando há acoplamento)

Pergunta do Meszaros adaptada ao nosso vocabulário

A resposta de 1M de dólares

Substituímos um componente do qual o SUT depende com um

"equivalente específico de teste".

Resposta do Meszaros adaptada ao nosso vocabulário

A resposta de 1M de dólares

Substituímos um componente do qual o SUT depende com um

~~"equivalente específico de teste".~~

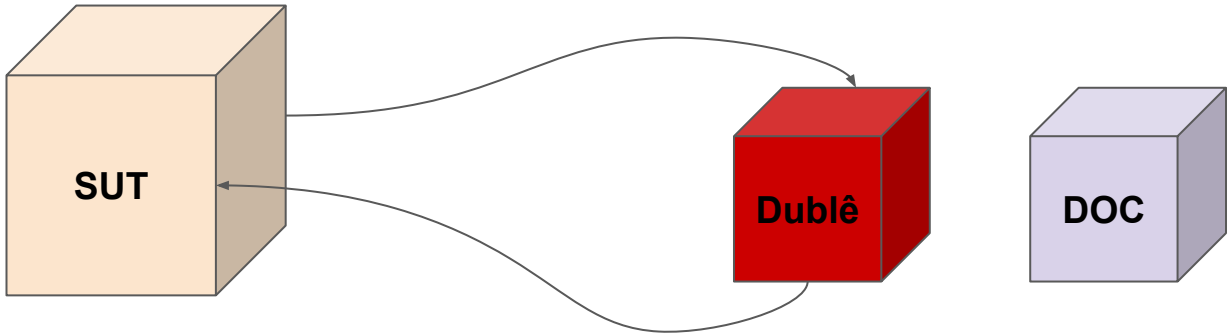
DUBLÊ

Resposta do Meszaros adaptada ao nosso vocabulário

Dublês de teste

Dublês de teste

Então você já deve ter entendido que o dublê faz o trabalho para que os testes possam ser **determinísticos**. Ele vai de ajudar a construir o ambiente que você precisa, tirar o DOC da parada e colocar um cara que a gente sabe exatamente o que ele vai fazer.



Dublês de teste

Existem vários tipos de dublês

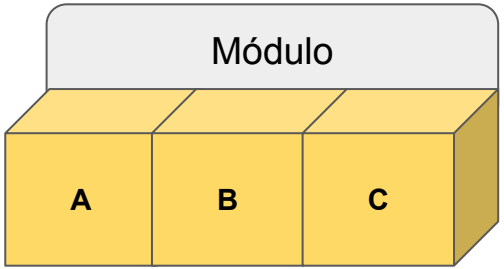
- Dummy
  - São os dublês mais simples. Imagine um parâmetro nulo, algo que o SUT tenha DOC, mas não entra no escopo do teste.
- Fake
  - A ideia do Fake é prover uma implementação de “mentira” para que o SUT consiga seguir o seu fluxo
- Spy
  - Tem a função de validar os inputs indiretos
- Stub
  - Substituem o DOC em um ambiente controlado
- Mock:
  - ???

Dublês de teste

Pra que dublê nessa porra?

Vou testar tudo acoplado

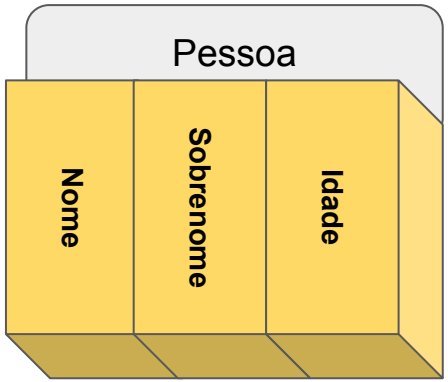
Dummy



Vamos supor que para que o módulo ser testado ele necessariamente precisa receber algo. Ou algo precisa existir para que ele funcione.

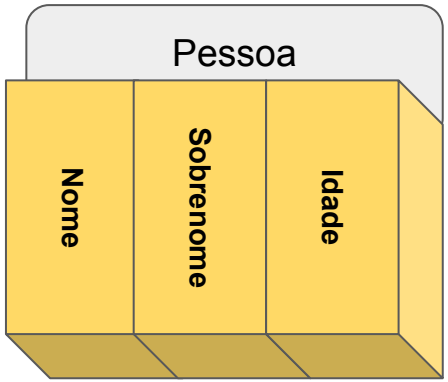
A ideia principal do Dummy não é que ele seja um objeto Nulo, puro e simplesmente, é que ele seja irrelevante para o SUT

Dummy




```python
class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade


Pessoa('Eduardo', 'Mendes')
```

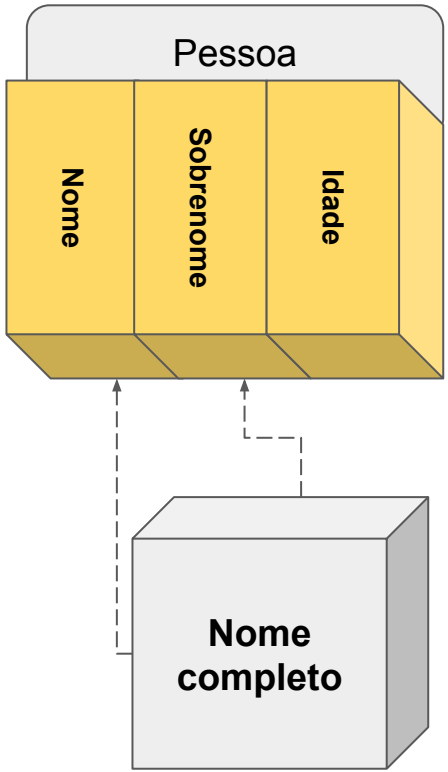
Dummy




```python
class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade


Pessoa('Eduardo', 'Mendes')
```

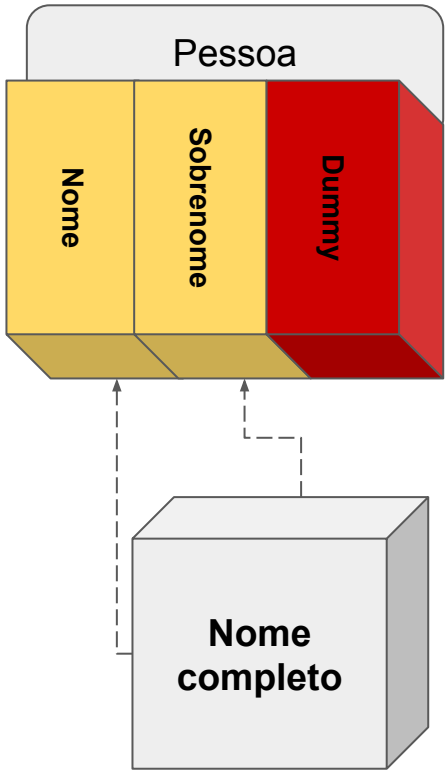
```
Traceback (most recent call last):
  File "dummy_exemplo.py", line 8, in <module>
    Pessoa('Eduardo', 'Mendes')
TypeError: __init__() missing 1 required positional argument: 'idade'
```

