



Universidade Federal de Sergipe  
Centro de Ciências Exatas e Tecnologia  
Departamento de Computação


# Atividade 1 – Testes Unitários e o Stack Overflow

Teste de Software - COMP0444 - T01  
Professor: Glauco de Figueiredo Carneiro

João Pablo da Paz de Jesus

# Tutorial: Seleção e Implementação de Soluções para Testes de Unidade

Link do Github: [https://github.com/Pablo-oficial/Teste\\_Software\\_2024\\_Jesus\\_Joao](https://github.com/Pablo-oficial/Teste_Software_2024_Jesus_Joao)

Link do Vídeo:  Joao\_Pablo\_atividade\_1.mkv

Este tutorial descreve as etapas percorridas para selecionar uma pergunta no Stack Overflow, reproduzir o problema em uma IDE, aplicar a solução proposta na resposta aceita e explicar por que as outras respostas não foram adotadas, conforme solicitado na atividade 1 da matéria de Teste de Software.

## Etapa 1: Escolha da Pergunta do StackOverflow

Para esta etapa, foi escolhida uma pergunta já cadastrada no Stack Overflow que discute obrigatoriamente um problema relacionado a testes de unidade ou teste de integração através da seguinte string de busca “[unit-testing] or [junit] or [pytest]”. A pergunta selecionada aborda a problemática **"how can I tell Moq to return null the first time and pageModel.Object the second?"**.

A dúvida apresentada pelo usuário se refere a como configurar o comportamento do Moq para que um método específico retorne valores diferentes em chamadas subsequentes.

Você pode acessar e visualizar a pergunta e as respostas dos usuário mais detalhadamente através do seguinte link: [Different return values the first and second time with Moq](#).

# Different return values the first and second time with Moq

Asked 12 years, 11 months ago   Modified 2 years, 1 month ago   Viewed 173k times

I have a test like this:

426

```
[TestCase("~/page/myaction")]
public void Page_With_Custom_Action(string path) {
    // Arrange
    var pathData = new Mock<IPathData>();
    var pageModel = new Mock<IPageModel>();
    var repository = new Mock<IPageRepository>();
    var mapper = new Mock<IControllerMapper>();
    var container = new Mock<IContainer>();

    container.Setup(x => x.GetInstance<IPageRepository>()).Returns(repository.Object);

    repository.Setup(x => x.GetPageByUrl<IPageModel>(path)).Returns(() => pageModel.Object);

    pathData.Setup(x => x.Action).Returns("myaction");
    pathData.Setup(x => x.Controller).Returns("page");

    var resolver = new DashboardPathResolver(pathData.Object, repository.Object, mapper.Object);

    // Act
    var data = resolver.ResolvePath(path);

    // Assert
    Assert.NotNull(data);
    Assert.AreEqual("myaction", data.Action);
    Assert.AreEqual("page", data.Controller);
}
```

`GetPageByUrl` runs twice in my `DashboardPathResolver`, how can I tell Moq to return `null` the first time and `pageModel.Object` the second?

c#   unit-testing   nunit   moq

## Etapa 2: Reprodução do Teste em uma IDE

A resposta selecionada no Stack Overflow para a pergunta sobre como fazer o Moq retornar valores diferentes em chamadas subsequentes utilizou uma abordagem específica para resolver o problema. A resposta fez uso do método **ReturnsInOrder** do Moq, uma técnica mencionada no post de Haacked, que permite definir uma sequência de valores a serem retornados pelo mock.



32



Adding a callback did not work for me, I used this approach instead  
<http://haacked.com/archive/2009/09/29/moq-sequences.aspx> and I ended up with a test like this:

```
[TestCase("~/page/myaction")]
[TestCase("~/page/myaction/")]
public void Page_With_Custom_Action(string virtualUrl) {

    // Arrange
    var pathData = new Mock<IPathData>();
    var pageModel = new Mock<IPageModel>();
    var repository = new Mock<IPageRepository>();
    var mapper = new Mock<IControllerMapper>();
    var container = new Mock<IContainer>();

    container.Setup(x => x.GetInstance<IPageRepository>()).Returns(repository.Object);
    repository.Setup(x => x.GetPageByUrl<IPageModel>(virtualUrl)).ReturnsInOrder(
        pageModel.Object,
        pageModel.Object);

    pathData.Setup(x => x.Action).Returns("myaction");
    pathData.Setup(x => x.Controller).Returns("page");

    var resolver = new DashboardPathResolver(pathData.Object, repository.Object);

    // Act
    var data = resolver.ResolvePath(virtualUrl);

    // Assert
    Assert.NotNull(data);
    Assert.AreEqual("myaction", data.Action);
    Assert.AreEqual("page", data.Controller);
}
```

## 1. Criação do Projeto

Primeiro, criei um projeto de teste unitário usando o .NET CLI. O projeto é configurado para usar o NUnit e o Moq, que são bibliotecas populares para testes e mocks em C#.

