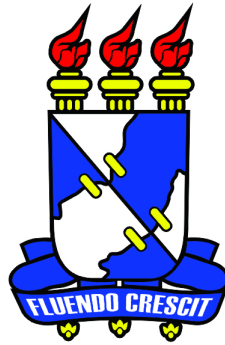


UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
TESTE DE SOFTWARE



ANDREY FELLIPE ALMEIDA CHAGAS

Atividade 1 – Testes Unitários e o Stack Overflow

SÃO CRISTÓVÃO - SE
2023

ANDREY FELLIPE ALMEIDA CHAGAS

Atividade 1 – Testes Unitários e o Stack Overflow

Atividade 1 da primeira unidade da disciplina de Teste de Software, ministrada pelo Professor Glauco de Figueiredo Carneiro.

SÃO CRISTÓVÃO - SE
2023

1. Escolha do Problema.....	4
2. Replicação do Problema.....	4
Problema Identificado.....	5
3. Solução Proposta.....	6
Alterações Realizadas.....	6
Código Main.java Refatorado.....	6
Explicação das Alterações:.....	7
Código FooTest.java Refatorado.....	7
Explicação das Alterações:.....	8
4. Execução Correta.....	9
Explicação da Execução Correta:.....	9
5. Respostas Não Adotadas Pelo Stack.....	9
Resposta 1:.....	10
Resposta 2:.....	11
Resposta 3:.....	12
Resposta 4:.....	13
Resposta 5:.....	14
Resposta 6:.....	15
Resposta 7:.....	16

1. Escolha do Problema

O primeiro passo foi escolher um problema no [Stackoverflow](https://stackoverflow.com). Utilizando a palavra-chave “unit-testing” para encontrar problemas, foi escolhido um problema específico para análise e resolução. O problema selecionado é o seguinte:

The screenshot shows a Stack Overflow question page. The question is titled "Mockito : how to verify method was called on an object created within a method?". It was asked 12 years, 4 months ago, modified 7 months ago, and has been viewed 991k times. The questioner is new to Mockito and asks how to verify that a method named `someMethod` was invoked exactly once after a method named `foo` was invoked. The code snippet provided is:

```
public class Foo {
    public void foo(){
        Bar bar = new Bar();
        bar.someMethod();
    }
}
```

The questioner wants to make the following verification call:

```
verify(bar, times(1)).someMethod();
```

where `bar` is a mocked instance of `Bar`. The tags are `java`, `unit-testing`, `junit`, and `mockito`. The question was asked on Mar 23, 2012 at 15:09 by user `mre` (44k reputation, 33 answers, 121 votes, 170 views). The question is featured on Meta and has a list of linked questions on the right.

2. Replicação do Problema

Dado o problema selecionado, onde o usuário não conseguiu usar o Mockito para verificar se o método `someMethod` foi invocado exatamente uma vez após a chamada do método `foo`, o próximo passo foi replicar o problema na IDE utilizada, que é o IntelliJ IDEA, criando um projeto utilizando o Maven.

```
1 package com.exemplo;
2
3 class Bar {
4     public void someMethod() {
5         // Método que será testado
6     }
7 }
8
9 class Foo {
10    public void foo() {
11        Bar bar = new Bar();
12        bar.someMethod();
13    }
14 }
15
```

Disponível em: [Repositório do Github](#) na branch [CodigoComErro](#).

```
1 package com.exemplo;
2
3 import static org.mockito.Mockito.*;
4
5 import org.junit.Test;
6
7 public class FooTest {
8
9     @Test
10    public void testFoo() {
11        Bar bar = mock(Bar.class);
12        Foo foo = new Foo();
13
14        foo.foo();
15
16        verify(bar, times(wantedNumberOfInvocations: 1)).someMethod(); // Este teste vai falhar
17    }
18 }
19
```