Dummy



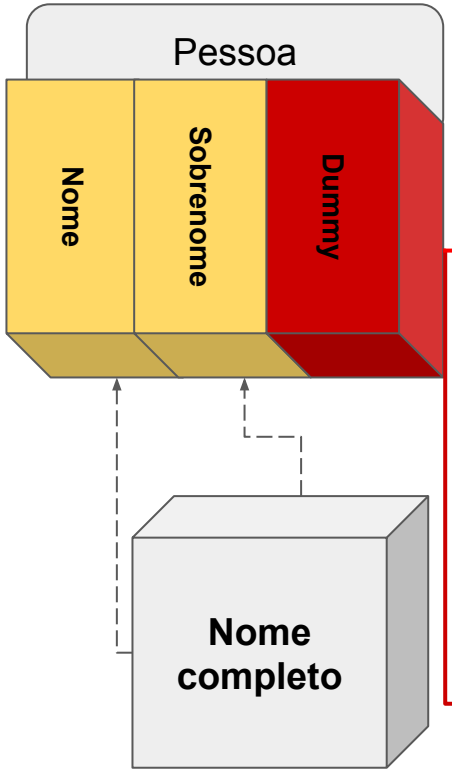

```python
class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade

    @property
    def nome_completo(self):
        return f'{self.nome} {self.sobrenome}'


Pessoa('Eduardo', 'Mendes', ?????)
```

Dummy



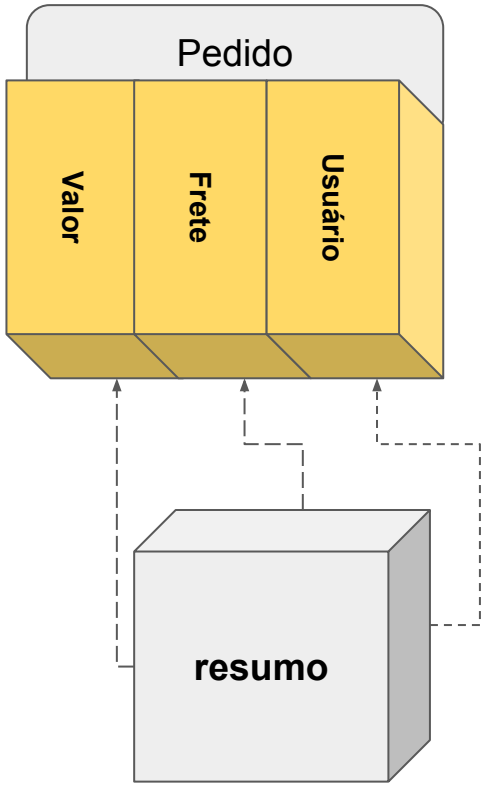

```python
from typing import NewType, Any

Dummy = NewType('Dummy', Any)

class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade

    @property
    def nome_completo(self):
        return f'{self.nome} {self.sobrenome}'


Pessoa('Eduardo', 'Mendes', Dummy(25))
```

Dummy




```python
from typing import NewType, Any

Dummy = NewType('Dummy', Any)

class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade

    @property
    def nome_completo(self):
        return f'{self.nome} {self.sobrenome}'


Pessoa('Eduardo', 'Mendes', Dummy(25))
```

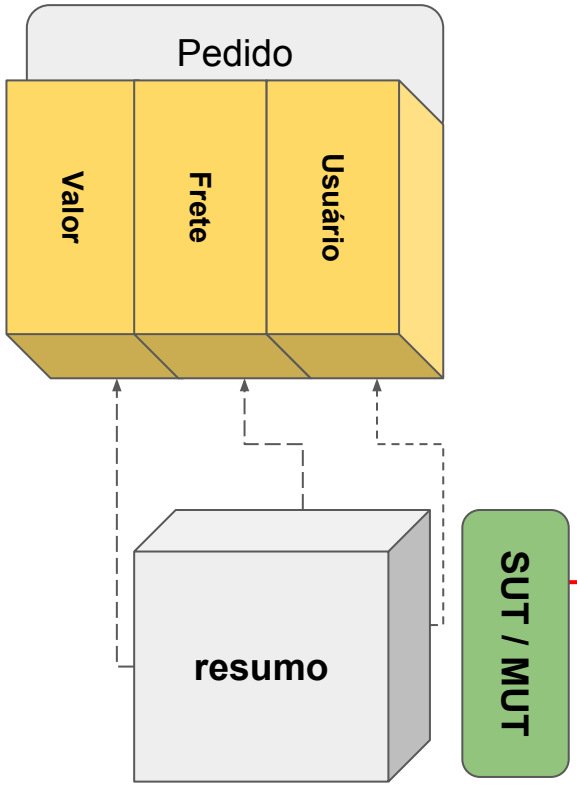
Fake



O fake é usado quando temos um módulo acoplado (DOC) e esse módulo é realmente chamado de alguma maneira.

Então o Fake tem que se comportar como o objeto, com a mesma API, na chamada do SUT.

