METHODPOET

Unit Testing in C# With Moq – Wake up the Testing Genius Inside You

Unit testing is a powerful way to ensure that your code works as intended. It's a great way to combat the common "works on my machine" problem.

Using Moq, you can mock out dependencies and make sure that you are testing the code in isolation.

Moq is a mock object framework for .NET that greatly simplifies the creation of mock objects for unit testing. Mocking is a popular technique for unit testing that creates test double objects, which gives you the ability to control the behavior of those objects by setting their outcomes.

This article will show you how to unit test your C# code with Moq.

Table of Contents

What is a test double, mock, and stub?

Test doubles are objects that mimic the behavior of real objects in controlled ways. The goal of test double objects is to allow for the concise development of tests that affect only one object in isolation. Mocking frameworks allow for the separation of a system under test from its dependencies to control the inputs and outputs of the system under test.

In a nutshell, a test double is a fake object used to ensure the behavior of some external system is the same as the original object.

For example, suppose you're building a web application, which gets data from a database. If you want to write tests for that application, you would need to provide a clean, pre-populated database with real data for every test. This is where test double comes into play. It can substitute the concrete class that performs database fetch and return fake data. But the code you are testing won't know the difference. This assumes you are using dependency injection in your code. So you can easily replace one implementation of an interface with another in tests.

Faking objects is a modern software engineering practice. By mocking objects, you ensure your unit test only covers the code you actually want to test. This also means your test code can be fast since your testable code is isolated from its environment.

Even though all these fake objects are often called mocks, when I'm talking about mocks, I distinguish them from stubs.

What is a stub?

A stub object is a fake object used to test some unit of code without using a real object. The idea is that you can simulate an object's behavior by setting up some test code that will throw errors, return incorrect data, or otherwise do what an object would do in certain situations. This allows you to run tests without connecting to a database or make any other calls to the outside world. This is important in tests because you don't want to wait for a web call to run the test.

What is a mock, then?

A mock object goes a bit further. When you use it in your unit test case, it checks that the system under test interacts with other objects as expected. It can check that a dependency was called with specific arguments or that a certain call didn't happen.

The outcome of the unit test is then checked against the mock object.

You can learn a lot more about mock vs. stub difference and find out when to use each of them in my separate blog post.

Moq framework

The Moq framework is a set of interfaces that allow you to stub or mock your code for unit testing purposes. Clarius, Manas, and InSTEDD developed it to facilitate the writing of unit tests. Due to the framework's ability to create test doubles, it is often used to develop automated tests.

The biggest benefit of using this mocking framework is that it can create those test doubles, dependencies on the fly. This greatly improves the speed of writing unit tests (since many developers don't have time for testing). You can use it for testing code that interacts with web services, databases, or any other class that is likely to be used in a unit test.

