

Lab Module 1

Introduction

In automated testing it is common to use objects that look and behave like their production equivalents but are actually simplified. This reduces complexity, allows to verify code independently from the rest of the system and sometimes it is even necessary to execute self-validating tests at all. A Test Double is a generic term used for these objects.

We are going to explore the 3 different types of unit tests (value, stated & interaction based) as well as what are the different types of test doubles before installing a Mocking framework which may lead developer to misunderstandings, influence test design and increase test fragility and refactoring.

Pre-requisites

Visual Studio 2015, 2017, 2019 and GitHub account.

Objectives

After completing this lab, you will be able to:

- Add a unit test project to your solution and snippets
- Design your own Fakes, Stubs and Mocks
- Understand the Strategy Pattern.

Scenario

We are writing a program for a smart home microcontroller, and one of the requirements is to automatically turn on the all the devices that are online.

Another requirement is being able to perform firmware updates on offline devices

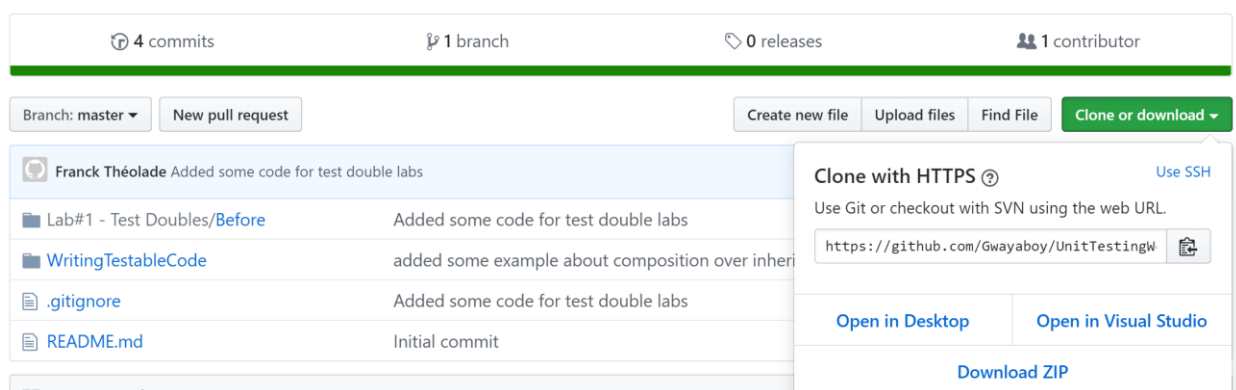
Lastly, we'll look into implementing a scheduled firmware update on online devices as well.

Setting up

Download and clone the repository

```
git clone https://github.com/Gwayaboy/UnitTestingWorkshop
```

Or clone navigate to the URL above and choose clone with Visual Studio:



Exercise 1: Write state-based and value-based unit tests with stub

Introduction

- 1) This code Snippet will ensure consistent test naming convention
- 2) Add value-based unit

Tasks 1.1: Install unit test code snippet

1. Download and save the snippet below

<https://gist.githubusercontent.com/osmyn/906c917653a30864cb52dee02c36c14e/raw/68ec9bd89ea2686ca2adeb540ef2759a07e7bbfd/unittest.snippet>

into your visual studio's code snippet directory.

For example, if you are using VS2019:

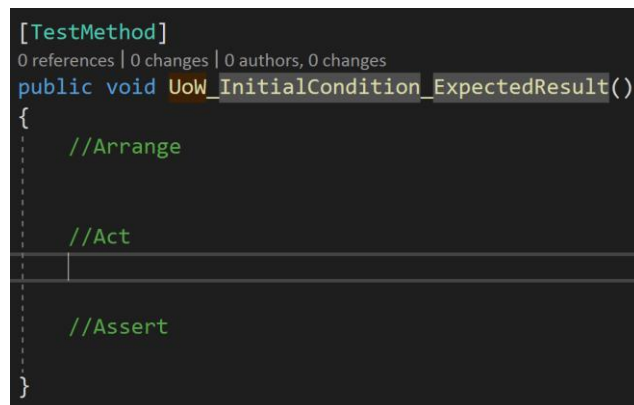
```
%USERPROFILE%\Documents\Visual Studio 2019\Code Snippets\Visual C#\My Code Snippets\
```

2. Name your snippet something like unittest.snippet.
3. In Visual Studio, you will then just type **ut** and hit tab twice to use this snippet.
4. If you have ReSharper installed, you will either need to switch back to Visual Studio Intellisense in ReSharper's options or use Ctrl+K, X to bring up the snippet menu.