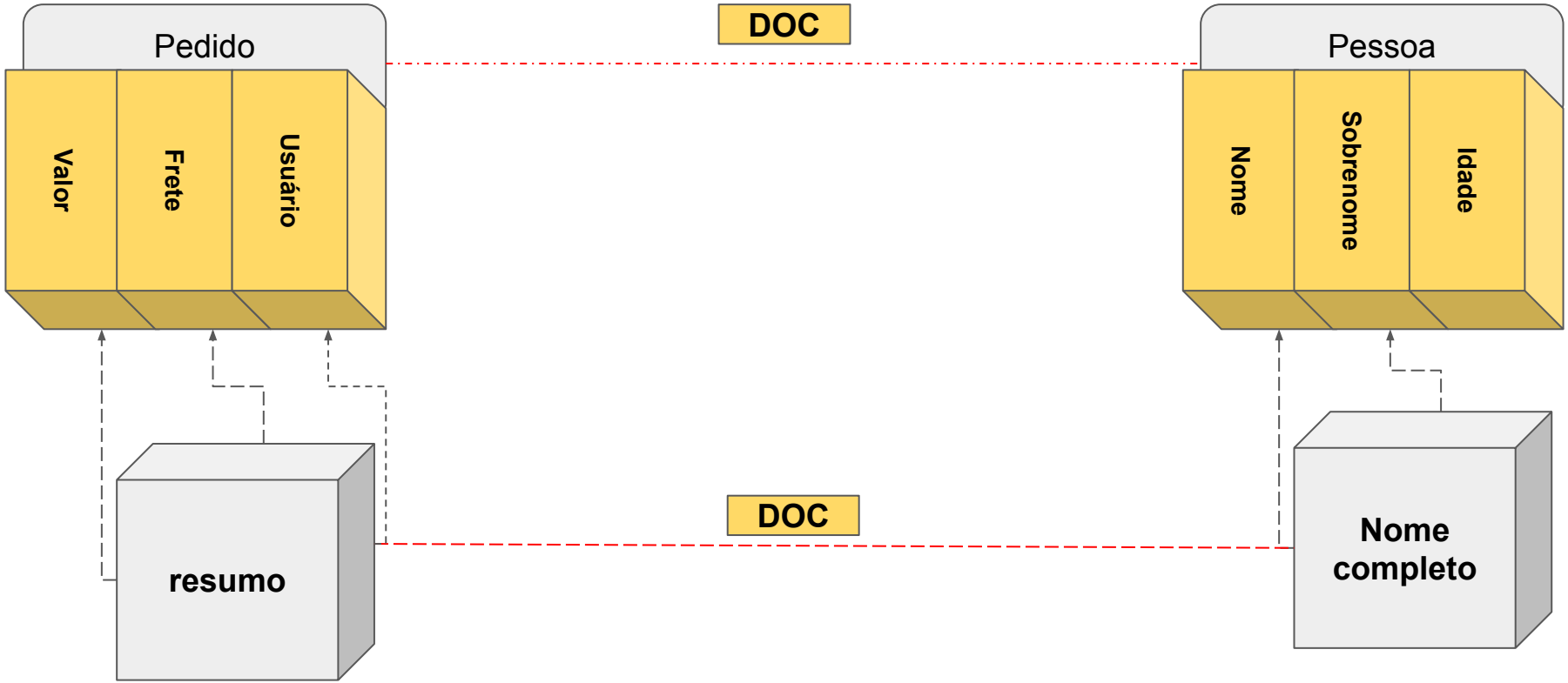
Fake

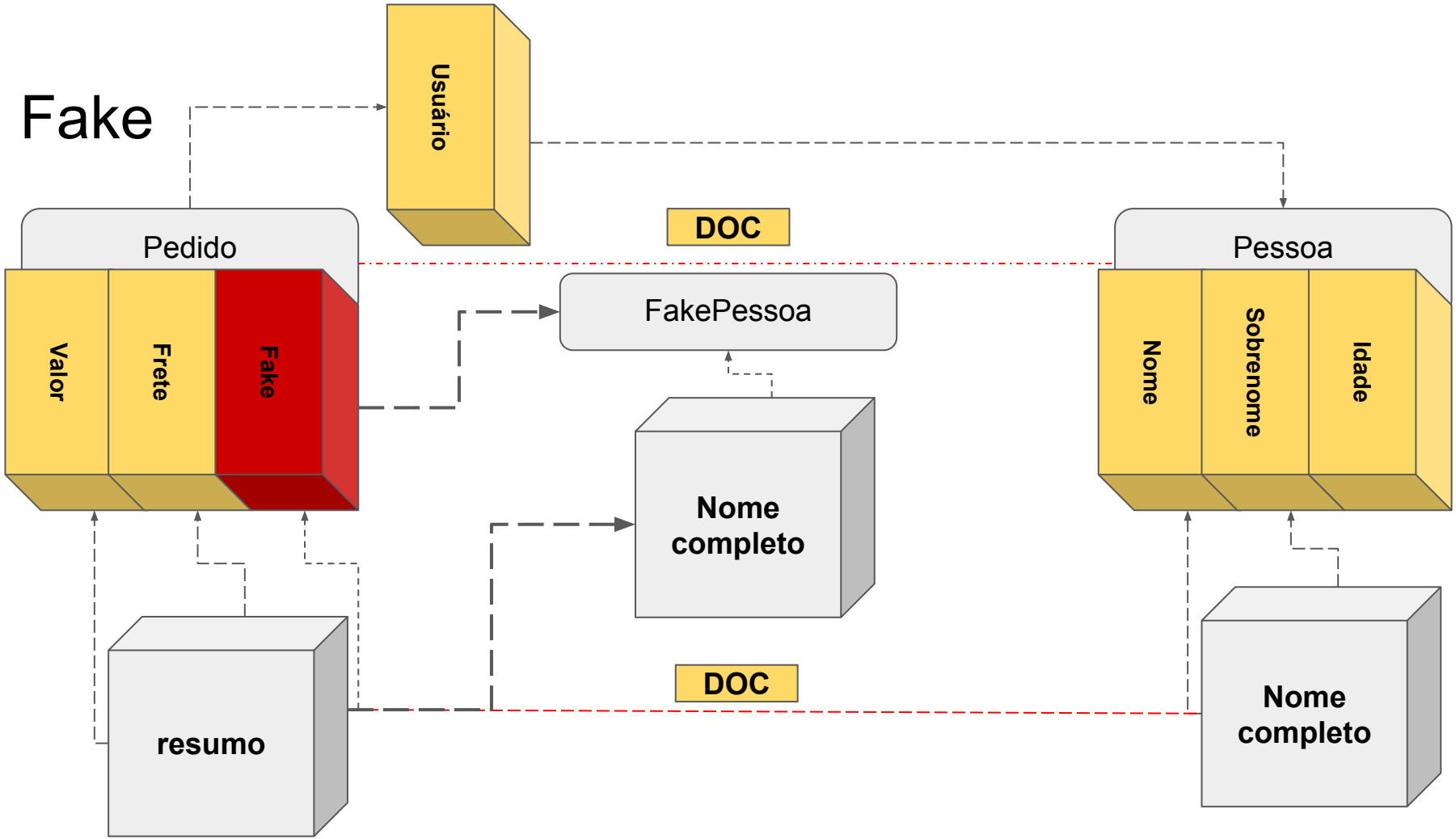



```python
class Pedido:
    def __init__(self, valor, frete, usuario):
        self.valor = valor
        self.frete = frete
        self.usuario = usuario

    @property
    def resumo(self):
        """Informações gerais sobre o pedido."""
        return f'''
        Pedido por: {self.usuario.nome_completo}
        Valor: {self.valor}
        Frete: {self.frete}
        '''
```