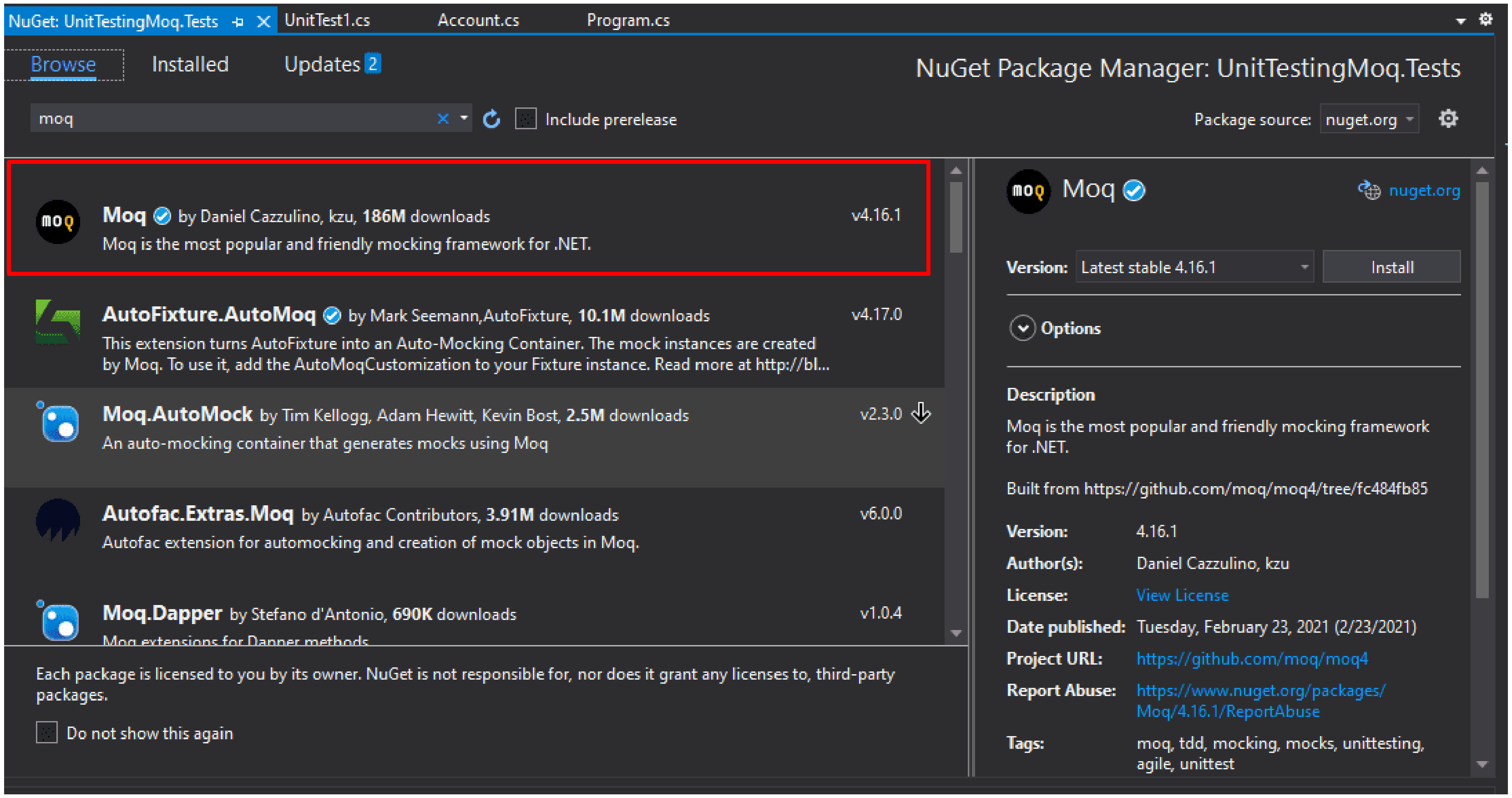
The result is that you can verify that your code behaves the way it is supposed to and quickly discover when a bug occurs.

How to install it?

To install the Moq framework to your test project, you can easily add it as a NuGet package:

```
Install-Package Moq -Version 4.16.1
```

The other option is to select your unit test project and manage NuGet packages through UI.


NuGet: UnitTestingMoq.Tests
UnitTest1.cs
Account.cs
Program.cs
Browse
Installed
Updates 2
NuGet Package Manager: UnitTestingMoq.Tests
moq
Include prerelease
Package source: nuget.org

Moq by Daniel Cazzulino, kzu, **186M** downloads v4.16.1
Moq is the most popular and friendly mocking framework for .NET.

AutoFixture.AutoMoq by Mark Seemann,AutoFixture, **10.1M** downloads v4.17.0
This extension turns AutoFixture into an Auto-Mocking Container. The mock instances are created by Moq. To use it, add the AutoMoqCustomization to your Fixture instance. Read more at http://bl...

Moq.AutoMock by Tim Kellogg, Adam Hewitt, Kevin Bost, **2.5M** downloads v2.3.0
An auto-mocking container that generates mocks using Moq

Autofac.Extras.Moq by Autofac Contributors, **3.91M** downloads v6.0.0
Autofac extension for automocking and creation of mock objects in Moq.