Task 1.2: Add a state-based unit test around turning on all online devices in the controller

Stubs provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as whether the device is turned on.

1. Add a Unit Test Project by right clicking to the TestDouble solution and choose from contextual menu add -> new project
2. Choose MS Test Project (.Net Core) and name the project UnitTestProject
3. Delete and add a new DeviceControllerTest class to that project and using the **"ut"** code snippet to scaffold your test method as below:



```
[TestMethod]
0 references | 0 changes | 0 authors, 0 changes
public void UoW_InitialCondition_ExpectedResult()
{
    //Arrange

    //Act

    //Assert
}
```

4. Give your test method a meaningful name like
TurnOnDevices_FromAllDevices_OnlyOnlineShouldBeTurnedOn

5. Create a (nested) FakeDeviceRepository implementing IDeviceRepository that will expose a property to hold an in-memory list of Devices:

```
private class FakeDeviceRepository : IDeviceRepository
{
    public IEnumerable<Device> AllDevices { get; set; }
}
```

6. Next introduce a stub Device engine that will keep track on whether the device got switched on/off by overriding the TurnOn() method.
7. Now utilise the FakeRepository and Stub Devices to setup everything required in your Arrange block to create an instance of your DeviceController (SUT). Apart from the FakeRepository you can ignore all other dependencies

```
var sut = new DeviceController(fakeDeviceRepository, null, null);
```

Note: Please come up with a few offline devices in your collection for that test and subscribe to the many principle (3 minimum items to cover start, middle and end)

8. Write an assertion that verify all online devices are turned on and the offline one remained turned off.

Task 1.2: Add a value-based unit test that verifies GetTimeOfDay returns correct portion of day based on passed date time.

1. Add a new DeviceTest class to the unit test project and add a method using a test method that verifies that when passed 6am, the method should return morning

```
[TestMethod]
public void GetTimeOfDay_For6AM_ReturnsMorning()
{
    // Arrange phase is empty: testing static method, nothing to initialize
    var sut = new Device(Guid.NewGuid(), "MyTV", null);

    // Act
    var result = sut.GetTimeOfDay(new DateTime(2019, 08, 06, 06, 00, 00));

    // Assert
    Assert.AreEqual(TimeOfDay.Morning, result);
}
```

2. Value based unit tests are perfect candidate to be driven by data, let's extend this test to verify all scenario using MSTest's [DataRow] attribute.
Let's rename the test method name above and change its signature to accept hour and expected time of day:

```
[TestMethod]
[DataRow(0, TimeOfDay.Night)]
[DataRow(6, TimeOfDay.Morning)]
...
public void GetTimeOfDay_ForDateTime_ReturnsTimeOfDay(int h, TimeOfDay expected)
```

Exercise 2: Writing unit tests using a Mocking Framework

Introduction

- 1) Use a testing framework like NSubstitute
- 2) Write a behaviour or interaction-based unit test

Tasks 2.1: Installing Mocking Framework

We are going to use a Mocking framework for the rest of this labs there a variety of open source out there. However it is important to remember that all this mocking framework create on demand Fakes, Stubs and perform complex proxy operations under the hood to cater for complex interaction based or a combination of the former test doubles scenarios.