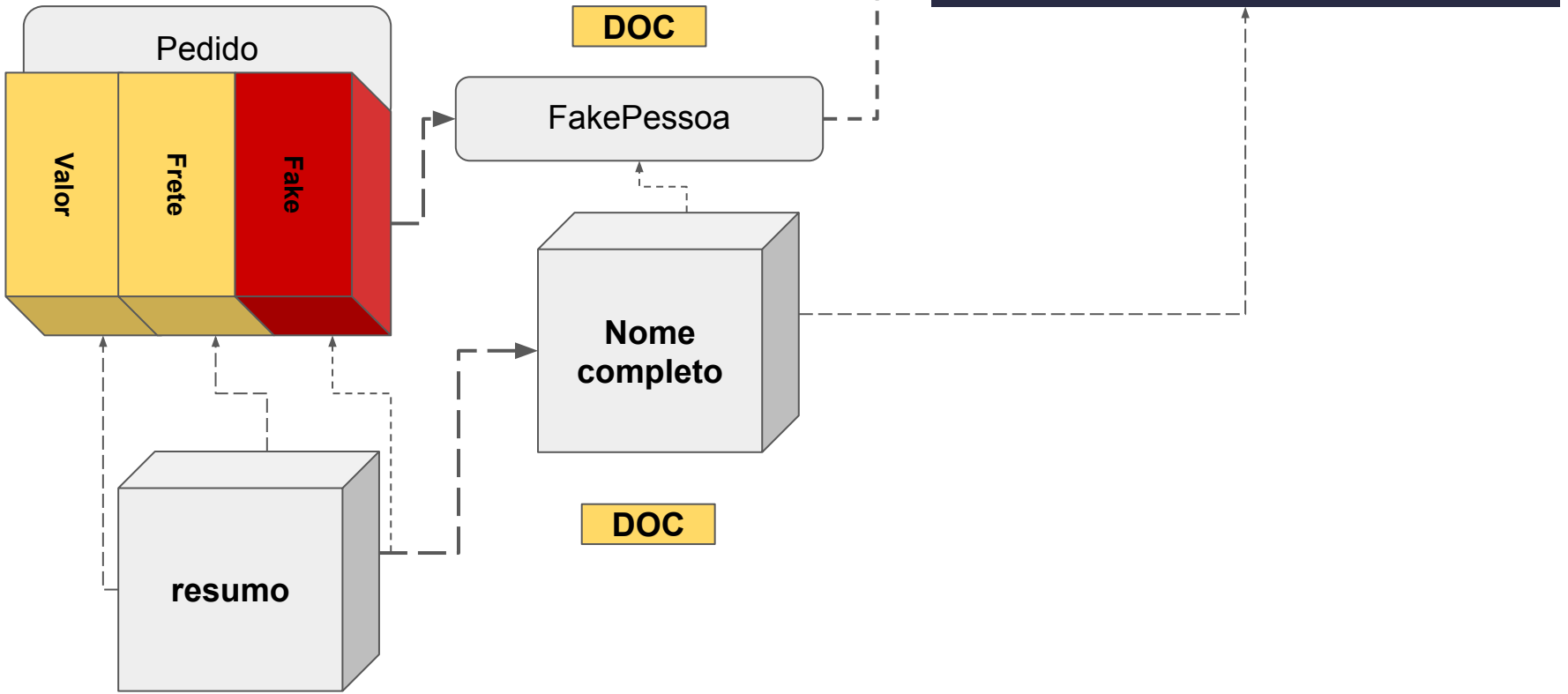
DOC

Fake


Pedido
Valor
Frete
Usuário
DOC
Pessoa
Nome
Sobrenome
Idade
resumo
DOC
Nome completo

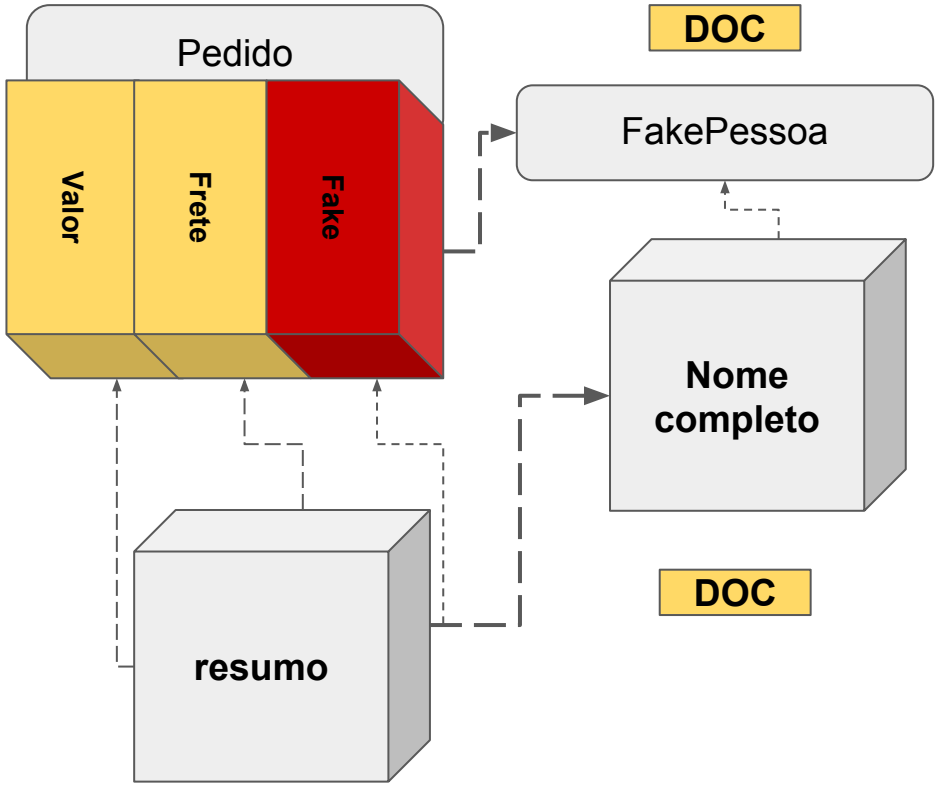

Fake
Usuário
Pedido
Valor
Frete
Fake
DOC
Pessoa
Nome
Sobrenome
Idade
FakePessoa
Nome completo
resumo
DOC
Nome completo

Fake



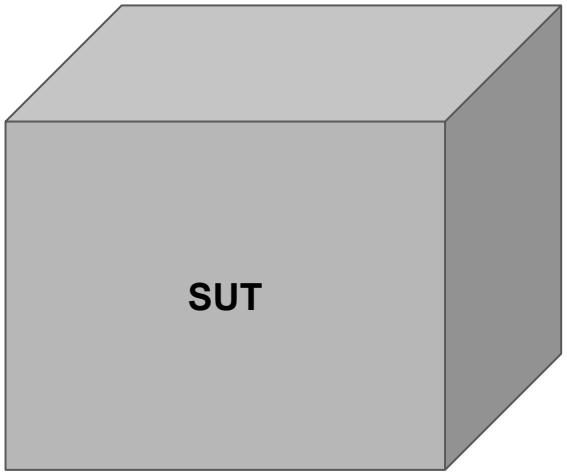

```python
class FakePessoa:
    @property
    def nome_completo(self):
        return 'Eduardo Mendes'
```

Fake




```python
class Pedido:
    def __init__(self, valor, frete, usuario):
        self.valor = valor
        self.frete = frete
        self.usuario = usuario

    @property
    def resumo(self):
        """Informações gerais sobre o pedido."""
        return f'''
        Pedido por: {self.usuario.nome_completo}
        Valor: {self.valor}
        Frete: {self.frete}
        '''


class FakePessoa:
    @property
    def nome_completo(self):
        return 'Eduardo Mendes'


Pedido(100.00, 13.00, FakePessoa()).resumo
```

Spy



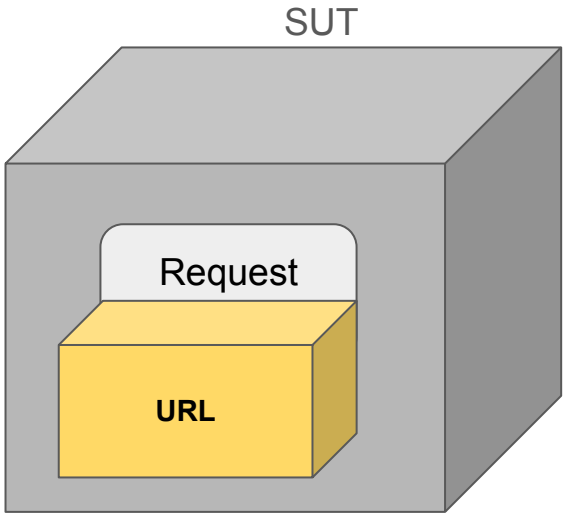
Spies são a KGB dos dublês. A ideia principal deles é saber quando o DOC foi chamado, com que valores ele foi chamado, quantas vezes foi chamado...

