12

Para saber mais

Diferentemente de testes de unidade clássicos, os testes com dublês têm o foco em verificação de **comportamento**, enquanto os testes clássicos fazem a verificação de **estado**. Vamos ver a diferença?

`Pedido` é a unidade sob teste (SUT – system under test), e `Estoque` é um colaborador.

Exemplo de testes clássicos:

```
def test_pedido_pode_ser_atendido_se_existe_estoque():
    estoque = Estoque()
    estoque.aumentar("Produto A", 5)

    pedido = Pedido("Produto A", 5)
    atendido = pedido.atender(estoque)

    assert atendido is True
    assert 0 == estoque.quantidade("Produto A")


def test_pedido_nao_diminui_o_estoque_se_nao_pode_ser_atendido():
    estoque = Estoque()
    estoque.aumentar("Produto A", 1)

    pedido = Pedido("Produto A", 5)
    atendido = pedido.atender(estoque)

    assert atendido is False
    assert 1 == estoque.quantidade("Produto A")
```

A verificação de estado é verificar os estados do SUT e de seus colaboradores depois do atributo (função) ter sido exercitado (executado no caso de teste). No caso, comparar os valores: a quantidade do estoque no final.

Em Python dizemos que tudo é objeto, inclusive as funções. É por este motivo que conseguimos com que os dublês sejam aplicados nas funções. Usando a função `patch` conseguimos identificar a função ou o objeto que queremos substituir pelo dublê. No entanto, quando não é o caso de uma função, mas, sim, um objeto, e ele é uma dependência injetada (ou seja, é passado como parâmetro), não precisamos da função `patch` para embutir o dublê. Para estes casos, criamos um objeto que representa o dublê e o usamos.

```
from unittest.mock import MagicMock, Mock, patch


class Pedido:
    def __init__(self, produto, qtd):
        self.produto = produto
        self.qtd = qtd

    def atender(self, estoque):
        try:
            estoque.diminuir(self.produto, self.qtd)
```

```
        except SemEstoque:
            return False
        else:
            return True


class SemEstoque(Exception):
    pass


class Estoque:
    def __init__(self):
        self.produtos = {}

    def aumentar(self, produto, qtd):
        self.produtos[produto] = self.produtos.get(produto, 0) + qtd

    def diminuir(self, produto, qtd):
        if self.quantidade(produto) >= qtd:
            self.produtos[produto] -= qtd
            return True
        raise SemEstoque

    def quantidade(self, produto):
        if produto in self.produtos.keys():
            return self.produtos[produto]
        return 0
```

Já na verificação de comportamento, é verificada a interação entre o SUT e os colaboradores. Então, a verificação foca em se foi possível diminuir o estoque.

```
def test_pedido_pode_ser_atendido_se_existe_estoque():
    estoque = Mock()
    estoque.diminuir.return_value = True

    pedido = Pedido("Produto A", 5)
    atendido = pedido.atender(estoque)

    assert atendido is True
    estoque.diminuir.assert_called_once_with("Produto A", 5)


def test_pedido_nao_diminui_o_estoque_se_nao_pode_ser_atendido():
    estoque = Mock()
    estoque.diminuir.side_effect = SemEstoque()
    estoque.quantidade.return_value = 1

    pedido = Pedido("Produto A", 5)
    atendido = pedido.atender(estoque)

    estoque.diminuir.assert_called_once_with("Produto A", 5)
    assert atendido is False
```

Usar ou não dublês tem suas vantagens e desvantagens. Por um lado, o uso de dublês foca na unidade sob teste, isolando-a de qualquer outra unidade. Mas, por outro lado, como os colaboradores reais não estão sendo testados, e por

isso é necessário que eles mesmos sejam, em algum momento, as unidades sob teste. Quando usamos testes de unidade sem os dublês, de alguma forma, estamos também testando seus colaboradores.

Você pode ler Mocks are not Stubs (https://martinfowler.com/articles/mocksArentStubs.html) de Martin Fowler. Nessa mesma página, você encontrará o link para a versão em Português. Sobre este mesmo tema, leia está página Test Double (https://github.com/testdouble/contributing-tests/wiki/Test-Double), que conta sobre duas escolas de TDD (Desenvolvimento guiado por testes): Detroit (TDD clássico) e London (uso de dublês de testes como padrão).