Moq.Dapper by Stefano d'Antonio, **690K** downloads v1.0.4
Moq extensions for Dapper methods

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Moq nuget.org

Version: Latest stable 4.16.1 Install

Options

Description
Moq is the most popular and friendly mocking framework for .NET.

Built from https://github.com/moq/moq4/tree/fc484fb85

Version:	4.16.1
Author(s):	Daniel Cazzulino, kzu
License:	View License
Date published:	Tuesday, February 23, 2021 (2/23/2021)
Project URL:	https://github.com/moq/moq4
Report Abuse:	https://www.nuget.org/packages/Moq/4.16.1/ReportAbuse
Tags:	moq, tdd, mocking, mocks, unittesting, agile, unittest

Install Moq through NuGet

Benefits of using Moq

There are many benefits of using the Moq testing framework:

- It is easy to set up a system under test – Moq makes testing easier because it generates appropriate mocks/stubs for you. This means that you can focus your time and energy on testing the logic of your code rather than on designing tests.
- It is easier to unit test more complicated objects – Moq makes it easier to unit test more complicated objects without writing a lot of boilerplate code. It also makes it easier to assert that your tests are passing.
- It is easy to use and understand – Moq has a clean and intuitive API interface, making it easy to use and understand for both new and experienced developers.
- It is easy to find examples on the web – Moq is a trendy open-source mocking framework. There are plenty of examples on the web, so you don't have to worry!
- You can use it early in the development – if you practice test-driven development, you can create an interface that doesn't yet have the implementation. After that, you can use Moq in your tests to simulate how the implemented interface will behave in the future. By doing this, you can refine methods of the interface to better suit your needs.

Simple Unit Test Example With Moq

Let's see some examples of how to use Moq while writing unit tests in C#.

The class under test

We have the following `AccountService` class that has dependencies on `IBookService` and `IEmailSender`.

```
public interface IBookService
{
    string GetISBNFor(string bookTitle);
    IEnumerable<string> GetBooksForCategory(string categoryId);
}
public interface IEmailSender
{
    public void SendEmail(string to, string subject, string body);
}
public class AccountService
{
    private IBookService _bookService;
    private IEmailSender _emailSender;
    public AccountService(IBookService bookService, IEmailSender emailSender)
    {
        _bookService = bookService;
        _emailSender = emailSender;
    }
    public IEnumerable<string> GetAllBooksForCategory(string categoryId)
    {
        var allBooks = _bookService.GetBooksForCategory(categoryId);
        return allBooks;
    }
    public string GetBookISBN(string categoryId, string searchTerm)
    {
        var allBooks = _bookService.GetBooksForCategory(categoryId);
        var foundBook = allBooks
                        .Where(x => x.ToUpper().Contains(searchTerm.ToUpper()))
                        .FirstOrDefault();
        if (foundBook == null)
        {
            return string.Empty;
        }
        return _bookService.GetISBNFor(foundBook);
    }
    public void SendEmail(string emailAddress, string bookTitle)
    {
        string subject = "Awesome Book";
        string body = $"Hi,\n\nThis book is awesome: {bookTitle}.\nCheck it out.";
        _emailSender.SendEmail(emailAddress, subject, body);
    }
}
```

We want to test these three methods, `GetAllBooksForCategory`**,** `GetBookISBN`**,** and `SendEmail`**.** The methods use the `IBookService` and `IEmailSender` dependency.

You will see how to use Moq to have a fake instance of `IBookService`.

What about `IEmailSender`? Should you also create a fake instance of it in your tests? Well, you don't need to if you test `GetAllBooksForCategory` and `GetBookISBN` methods. They don't use the email sending module, and that's why feel free to pass ***`null`*** in those tests to the `AccountService` constructor.

> *Always write the least amount of code in your test method you need for the test to pass.*
>
> **– Kristijan Kralj**

You can use Moq to provide fake instances of the dependencies, but if the test will pass with null, use the null value instead. This will decrease the logic in your unit test and make it simpler to understand.