Você entendeu...

“Procedural Behavior Verification” é o termo técnico

A função deles é exercitar os inputs indiretos e validar se o DOC foi de fato executado

Spy



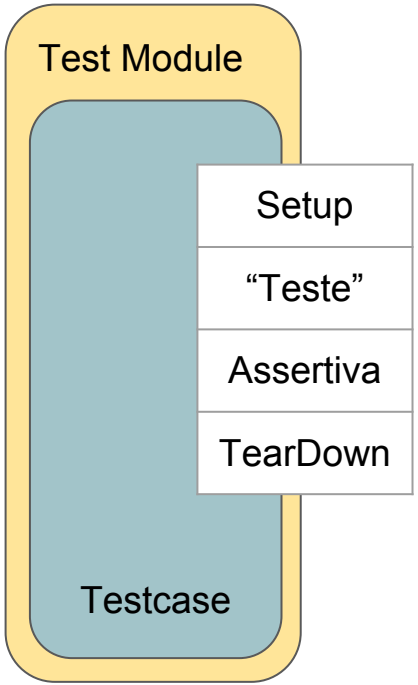
Spies são a KGB dos dublês. A ideia principal deles é saber quando o DOC foi chamado, com que valores ele foi chamado, quantas vezes foi chamado...

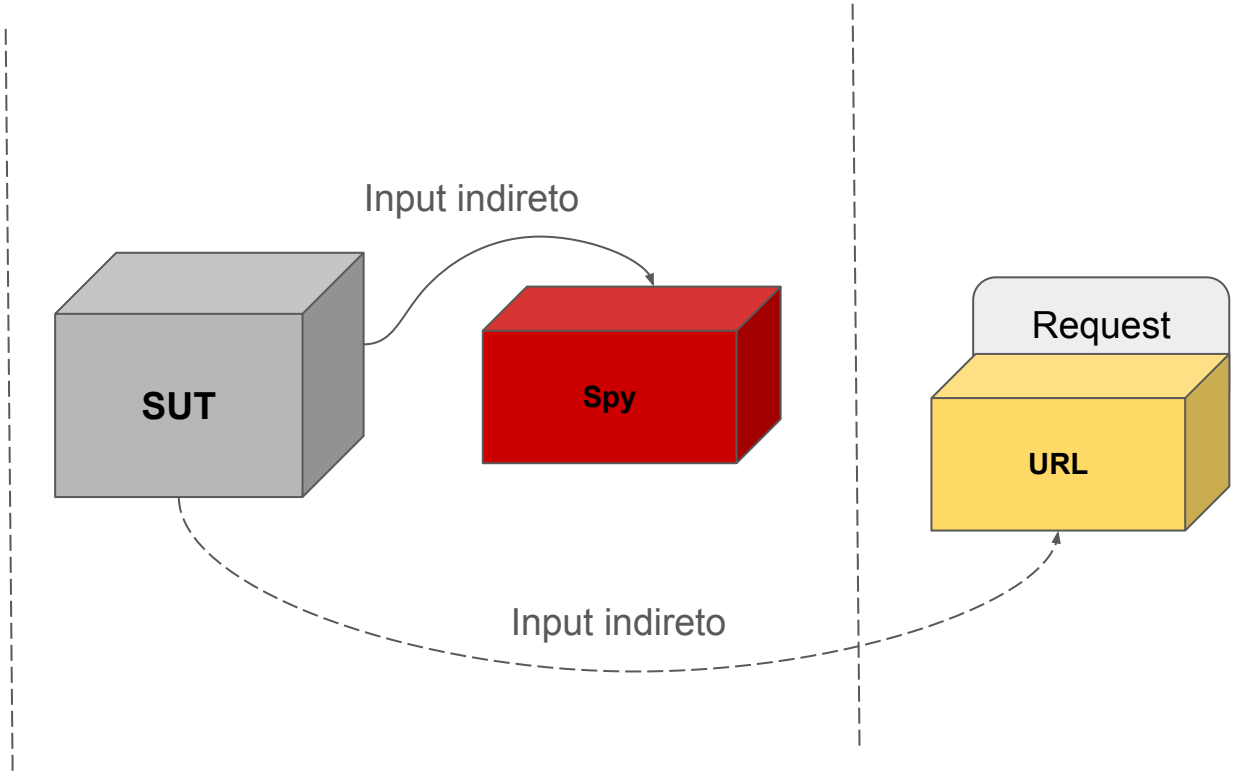
Você entendeu...

“Procedural Behavior Verification” é o termo técnico

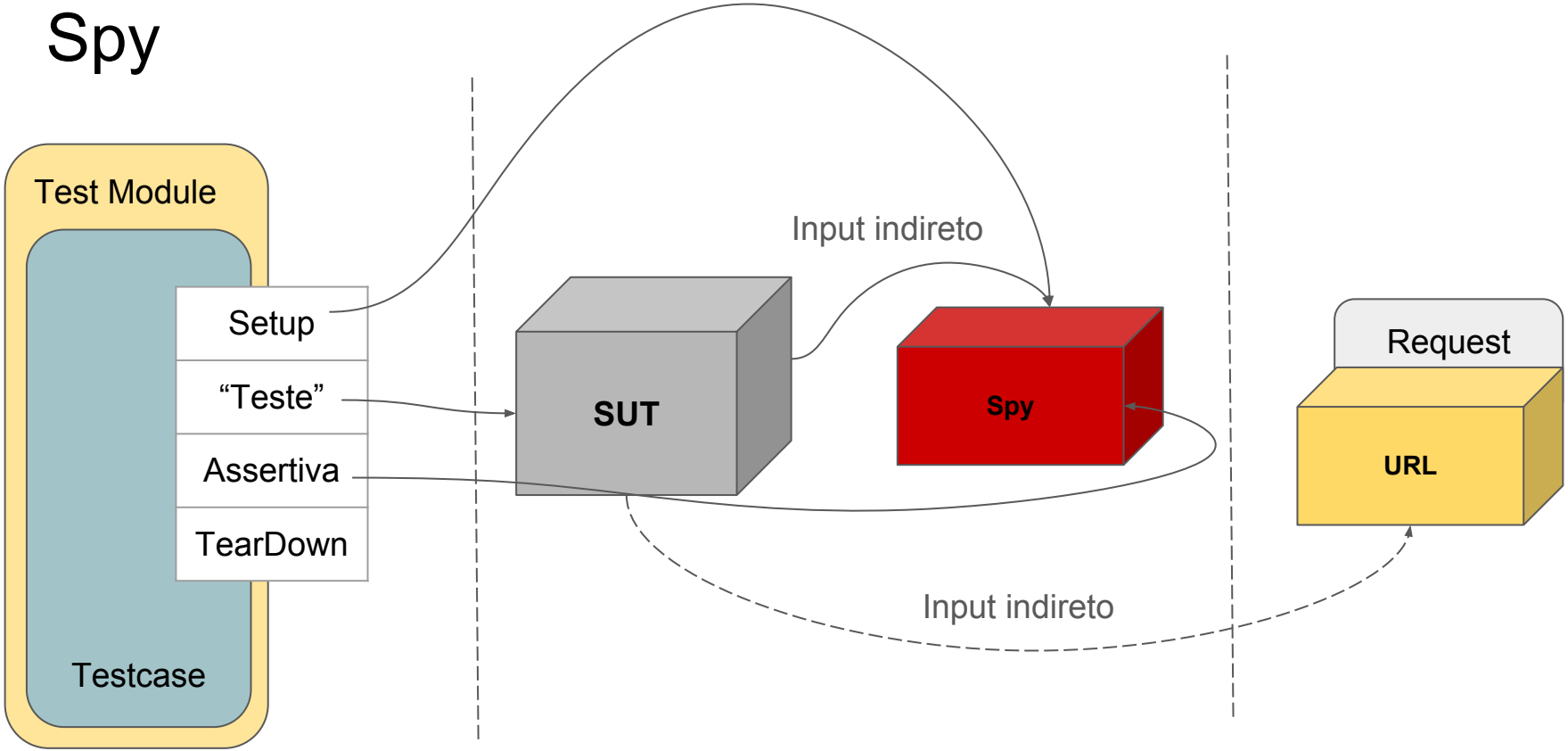
A função deles é exercitar os inputs indiretos e validar se o DOC foi de fato executado