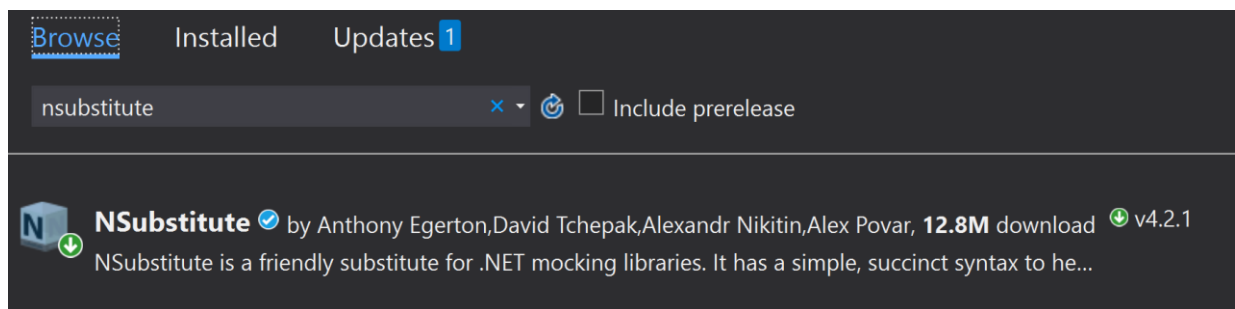
For simplicity we chose a mocking framework like NSubstitute which is designed for Arrange-Act-Assert (AAA) testing, by minimising the noise around setting up your SUT (System Under Test) and help with asserting expected behaviour.

<https://nsubstitute.github.io/>

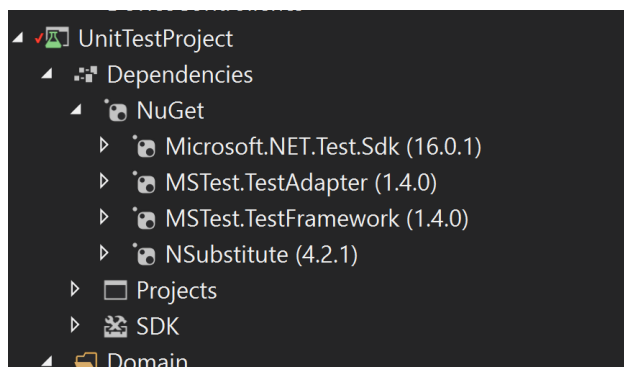
We are going to install NSubstitute using NuGet which a well know package management

<https://docs.microsoft.com/en-us/nuget/what-is-nuget>

1. In Visual Studio right click on the UnitTest Project, select Manage NuGet Dependencies choose the Browse tab and type NSubstitute (casing doesn't matter)



2. Choose NSubstitute and accept the dependencies acknowledgement dialogue to install the package. Your NuGet Dependencies tree should look as below:



Tasks 2.2: Writing an interaction-based unit test

1. Open the DeviceControllerTests in the UnitTest Project and add a new test method (using our “ut2” code snippet)
2. Name the test method according
3. Write an interaction based test that verify that updating firmware require to fetch the latest firmware for that device and needs to be applied to it *(the logic to ensure that only offline device can have their firmware updated is encapsulated within the Device domain class)*

```
[TestMethod]
public void UpdateFirmWare_FetchLatestFirmware_ApplyToDevice()
{
    //Arrange
    var fetcher = Substitute.For<IDeviceFirmwareFetcher>();
    var repo = Substitute.For<IDeviceRepository>();
    var device = new StubDevice("MyDevice", false);

    repo.AllDevices.Returns(new[] { device });
    var sut = new DeviceController(repo, fetcher, new FakeDateTimeProvider());

    //Act
    sut.UpdateFirmWare();

    //Assert
    deviceFirmwareFetcher.Received(1).GetLatestFirmwareFor(device);
}
```

Tasks 2.3: Partial Mocking and throwing exception with NSubstitute

1. Add a new test to DeviceTest (using “ut” code snippet) that verifies that Updating firmware on online device is aborted if device could not be turned on

2. Construct a partially mocked Device with Substitute as follow

```
var sut = Substitute.ForPartsOf<Device>(Guid.NewGuid(), "SomeDevice",
factoryFirmware, true);
```

3. Throw an exception when attempting to set the device online as follow:

```
sut
    .When(x => x.SetOnLine(true))
    .Do(x => throw new Exception("Could not turn device on"));
```

4. Finally assert that the device’s current firmware is equal to the initial factoryFirmware

```
//Assert
Assert.AreEqual(sut.CurrentFirmware, factoryFirmware);
```