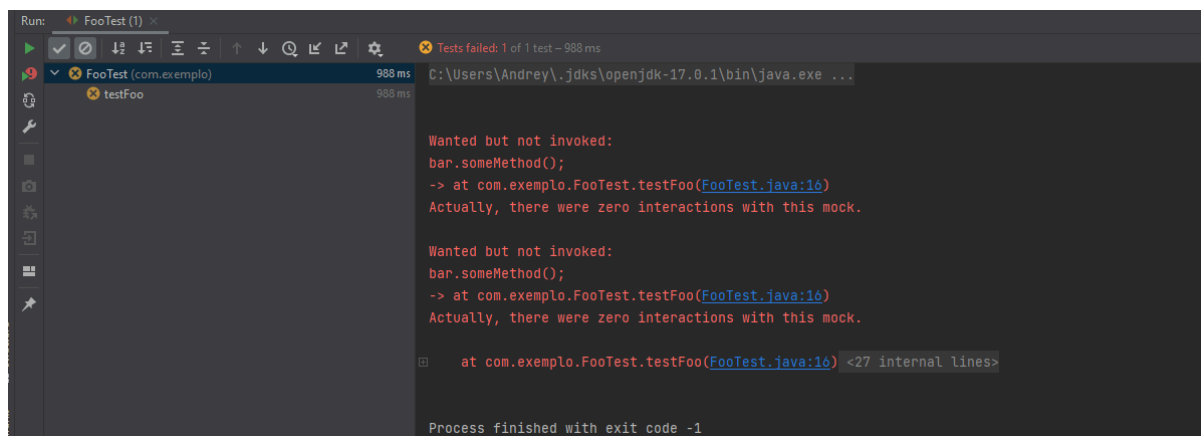
Disponível em: [Repositório do Github](#) na branch [CodigoComErro](#).

Problema Identificado

O código com erro mostrava que o usuário estava tentando usar o Mockito para verificar se o método `someMethod` de uma instância de `Bar` foi chamado exatamente uma vez após a invocação do método `foo` da classe `Foo`. No entanto, o código original criava uma nova instância de `Bar` diretamente dentro do método `foo`:

```
Bar bar = new Bar();
bar.someMethod();
```

Isso impedia o uso de mocks, pois a instância criada no método `foo` não podia ser substituída por um mock, resultando em falha ao tentar verificar a chamada do método `someMethod`.



O erro exibido indicava que o método `someMethod` não foi chamado no mock, pois o mock não foi utilizado na execução do código.

3. Solução Proposta

A resposta aceita no StackOverflow sugeriu que o problema poderia ser corrigido injetando a instância de `Bar` diretamente ou utilizando uma fábrica para criar a instância de `Bar`. Com isso, seria possível acessar a instância de `Bar` de maneira a permitir a execução do teste com Mockito.

Alterações Realizadas

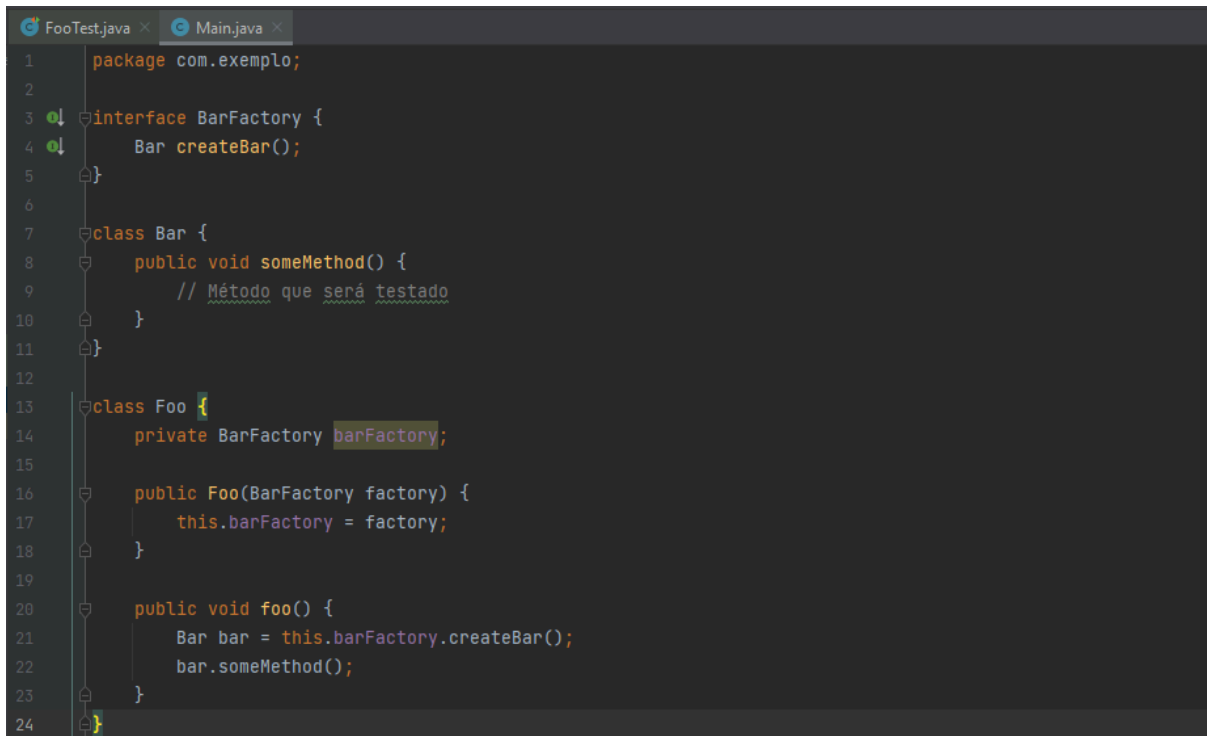
Para corrigir o problema, as seguintes alterações foram feitas:

Código `Main.java` Refatorado

O código foi refatorado para permitir a injeção da instância de `Bar` por meio de uma fábrica. As principais alterações foram:

- Introdução da interface `BarFactory` para criar instâncias de `Bar`.

- Alteração na classe `Foo` para usar a fábrica em vez de criar a instância de `Bar` diretamente.



```
1 package com.exemplo;
2
3 interface BarFactory {
4     Bar createBar();
5 }
6
7 class Bar {
8     public void someMethod() {
9         // Método que será testado
10    }
11 }
12
13 class Foo {
14     private BarFactory barFactory;
15
16     public Foo(BarFactory factory) {
17         this.barFactory = factory;
18     }
19
20     public void foo() {
21         Bar bar = this.barFactory.createBar();
22         bar.someMethod();
23     }
24 }
```

Disponível em: [Repositório do Github](#) na branch [main](#).

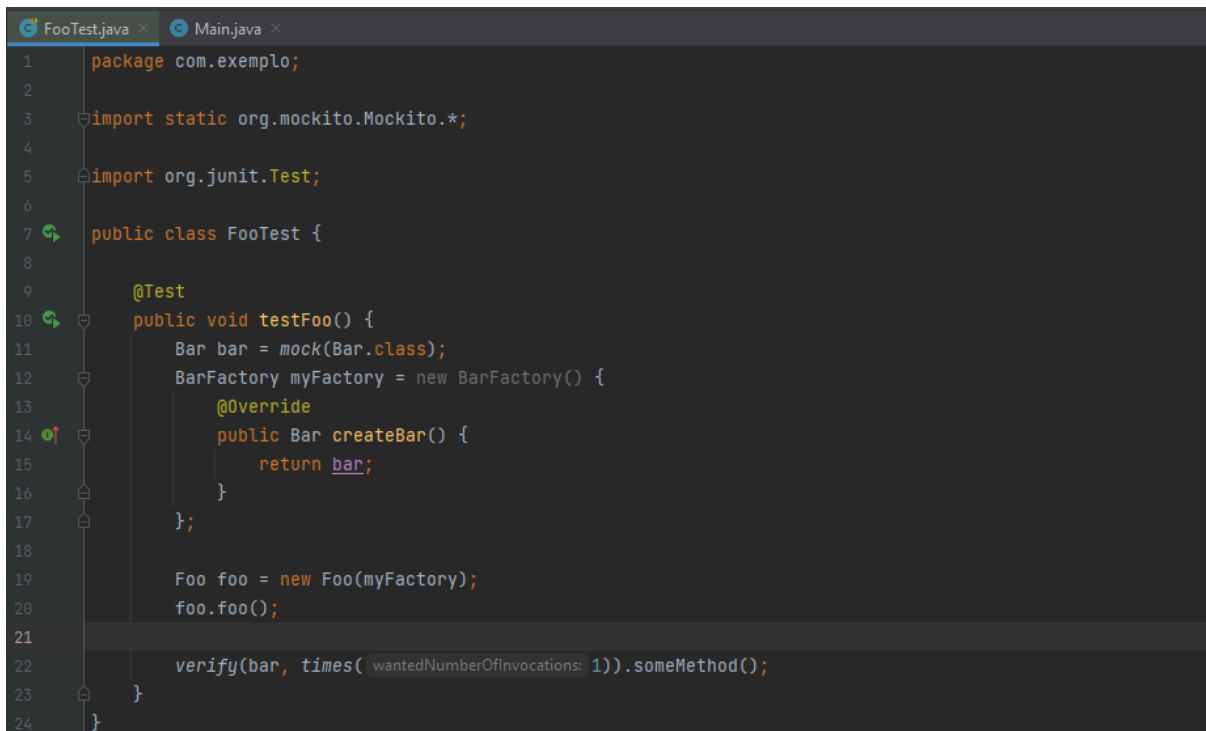
Explicação das Alterações:

A introdução da interface `BarFactory` e a modificação na classe `Foo` permitem que a instância de `Bar` seja fornecida externamente. Isso facilita o uso de mocks em testes, pois a instância de `Bar` criada dentro do método `foo` é agora substituída pela instância fornecida pela fábrica.

Código `FooTest.java` Refatorado

O teste foi atualizado para criar um mock de `Bar` e uma implementação da fábrica que retorna esse mock. As principais alterações foram:

- Criação de um mock para a classe **Bar**.
- Implementação da interface **BarFactory** para retornar o mock.
- Verificação da chamada ao método **someMethod** no mock.



```
1 package com.exemplo;
2
3 import static org.mockito.Mockito.*;
4
5 import org.junit.Test;
6
7 public class FooTest {
8
9     @Test
10    public void testFoo() {
11        Bar bar = mock(Bar.class);
12        BarFactory myFactory = new BarFactory() {
13            @Override
14            public Bar createBar() {
15                return bar;
16            }
17        };
18
19        Foo foo = new Foo(myFactory);
20        foo.foo();
21
22        verify(bar, times(wantedNumberOfInvocations: 1)).someMethod();
23    }
24 }
```

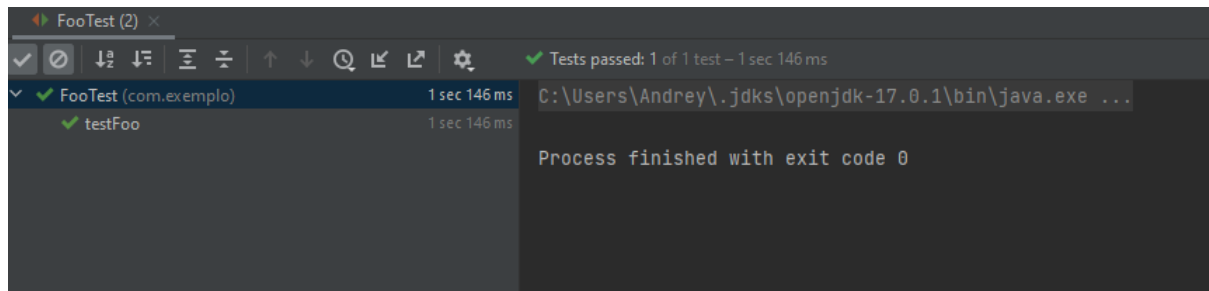
Disponível em: [Repositório do Github](#) na branch [main](#).

Explicação das Alterações:

O teste foi ajustado para utilizar um mock da classe **Bar** em vez de uma instância real. Isso foi possível devido à introdução da interface **BarFactory**, que permite fornecer o mock em vez da instância real. Assim, o teste agora verifica corretamente se o método **someMethod** foi chamado uma vez.

4. Execução Correta

Após refatorar o código e os testes, a execução foi realizada com sucesso, demonstrando que o método `someMethod` foi chamado exatamente uma vez conforme esperado.



Explicação da Execução Correta:

A execução bem-sucedida indica que as alterações realizadas permitiram que o Mockito verificasse corretamente a chamada ao método `someMethod`. O código refatorado agora permite a substituição da instância real de `Bar` por um mock, tornando possível a verificação da chamada do método em testes unitários.

5. Respostas Não Adotadas Pelo Stack

O Stack Overflow não adota algumas respostas por uma série de razões, como a falta de alinhamento com a necessidade específica do problema ou a complexidade desnecessária das soluções propostas. Abaixo, explico os motivos pelos quais outras respostas para o problema selecionado não foram aceitas e a solução escolhida foi preferida:

Resposta 1:

I think Mockito `@InjectMocks` is the way to go.