Spy


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase


Input indireto
SUT
Spy
Request
URL
Input indireto

Spy


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Input indireto
Spy
Request
URL
Input indireto

Spy

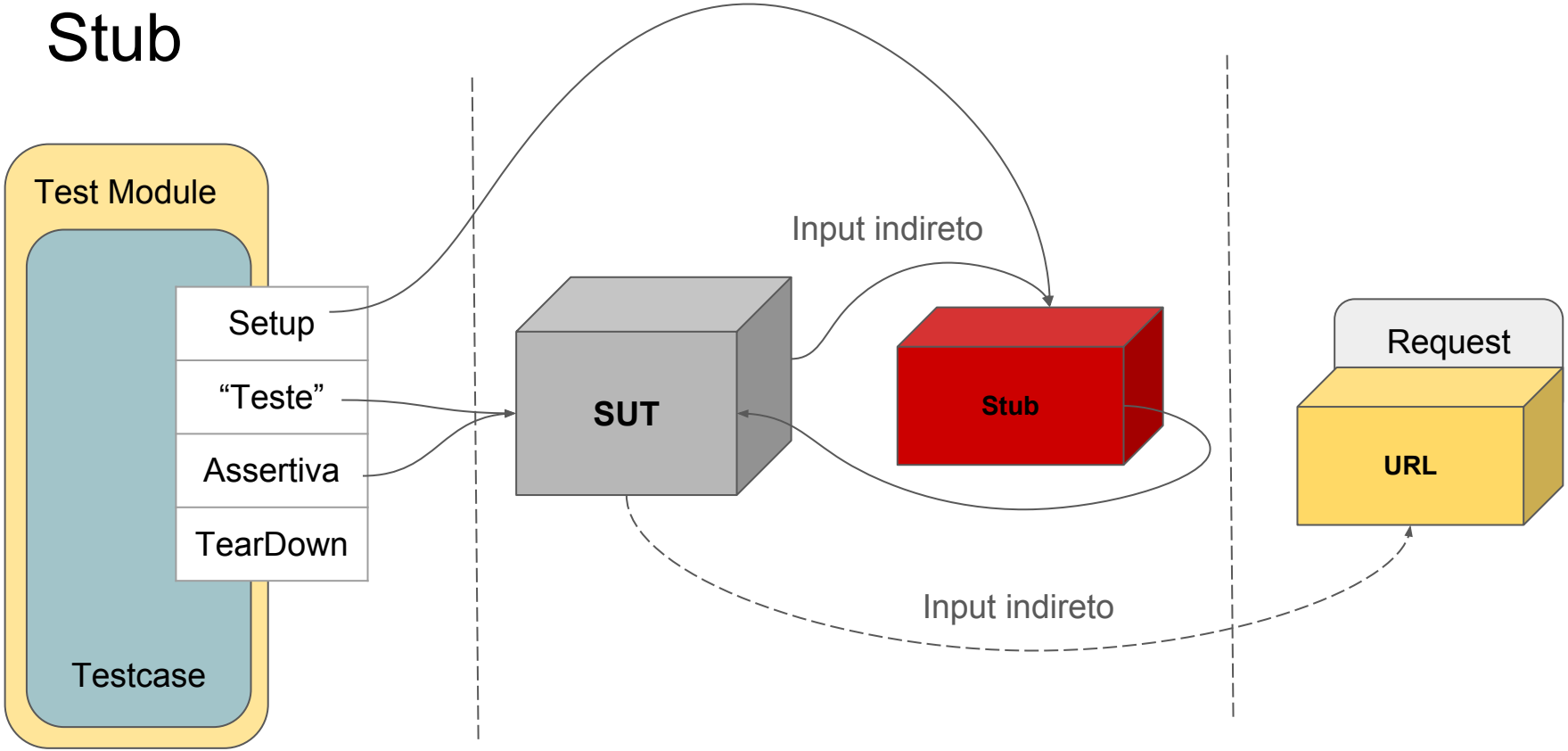
```python
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

```python
class TestPageContent(TestCase):
    def test_page_content_deve_ser_chamada_com_http(self):

        with mock.patch('acomplamento_exemplo.get') as spy:
            page_content('bababa', False)

        spy.assert_called_with('http://bababa', None)
```

Stub

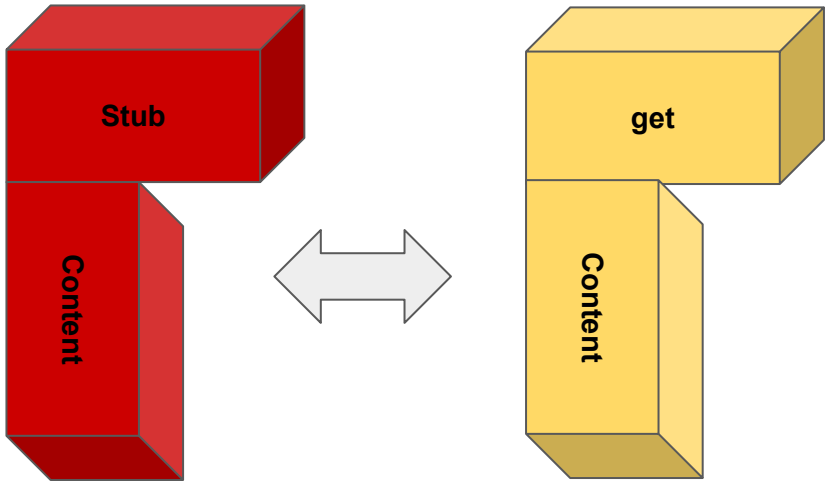
Stub é um objeto que armazena dados predefinidos e os utiliza para atender chamadas durante os testes. Ele é usado quando não podemos ou não queremos envolver objetos que respondam com dados reais ou que tenham efeitos colaterais indesejáveis.

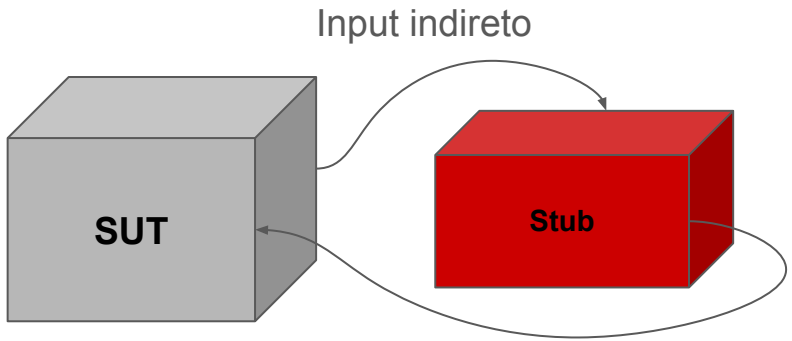
Stub


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Input indireto
Stub
Request
URL
Input indireto

Stub