Later, when we test the `SendEmail` method of the `AccountService`, we will use Moq to create a mock instance of the `IEmailSender`.

The first test using Moq

I will create a new test class `AccountServiceTests`, to write tests for these methods. If you want to learn how to name your methods, classes, and unit test project properly, I suggest you read the post about unit testing naming conventions.

The first method, `GetAllBooksForCategory`, is simple. It queries the `IBookService` and returns the result. To write a unit test that will check that the method works as expected, we need to write a stub for the `IBookService`.

```
public void GetAllBooksForCategory_returns_list_of_available_books()
{
    //1
    var bookServiceStub = new Mock<IBookService>();
    //2
    bookServiceStub
        .Setup(x => x.GetBooksForCategory("UnitTesting"))
        .Returns(new List<string>
        {
            "The Art of Unit Testing",
            "Test-Driven Development",
            "Working Effectively with Legacy Code"
        });
    //3
    var accountService = new AccountService(bookServiceStub.Object, null);
    IEnumerable<string> result = accountService.GetAllBooksForCategory("UnitTesting");
    Assert.Equal(3, result.Count());
}
```

What does the code mean in the `GetAllBooksForCategory_returns_list_of_available_books` unit test:

1. Start by creating a new fake object using a generic Mock class.
2. We need to set up this mock instance to return a list of books when the `GetBooksForCategory` method is called with the "UnitTesting" parameter.
3. Pass the fake instance to the AccountService's constructor by calling the `Object` property.

You can see in the test runner output window that the test passes if you run the test. This means that the fake service has returned 3 books when called.

How to set up multiple methods on a stub

The next method of `AccountService` we want to test is the `GetBookISBN`. It uses two methods of the `IBookService`, `GetBooksForCategory` and `GetISBNFor`.

```
[Fact]
public void GetBookISBN_founds_the_correct_book_for_search_term()
{
    var bookServiceStub = new Mock<IBookService>();
    //1
    bookServiceStub
        .Setup(x => x.GetBooksForCategory("UnitTesting"))
        .Returns(new List<string>
        {
            "The Art of Unit Testing",
            "Test-Driven Development",
            "Working Effectively with Legacy Code"
        });
    //2
    bookServiceStub
        .Setup(x => x.GetISBNFor("The Art of Unit Testing"))
```

```
            .Returns("0-9020-7656-6");
        var accountService = new AccountService(bookServiceStub.Object, null);
        string result = accountService.GetBookISBN("UnitTesting", "art");
        Assert.Equal("0-9020-7656-6", result);
    }
```

1. As in the last test, start by adding a stub for the `GetBooksForCategory` method.
2. After that, set up the second method, `GetISBNFor`.

If you run the test, it should pass.

Common usage examples

Before we move to implement mocks with the Moq, let me show you some common usage examples.

Ignore input parameter value

In the last example, I have used the parameter matching to set up the fake object. This expects that the method is called with the “UnitTesting” paremeter.

```
    bookServiceStub
        .Setup(x => x.GetBooksForCategory("UnitTesting"))
        .Returns(new List<string>
        {
            "The Art of Unit Testing",
            "Test-Driven Development",
            "Working Effectively with Legacy Code"
        });
```

However, you can simplify this. If you don't care about the input parameter of the mocked method, you can use It.IsAny.