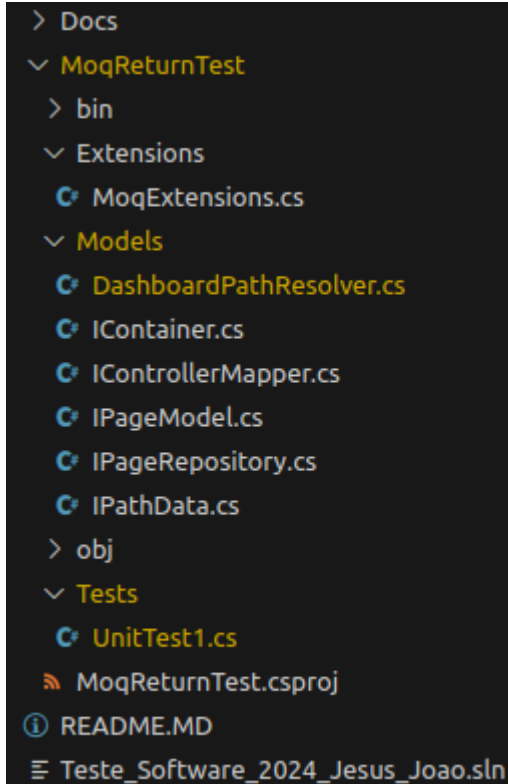
```
pablo@pablo-Nitro-AN515-45:~/Faculdade/Teste de Software/Teste_Software_2024_Jesus_Joao$ dotnet new nunit -n MoqReturnTest
cd MoqReturnTest
dotnet add package Moq

```

Este comando cria um novo projeto NUnit e adiciona a biblioteca Moq como uma dependência.

## 2. Estrutura do Projeto

Dentro do projeto, a estrutura é a seguinte:



```
> Docs
▼ MoqReturnTest
  > bin
  ▼ Extensions
    • MoqExtensions.cs
  ▼ Models
    • DashboardPathResolver.cs
    • IContainer.cs
    • IControllerMapper.cs
    • IPageModel.cs
    • IPageRepository.cs
    • IPathData.cs
  > obj
  ▼ Tests
    • UnitTest1.cs
  • MoqReturnTest.csproj
  ⓘ README.MD
  ≡ Teste_Software_2024_Jesus_Joao.sln
```

- **MoqReturnTest:**
  - **Models/** (Contém interfaces e modelos de dados)
  - **Extensions/** (Contém quaisquer extensões ou métodos auxiliares)
  - **Tests/** (Contém os arquivos de teste unitário)

## 3. Criação e Instanciação dos Mocks

Para testar o comportamento esperado, criei mocks para as interfaces e classes envolvidas. As interfaces são mockadas para que possamos definir comportamentos esperados para métodos específicos sem depender de implementações reais.

Também criei instâncias mock das interfaces e configurei o comportamento esperado usando o Moq como foi apresentado na pergunta.

```
// Arrange
var pathData = new Mock<IPathData>();
var pageModel = new Mock<IPageModel>();
var repository = new Mock<IPageRepository>();
var mapper = new Mock<IControllerMapper>();
var container = new Mock<IContainer>();
```

### 3. Criação do Método ReturnsInOrder no MoqExtensions

O **MoqExtensions** é uma classe de extensão que adiciona um método personalizado para os setups de mocks com a biblioteca Moq. No caso específico, estamos adicionando um método chamado **ReturnsInOrder** que permite definir uma sequência de valores a serem retornados em chamadas sucessivas a um método mockado.

```
using System.Collections.Generic;
using Moq.Language.Flow;

namespace MoqReturnTest.Extensions
{
    0 referências
    public static class MoqExtensions
    {
        0 referências
        public static void ReturnsInOrder<T, TResult>(this ISetup<T, TResult> setup,
            params TResult[] results) where T : class {
            setup.Returns(new Queue<TResult>(results).Dequeue);
        }
    }
}
```

Quando você precisa que um método mockado retorne uma sequência específica de valores, como retornar **null** na primeira chamada e um objeto na segunda chamada, você pode usar o método **ReturnsInOrder** para simplificar essa configuração. Esse padrão é útil quando você precisa testar comportamentos que dependem de respostas diferentes em chamadas subsequentes.