```python
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

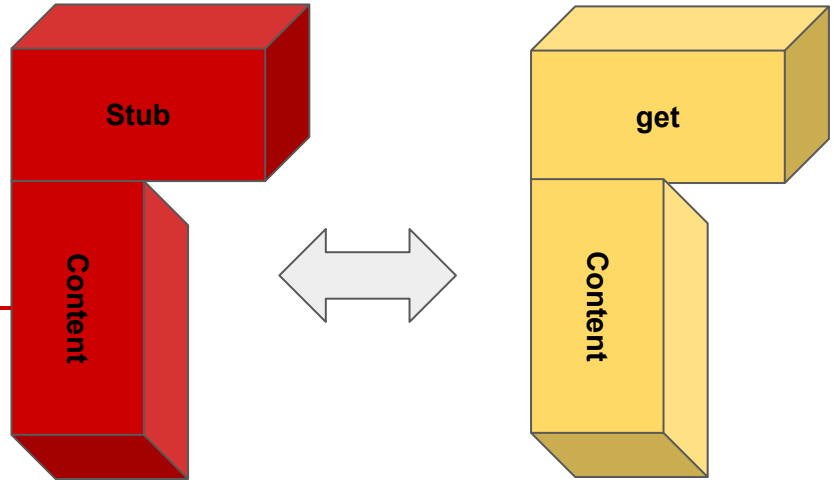




Stub

```python
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

```python
class StubGet:
    @property
    def content(self):
        return b'<html> ... </html>'
```



Stub

```
class StubGet:
    @property
    def content(self):
        return b'<html> ... </html>'
```

```
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

```
class TestPageContent(TestCase):
    def test_page_content_deve_ser_chamada_com_http(self):
        esperado = '<html> ... </html>'

        with mock.patch('acomplamento_exemplo.get', return_value=StubGet()):
            result = page_content('bababa', False)

        self.assertEqual(esperado, result)
```

Mock

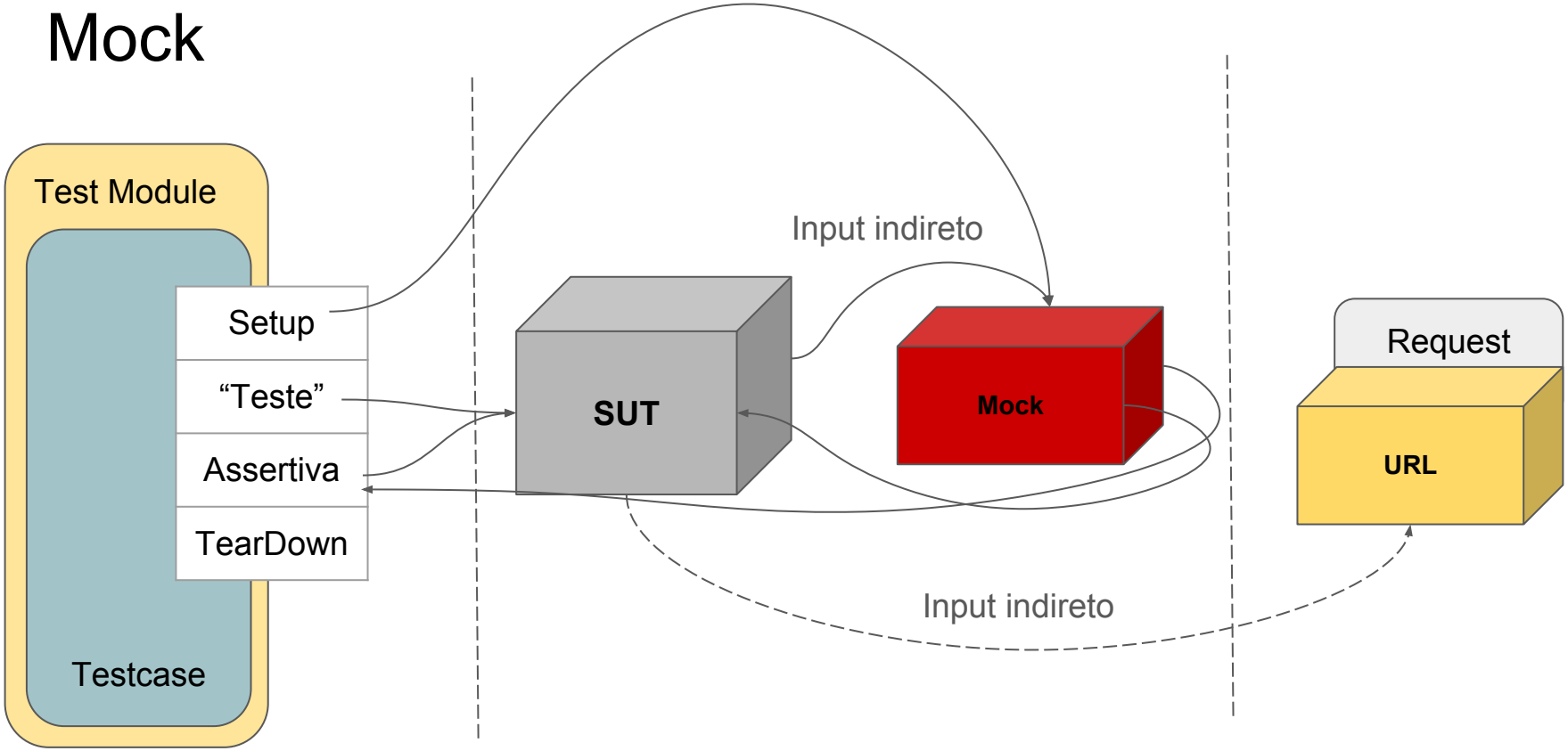
Os mocks são uma loucura que só. Sério, eles são de mais. Ninguém pode dizer o contrário.

Vamos esquecer da teoria por um pequeno minuto.