Example:

```
bookServiceStub
    .Setup(x => x.GetBooksForCategory(It.IsAny<string>()))
    .Callback<string>(s => passedParameter = s)
    .Returns(new List<string>
    {
        "The Art of Unit Testing",
        "Test-Driven Development",
        "Working Effectively with Legacy Code"
    });
```

Throw exception

Sometimes, you want to check your code's robustness and resistance to failure by throwing an exception. To throw an exception, use the `Throws` mock function:

```
bookServiceStub
    .Setup(x => x.GetBooksForCategory(It.IsAny<string>()))
    .Throws<InvalidOperationException>();
```

Setup Properties

Often, you need to set up properties of an interface to return a specific value:

```
emailSenderMock.Setup(x => x.EmailServer).Returns("Gmail");
```

If you want to stub all properties, you can use the `StubAllProperties` setup method:

```
bookServiceStub.SetupAllProperties();
```

Events

Moq support various event's setup options:

```
// Attach and detach from event (Moq 4.13 or later):
mock.SetupAdd(m => m.CustomEvent += It.IsAny<EventHandler>())...;
mock.SetupRemove(m => m.CustomEvent -= It.IsAny<EventHandler>())...;
// How to raise an event on the mock
mock.Raise(m => m.CustomEvent += null, new CustomEventArgs(someValue));
// How to raise an event on the mock that has sender in handler parameters
mock.Raise(m => m.CustomEvent += null, this, new CustomEventArgs(someValue));
```

Callbacks

The `Callback` method is a nice way to capture the parameter that was passed to a method. Let's take a look at the following test.

```
[Fact]
public void GetAllBooksForCategory_returns_list_of_available_books_with_callback()
{
    var bookServiceStub = new Mock<IBookService>();
    string passedParameter = string.Empty;
    bookServiceStub
    .Setup(x => x.GetBooksForCategory(It.IsAny<string>()))
    .Callback<string>(s => passedParameter = s)
    .Returns(new List<string>
    {
    "The Art of Unit Testing",
    "Test-Driven Development",
    "Working Effectively with Legacy Code"
    });
    var accountService = new AccountService(bookServiceStub.Object, null);
    IEnumerable<string> result = accountService.GetAllBooksForCategory("UnitTesting");
    Assert.Equal("UnitTesting", passedParameter);
}
```

In this example, the ***passedParameter*** variable will get the value “UnitTesting”. That’s the value that was passed to the `GetBooksForCategory` method.

Sequential calls

There are also times when you will call a single method multiple times during a single test. In that case, you can choose what value will be returned on every call.

```
bookServiceStub
    .SetupSequence(x => x.GetISBNFor(It.IsAny<string>()))
    .Returns("0-9020-7656-6") //returned on 1st call
    .Returns("0-9180-6396-5") //returned on 2nd call
    .Returns("0-3860-1173-7") //returned on 3rd call
    .Returns("0-5570-1450-6");//returned on 4th call
```

How to create mocks and verify expectations

Creating mocks and verifying expectations in unit tests using can be a tricky and frustrating process. Especially if you don't use a mocking framework. Then you need to implement your mocks manually and track which method was called and when.

If you use a Moq and know how to set up mocks, this doesn't have to be difficult. You just need to follow a few simple steps, and you can create effective mocks and verifying code that works for you.

The easiest way to understand how mocks works is by writing a test.

We want to test that the `SendEmail` method calls `IEmailSender` with correct arguments. We can't test the actual email sending, since that wouldn't be a unit test anymore.

```
[Fact]
public void SendEmail_sends_email_with_correct_content()
{
    //1
    var emailSenderMock = new Mock<IEmailSender>();
    //2
```

```
        var accountService = new AccountService(null, emailSenderMock.Object);
        //3
        accountService.SendEmail("test@gmail.com", "Test - Driven Development");
        //4
        emailSenderMock.Verify(x => x.SendEmail("test@gmail.com", "Awesome Book", $"Hi,\n\nThis book is awesome: Te
    }
```

The test does the following:

1. It creates the `IEmailSender` mock.
2. It passes that mock to the constructor.
3. It calls the method we want to check, `SendEmail`, with parameters.
4. The `Verify` method is used to check that what were the passed arguments to the `SendEmail` method. It also asserts that the method was called only once using the `Times.Once` parameter.

Testing Asynchronous Code with Moq

Modern applications are typically made up of large microservices with many different moving parts. When you're writing these sorts of applications, the applications need to talk asynchronously. But it also important to test your code.

Unit testing asynchronous code is the black sheep of the software testing world. For some reason, it's often thought to be so difficult that people avoid it altogether.