Depending on your intention you can use:

1. Constructor injection
2. Property setter injection
3. Field injection

More info in [docs](#)

Below is an example with field injection:

Classes:

```
public class Foo
{
    private Bar bar = new Bar();

    public void foo()
    {
        bar.someMethod();
    }
}

public class Bar
{
    public void someMethod()
    {
        //something
    }
}
```

Test:

```
@RunWith(MockitoJUnitRunner.class)
public class FooTest
{
    @Mock
    Bar bar;

    @InjectMocks
    Foo foo;

    @Test
    public void FooTest()
    {
        doNothing().when( bar ).someMethod();
        foo.foo();
        verify(bar, times(1)).someMethod();
    }
}
```


Proposta:

- O uso de `@InjectMocks` para injeção automática de dependências e o exemplo fornecido com injeção de campos e métodos.

Razão para não adoção:

- A classe `Foo` ainda cria uma instância de `Bar` internamente, o que significa que `@InjectMocks` não afeta a criação da instância de `Bar`. Portanto, a solução não resolve o problema de teste com mocks diretamente, pois `Bar` ainda é instanciado internamente no método `foo`.



Resposta 2:


23


The classic response is, "You don't." You test the public API of `Foo`, not its internals.

Is there any behavior of the `Foo` object (or, less good, some other object in the environment) that is affected by `foo()`? If so, test that. And if not, what does the method do?

[Share](#) [Improve this answer](#) [Follow](#)

answered Mar 23, 2012 at 15:16

**Michael Brewer-Davis**
14.2k ● 5 ● 38 ● 49

Proposta:

- A abordagem sugere testar a API pública de `Foo` e não seus detalhes internos, enfatizando a importância de testar o comportamento visível.

Razão para não adoção:

- Embora essa abordagem seja válida, ela não aborda diretamente a necessidade de verificar se `someMethod` é chamado, que é o foco do problema. O comentário se baseia na premissa de que não se deve verificar a implementação interna, mas sim o comportamento do sistema como um todo.

Resposta 3:



If you don't want to use DI or Factories. You can refactor your class in a little tricky way:

20



```
public class Foo {
    private Bar bar;

    public void foo(Bar bar){
        this.bar = (bar != null) ? bar : new Bar();
        bar.someMethod();
        this.bar = null; // for simulating local scope
    }
}
```

And your test class:

```
@RunWith(MockitoJUnitRunner.class)
public class FooTest {
    @Mock Bar barMock;
    Foo foo;

    @Test
    public void testFoo() {
        foo = new Foo();
        foo.foo(barMock);
        verify(barMock, times(1)).someMethod();
    }
}
```

Then the class that is calling your foo method will do it like this:

```
public class thirdClass {

    public void someOtherMethod() {
        Foo myFoo = new Foo();
        myFoo.foo(null);
    }
}
```

As you can see when calling the method this way, you don't need to import the Bar class in any other class that is calling your foo method which is maybe something you want.

Of course the downside is that you are allowing the caller to set the Bar Object.

Hope it helps.

[Share](#) [Improve this answer](#) [Follow](#)

edited Aug 1, 2015 at 0:41

answered Dec 18, 2013 at 22:57



[raspacorp](#)

5,260 ● 11 ● 41 ● 51

Proposta:

- Refatorar a classe `Foo` para aceitar um parâmetro `Bar` e permitir a injeção durante o teste.

Razão para não adoção:

- Esta solução introduz complexidade adicional, permitindo que o parâmetro `Bar` seja alterado, o que pode ser visto como um design menos limpo. Além disso, permite que o chamador defina o objeto `Bar`, o que pode não ser desejável em todas as situações.