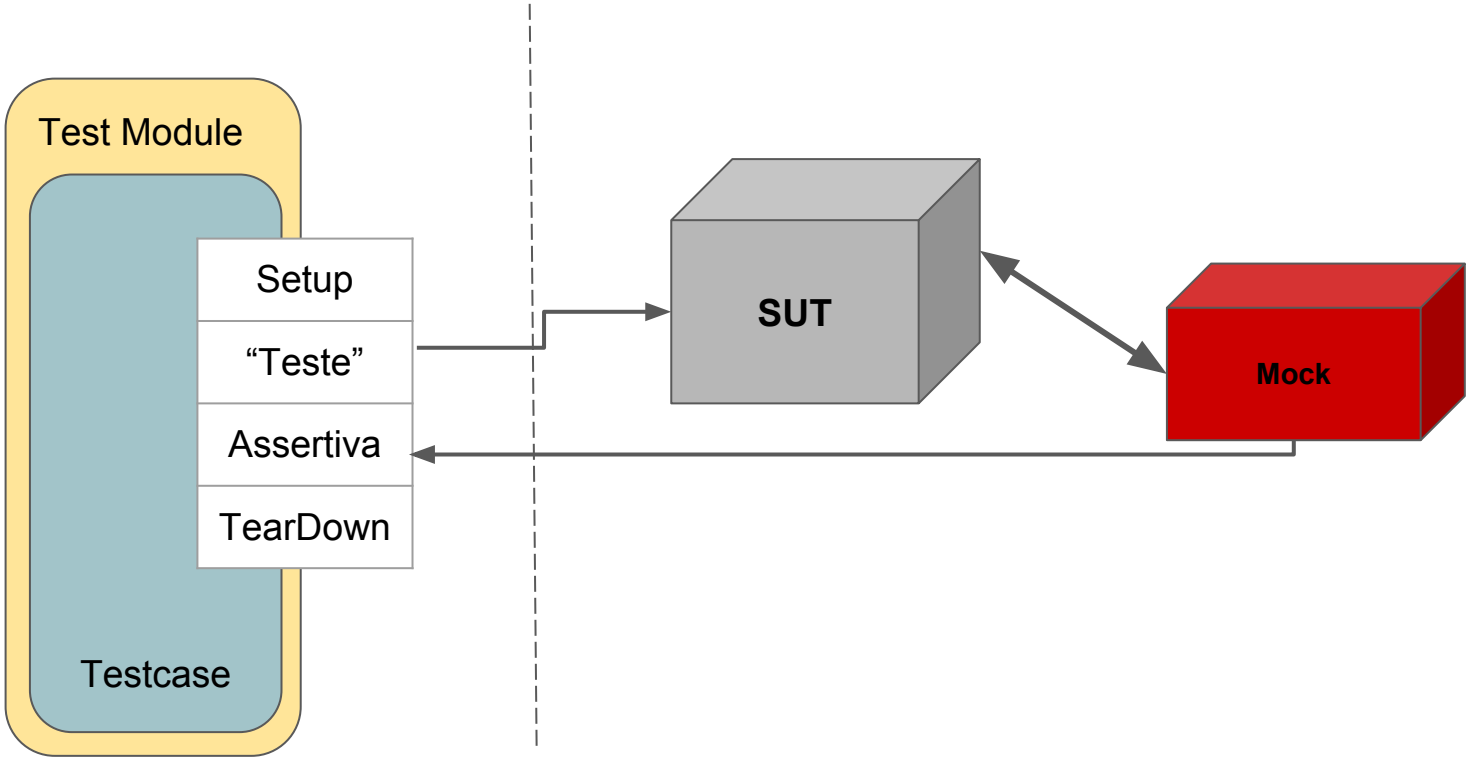
Mocks controlam o fluxo, se colocam no lugar de outras coisas, e ainda pode checar seus resultados.

Ele é um Stub e um Spy ao mesmo tempo.

Mock

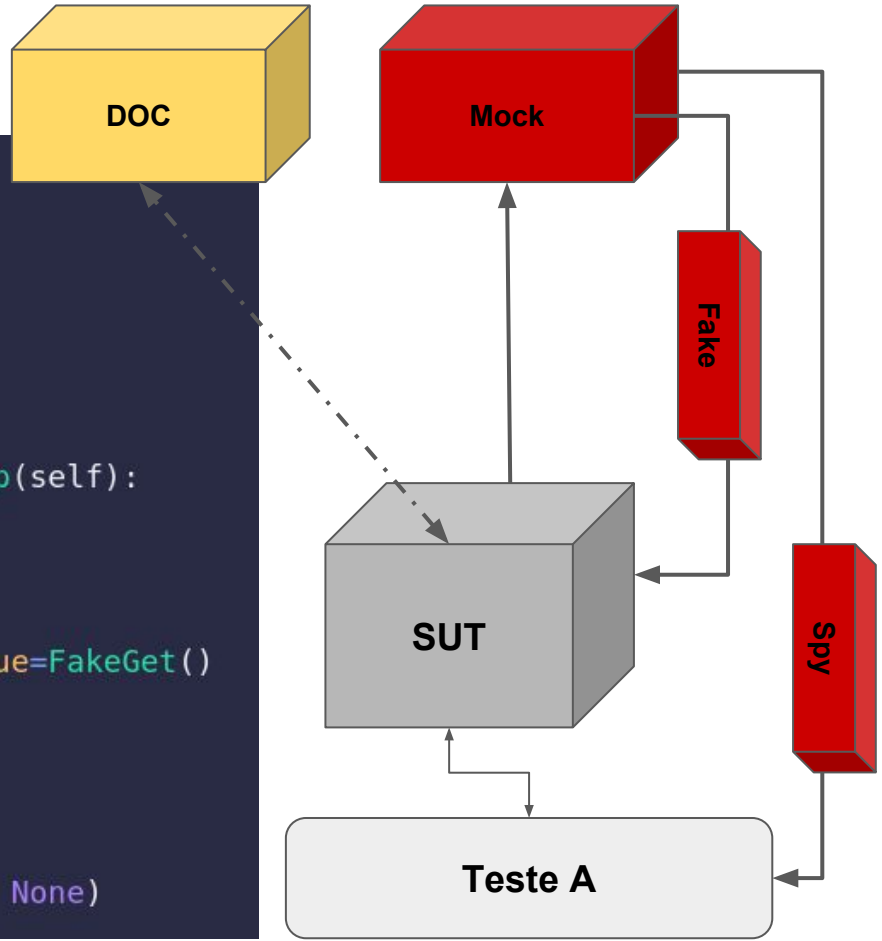

Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Input indireto
Mock
Request
URL
Input indireto

Mock


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Mock

Mock

```python
class FakeGet:
    @property
    def content(self):
        return b'<html> ... </html>'


class TestPageContent(TestCase):
    def test_page_content_deve_ser_chamada_com_http(self):
        esperado = '<html> ... </html>'

        with patch(
            'acomplamento_exemplo.get', return_value=FakeGet()
        ) as mocked:
            result = page_content('bababa', False)

        self.assertEqual(esperado, result)
        mocked.assert_called_with('http://bababa', None)
```



Mock

Referências

1. Testes Unitários 101: Mocks, Stubs, Spies e todas essas palavras difíceis
2. xUnit Test Patterns: Refactoring test code - Gerand Meszaros (2007)