This can be a huge mistake because testing asynchronous code is not only possible but also relatively straightforward.

So, how can you tell Moq to return a `Task` or `Task<T>`?

Well, you can use `ReturnsAsync`:

```
httpClientMock
.Setup(x => x.GetAsync(It.IsAny<string>()))
.ReturnsAsync(true);
```

Starting with Moq 4.16, you can also use the `Returns` method by combining it with the `Result` property of your task:

```
httpClientMock
.Setup(x => x.GetAsync(It.IsAny<string>()).Result)
.Returns(true);
```

Testing Exception was thrown

Modern applications use exceptions to signalize that the error has occurred or something unexpected has happened. When you get an exception in your code, there is a whole chain of events that occur. The exception usually propagates through the call stack until it is handled properly.

While writing unit tests, you want to cover cases where an exception will happen if the input is invalid.

Luckily, Moq also covers this.

```
Action act = () => sut.PerformValidation();
act.ShouldThrow<NullReferenceException>();
```

Multiple mocks in a single test

Can you have more than one mock instance in a single test method? The answer is an absolute yes.

Should you have more than one mock instance? The answer is absolutely no.

You can have several stubs and one mock, but avoid having more then one mock in a single test case.

What’s the problem if you have more than one mock in a single unit test?

If you have two or more mocks that you are asserting against in your test, this means you are probably checking more than one scenario. Remember, a single unit test should only cover one scenario of your system under test. By checking more scenarios, you are making the test more unstable and less maintainable. Once the test fails, you need to go and check every object you have mocked to see what’s the issue. And this will take more time if you have multiple mocks.

The best thing you can do for your unit test is to make it as simple as possible.

FAQ

What’s the big deal about testing in isolation?

You should write unit tests to test the smallest possible piece of functionality in an application. By isolating the unit tests from any external influence, you can be sure that the test is testing the functionality and not some other part of the application that may change in the future.

Can you mock a class?

While mocking a class is not recommended, it is possible to do so. To make a class mockable, it needs to be a public class, and the method you want to mock has to be virtual.

Can you use Moq with MSTest?

Yes, you can use Moq with MSTest. Moq should support every popular unit testing framework, including xUnit, NUnit, and MSTest.

Why is mocking necessary in unit testing?

Mocking is necessary in unit testing because you have to isolate the code that you are testing from the surrounding code. This can be done by replacing the actual dependencies with a mock or stub.

What is a mock repository? How do you create a mock repository?

A mock repository is a way to centralize the management of your mocks. You can do that using MockRepository.

Kristijan Kralj

Recent Posts

4 Amazing Benefits of Integration Testing (+ 4 Drawbacks, Sadly)

So you're working on a new software project and about to reach the testing phase. That's great news! But have you considered the different testing types you need to perform? One of them...

CONTINUE READING

Integration Testing 101: A Beginner's Guide

With the growing demand for software applications, software development has become a complex process. It involves numerous stages, from planning and coding to testing and deploying. Every stage...

CONTINUE READING

ABOUT ME

Hello! My name is Kristijan Kralj, and I am a C# software developer with 10 years of experience.

I have worked on various software projects ranging from simple programs to large enterprise systems. As a developer, I have acquired a wealth of experience and knowledge in C#, software architecture, unit testing, DevOps, and Azure. I enjoy working on complex systems that require creative solutions.

When I'm not glued to my computer screen, I like to spend time with my wife and two kids.

5 Secret Steps To Improve Your Code Quality

Do you want to build better, cleaner, and more manageable C# code?

Improve your coding habits with these not-so-obvious 5 tips and tricks.

This *FREE* email course (value $37) shows you how to **automatically** identify and remove the most common code smells.

You don't need any third-party tool or plugin, only Visual Studio.

Join 200+ developers who already use these amazing steps in their daily job.

Your Email

I WANT TO SIMPLIFY MY CODE!

We respect your privacy. Unsubscribe at any time.