### 4. Testes Unitários no UnitTest1.cs

O arquivo **UnitTest1.cs** é responsável por realizar testes unitários para verificar o comportamento da classe **DashboardPathResolver** em um cenário específico. Este arquivo de teste foi criado com base na pergunta do usuário no Stack Overflow e segue a estrutura e lógica proposta na resposta aceita.

```
0 referências
public class Tests
{
    [TestCase("~/page/myaction")]
    [TestCase("~/page/myaction/")]
    0 referências
    public void Page_With_Custom_Action(string virtualUrl)
    {
        // Arrange
        var pathData = new Mock<IPathData>();
        var pageModel = new Mock<IPageModel>();
        var repository = new Mock<IPageRepository>();
        var mapper = new Mock<IControllerMapper>();
        var container = new Mock<IContainer>();

        container.Setup(x => x.GetInstance<IPageRepository>()).Returns(repository.Object);
        repository.Setup(x => x.GetPageByUrl<IPageModel>(virtualUrl)).ReturnsInOrder([null, pageModel.Object]);

        pathData.Setup(x => x.Action).Returns("myaction");
        pathData.Setup(x => x.Controller).Returns("page");

        var resolver = new DashboardPathResolver(pathData.Object, repository.Object, mapper.Object, container.Object);

        // Act
        var data = resolver.ResolvePath(virtualUrl);

        // Assert
        Assert.NotNull(data);
        Assert.AreEqual("myaction", data.Action);
        Assert.AreEqual("page", data.Controller);
    }
}
```

### Mock das Dependências:

**pathData**, **pageModel**, **repository**, **mapper**, e **container** são objetos mock criados usando a biblioteca Moq. Esses mocks simulam o comportamento das interfaces sem precisar de implementações reais.

**container.Setup(x => x.GetInstance<IPageRepository>()).Returns(repository.Object)** configura o mock container para retornar repository quando **GetInstance<IPageRepository>** for chamado.

### Retornos Sequenciais:

**repository.Setup(x => x.GetPageByUrl<IPageModel>(virtualUrl)).ReturnsInOrder(null, pageModel.Object)** utiliza a extensão **ReturnsInOrder** para configurar o mock **repository** para retornar **null** na primeira chamada e **pageModel.Object** na segunda. Esta configuração é diretamente inspirada pela resposta aceita na pergunta do Stack Overflow.

### Instância do Resolver:

**var resolver = new DashboardPathResolver(pathData.Object, repository.Object, mapper.Object, container.Object)** cria uma instância da classe **DashboardPathResolver**, passando os mocks configurados.

Este objeto é o alvo do teste, e o método **ResolvePath** é chamado com o **virtualUrl** especificado nos casos de teste.

**Validação:**

**Assert.NotNull(data)** verifica se o objeto retornado por **ResolvePath** não é nulo.

**Assert.AreEqual("myaction", data.Action)** e **Assert.AreEqual("page", data.Controller)** verificam se as propriedades Action e Controller do objeto retornado são as esperadas.

## 5. Validação do Teste

Após a implementação do teste unitário no arquivo **UnitTest1.cs**, é essencial validar o funcionamento correto através da execução dos testes. Isso pode ser feito usando o comando **dotnet test**, que faz parte do .NET CLI.