Resposta 4:

▲ Solution for your example code using `PowerMockito.whenNew`

11

▼

- mockito-all 1.10.8
- powermock-core 1.6.1
- powermock-module-junit4 1.6.1
- powermock-api-mockito 1.6.1
- junit 4.12

FooTest.java

```
//Both @PrepareForTest and @RunWith are needed for 'whenNew' to work
@RunWith(PowerMockRunner.class)
@PrepareForTest({ Foo.class })
public class FooTest {

    // Class Under Test
    Foo cut;

    @Mock
    Bar barMock;

    @Before
    public void setUp() throws Exception {
        cut = new Foo();
    }

    @After
    public void tearDown() {
        cut = null;
    }

    @Test
    public void testFoo() throws Exception {

        // Setup
        PowerMockito.whenNew(Bar.class).withNoArguments()
            .thenReturn(this.barMock);

        // Test
        cut.foo();

        // Validations
```


Proposta:

- Utilização do PowerMock para interceptar a criação de novas instâncias de `Bar` no construtor de `Foo`.

Razão para não adoção:


- O uso de PowerMock adiciona complexidade e dependência adicional ao projeto. PowerMock é geralmente considerado uma solução de último recurso devido à sua complexidade e ao impacto potencial na manutenção e legibilidade do código.

Resposta 5:




7

Yes, if you really want / need to do it you can use PowerMock. This should be considered a last resort. With PowerMock you can cause it to return a mock from the call to the constructor. Then do the verify on the mock. That said, csturtz's is the "right" answer.




Here is the link to [Mock construction of new objects](#)

 Share Improve this answer Follow

edited Mar 23, 2012 at 16:50

answered Mar 23, 2012 at 16:44



John B
32.9k ● 7 ● 78 ● 98

Proposta:

- Similar ao comentário anterior, mas enfatiza que o uso de PowerMock deve ser considerado uma última alternativa.

Razão para não adoção:

- Embora reconheça que PowerMock pode ser uma solução, reforça que é um método complexo e não ideal, que pode ser evitado com soluções mais simples e diretas.

Resposta 6:



5



I had this very issue today, and I didn't want to use PowerMock or other stuff. I just wanted to make a test that made sure a certain method was called. I found this post and I saw that nobody had mentioned this approach.

One way of achieving this without adding in more dependencies or similar is pretty low tech, but it works:

```
@Test
public void testSomeMethodIsCalledOnce() throws Exception {
    final AtomicInteger counter = new AtomicInteger(0);
    Mockito.when(someObject.theMethodIWant(anyString()))
        .then((Answer<ReturnValue>) __ -> {
            counter.incrementAndGet();
            return theExpectedAnswer;
        });
    theObjectUnderTest.theMethod(someTestValue);

    assertEquals(1, counter.get());
}
```

This is pretty simple, and it's easy to see what's going on. When the method I want is called (it's mocked here), do this stuff. Amongst the stuff is a call to `incrementAndGet` for the `AtomicInteger`. You *could* use an `int[]` here, but that's not as clear in my opinion. We're just using something that's `final`, which we can increment. That's a limitation of the lambda we're using.

It's a bit crude, but it gets the job done in a simple and straightforward matter. At least if you know your lambdas and Mockito.

Share Improve this answer Follow

edited Dec 29, 2023 at 6:48



Nayan

1,595 ● 2 ● 14 ● 28

answered Mar 10, 2022 at 9:34



Haakon Løvteit

1,037 ● 12 ● 19

Add a comment

Proposta:

- Sugere o uso de um contador (`AtomicInteger`) para verificar quantas vezes um método é chamado, manipulando o retorno do método mockado.

Razão para não adoção:

- Embora a solução seja válida, ela pode ser vista como uma abordagem mais "crua" e menos clara. A abordagem é mais um hack para contornar o problema do que uma solução limpa e direta para garantir que o método foi chamado.

Resposta 7:



1

Another simple way would be add some log statement to the `bar.someMethod()` and then ascertain you can see the said message when your test executed, see examples here: [How to do a JUnit assert on a message in a logger](#)



That is especially handy when your `Bar.someMethod()` is `private`.



Share Improve this answer Follow

edited Sep 26, 2018 at 11:40

answered Sep 26, 2018 at 11:26



Nestor Milyaev

6,365 ● 4 ● 39 ● 54

Add a comment

Proposta:

- Sugere adicionar uma instrução de log ao método `someMethod` e verificar se a mensagem é exibida durante o teste.

Razão para não adoção:

- Adicionar logs para verificar a execução do método é uma abordagem indireta e pode adicionar complexidade desnecessária aos testes. Além disso, pode ser menos preciso e mais difícil de manter, especialmente em comparação com métodos de verificação direta como mocks.