### Passos para Executar o Teste

**Navegar até o Diretório do Projeto:** Certifique-se de estar no diretório raiz do projeto onde o arquivo de teste **UnitTest1.cs** está localizado.

```
/Teste_Software_2024_Jesus_Joao$ cd MoqReturnTest
```

**Executar o Comando de Teste:** Use o comando **dotnet test** para compilar e executar todos os testes no projeto.

```
/Teste_Software_2024_Jesus_Joao/MoqReturnTest$ dotnet test
```

**Analisar a Saída dos Testes:** A saída do comando **dotnet test** fornecerá informações detalhadas sobre os testes executados, incluindo quais testes passaram, falharam ou foram ignorados. Um exemplo de saída bem-sucedida pode se parecer com o seguinte:

```
Determining projects to restore...
Todos os projetos estão atualizados para restauração.
MoqReturnTest -> /home/pablo/Faculdade/Teste de Software/Teste Software 2024 Jesus Joao/MoqReturnTest/bin/Debug/net8.0/MoqReturnTest.dll
Execução de teste para /home/pablo/Faculdade/Teste de Software/Teste Software 2024 Jesus Joao/MoqReturnTest/bin/Debug/net8.0/MoqReturnTest.dll
Ferramenta de Linha de Comando de Execução de Teste da Microsoft (R) Versão 17.10.0 (x64)
Copyright (c) Microsoft Corporation. Todos os direitos reservados.

Iniciando execução de teste, espere...
1 arquivos de teste no total corresponderam ao padrão especificado.

Aprovado! - Com falha: 0, Aprovado: 2, Ignorado: 0, Total: 2, Duração: 62 ms - MoqReturnTest.dll (net8.0)
```



## Objetivo: Motivo da Escolha da Resposta

**Implementação Prática:** A resposta incluiu uma implementação prática que resolve o problema diretamente, sem a necessidade de callbacks complexos ou outras abordagens mais difíceis de entender.

**Utilidade Geral:** A abordagem **ReturnsInOrder** pode ser reutilizada em outros contextos, o que a torna uma solução versátil para problemas semelhantes de mock.

**Link para Recurso Externo:** A resposta fornece uma referência para uma fonte confiável (<http://haacked.com/archive/2009/09/29/moq-sequences.aspx>) que explica a lógica por trás da solução, permitindo que os usuários interessados aprofundem seu entendimento.

## Objetivo: Motivo das Outras Não Serem Aceitas



With the latest version of Moq(4.2.1312.1622), you can setup a sequence of events using **SetupSequence**. Here's an example:

667



```
_mockClient.SetupSequence(m => m.Connect(It.IsAny<String>(), It.IsAny<int>(), It.IsAny<int>()))
    .Throws(new SocketException())
    .Throws(new SocketException())
    .Returns(true)
    .Throws(new SocketException())
    .Returns(true);
```

Calling connect will only be successful on the third and fifth attempt otherwise an exception will be thrown.

So for your example it would just be something like:

```
repository.SetupSequence(x => x.GetPageByUrl<IPageModel>(virtualUrl))
    .Returns(null)
    .Returns(pageModel.Object);
```

**Compatibilidade de Versão:** **SetupSequence** requer uma versão específica do Moq (4.2.1312.1622 ou posterior), o que pode não ser viável para todos os usuários.



127

The existing answers are great, but I thought I'd throw in my alternative which just uses `System.Collections.Generic.Queue` and doesn't require any special knowledge of the mocking framework - since I didn't have any when I wrote it! :)



```
var pageModel = new Mock<IPageModel>();
IPageModel pageModelNull = null;
var pageModels = new Queue<IPageModel>();
pageModels.Enqueue(pageModelNull);
pageModels.Enqueue(pageModel.Object);
```

Then...

```
repository.Setup(x => x.GetPageByUrl<IPageModel>(path)).Returns(pageModels.Dequeue);
```

**Uso de Estruturas Adicionais:** Utilizar **Queue** adiciona uma camada extra de complexidade ao código, o que pode não ser ideal para todos os cenários.



102

Now you can use `SetupSequence`. See [this post](#).



```
var mock = new Mock<IFoo>();
mock.SetupSequence(f => f.GetCount())
    .Returns(3) // will be returned on 1st invocation
    .Returns(2) // will be returned on 2nd invocation
    .Returns(1) // will be returned on 3rd invocation
    .Returns(0) // will be returned on 4th invocation
    .Throws(new InvalidOperationException()); // will be thrown on 5th invocation
```

**Excesso de Complexidade para o Problema Específico:** Embora `SetupSequence` seja uma solução poderosa, ela pode ser considerada excessiva para um problema simples como o retorno de `null` na primeira chamada e um objeto na segunda



You can use a callback when setting up your mock object. Take a look at the example from the Moq Wiki (<https://github.com/Moq/moq4/wiki/Quickstart>).

39



```
// returning different values on each invocation
var mock = new Mock<IFoo>();
var calls = 0;
mock.Setup(foo => foo.GetCountThing())
    .Returns(() => calls)
    .Callback(() => calls++);
// returns 0 on first invocation, 1 on the next, and so on
Console.WriteLine(mock.Object.GetCountThing());
```

Your setup might look like this:

```
var pageObject = pageModel.Object;
repository.Setup(x => x.GetPageByUrl<IPageModel>(path)).Returns(() => pageObject).Callback(() => {
    // assign new value for second call
    pageObject = new PageModel();
});
```

**Inconsistência nos Resultados:** A abordagem usando **Callback** demonstrou problemas práticos, onde a solução não funcionou conforme esperado. Usuários relataram que a função ainda retornava **null** em ambas as chamadas, mesmo após a implementação da callback.



The [accepted answer](#), as well as the [SetupSequence answer](#), handles returning constants.

6



`Returns()` has some useful overloads where you can return a value based on the parameters that were sent to the mocked method. Based on [the solution](#) given in the accepted answer, here is another extension method for those overloads.



```
public static class MoqExtensions
{
    public static IReturnsResult<TMock> ReturnsInOrder<TMock, TResult, T1>(this ISetup<TMock> setup, TResult value)
        where TMock : class
    {
        var queue = new Queue<Func<T1, TResult>>(valueFunctions);
        return setup.Returns<T1>(arg => queue.Dequeue()(arg));
    }
}
```

Unfortunately, using the method requires you to specify some template parameters, but the result is still quite readable.

```
repository
    .Setup(x => x.GetPageByUrl<IPageModel>(path))
    .ReturnsInOrder(new Func<string, IPageModel>[]
    {
        p => null, // Here, the return value can depend on the path parameter
        p => pageModel.Object,
    });
```

Create overloads for the extension method with multiple parameters (`T2`, `T3`, etc) if needed.

**Complemento:** A resposta em questão funciona como um complemento da resposta que foi aceita como correta pelo StackOverflow



Reached here for the same kind of problem with slightly different requirement.  
I need to get **different return values from mock based in different input values** and found solution which IMO more readable as it uses Moq's declarative syntax (linq to Mocks).

4



```
public interface IDataAccess
{
    DbValue GetFromDb(int accountId);
}

var dataAccessMock = Mock.Of<IDataAccess>
(da => da.GetFromDb(It.Is<int>(acctId => acctId == 0)) == new Account { AccountStat
&& da.GetFromDb(It.Is<int>(acctId => acctId == 1)) == new DbValue { AccountStatus :
&& da.GetFromDb(It.Is<int>(acctId => acctId == 2)) == new DbValue { AccountStatus :

var result1 = dataAccessMock.GetFromDb(0); // returns DbValue of "None" AccountStat
var result2 = dataAccessMock.GetFromDb(1); // returns DbValue of "InActive" Accou
var result3 = dataAccessMock.GetFromDb(2); // returns DbValue of "Deleted" Account
```

**Limitações da Abordagem:** O uso de **Mock.Of** com sintaxe declarativa é limitado a casos simples e não permite uma configuração dinâmica de comportamentos complexos



We can simply declare one variable with **int** as a datatype. initialize it to **zero**, and then increase it's value as follows:

2



```
int firstTime = 0;
repository.Setup(_ => _.GetPageByUrl<IPageModel>(path)).Returns(() =>
{
    if (firstTime == 0)
    {
        firstTime = 1;
        return null;
    }
    else if (firstTime == 1)
    {
        firstTime = 2;
        return pageModel.Object;
    }
    else
    {
        return null;
    }
});
```

**Sem Configuração Dinâmica:** A abordagem não permite a configuração dinâmica de comportamentos que podem mudar entre chamadas

# Referências

**Stack Overflow. Different return values the first and second time with Moq.**

Disponível em:

<https://stackoverflow.com/questions/7287540/different-return-values-the-first-and-second-time-with-moq/7300657#7300657>. Acesso em: 4 ago. 2024.

**Haacked. Moq Sequences.** Disponível em:

<https://haacked.com/archive/2009/09/29/moq-sequences.aspx/>. Acesso em: 5 ago. 2024.