Module "C#"
Submodule "Unit Testing"

UA Resource Development Unit

2020

AGENDA

1. Types of Testing
2. Unit test frameworks and extensions
3. Triple A
4. Test-Driven Development (TDD)
5. Stub, Mock, Fake

Types of Testing

Types of Testing: What is software testing?

Software testing is an organizational process within software development in which business-critical software is verified for correctness, quality, and performance. Software testing is used to ensure that expected business systems and product features behave correctly as expected.

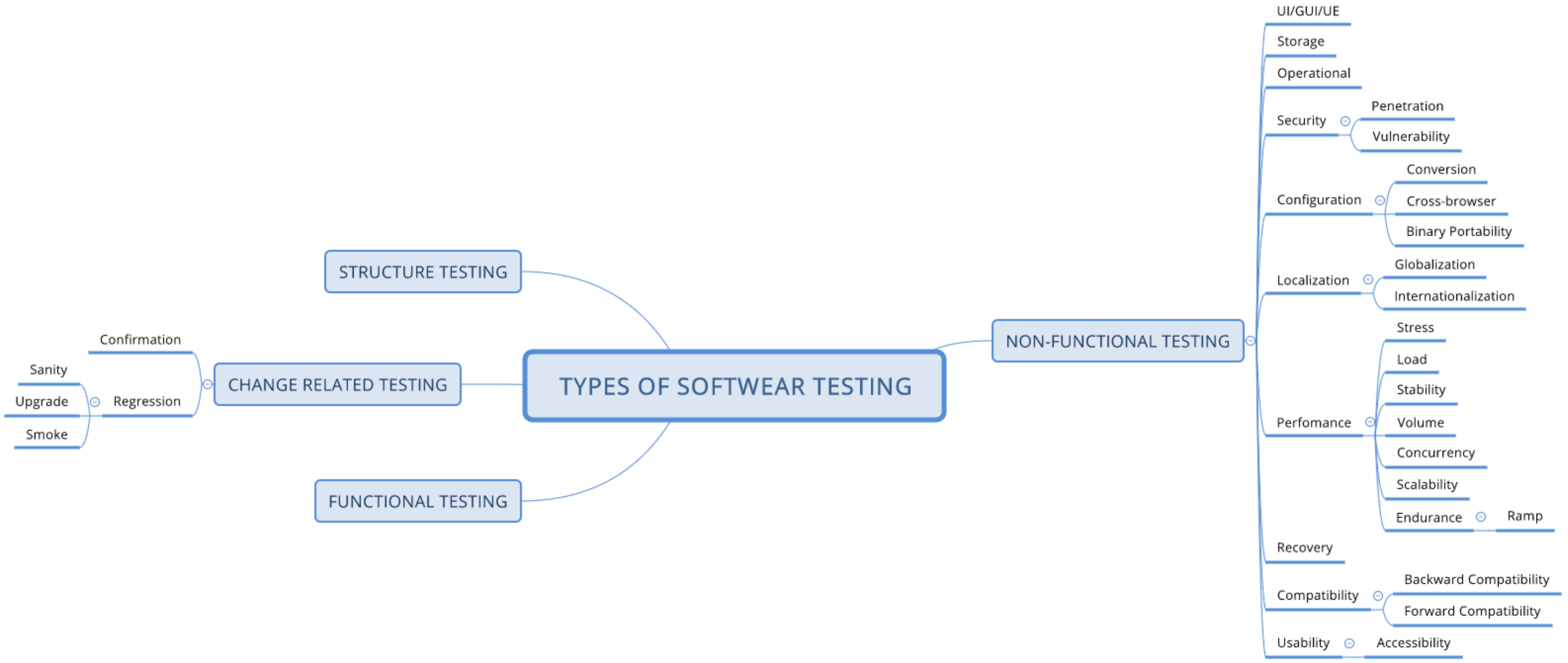
- **Manual software testing** is led by a team or individual who will manually operate a software product and ensure it behaves as expected.

- **Automated software testing** is composed of many different tools which have varying capabilities, ranging from isolated code correctness checks to simulating a full human-driven manual testing experience.

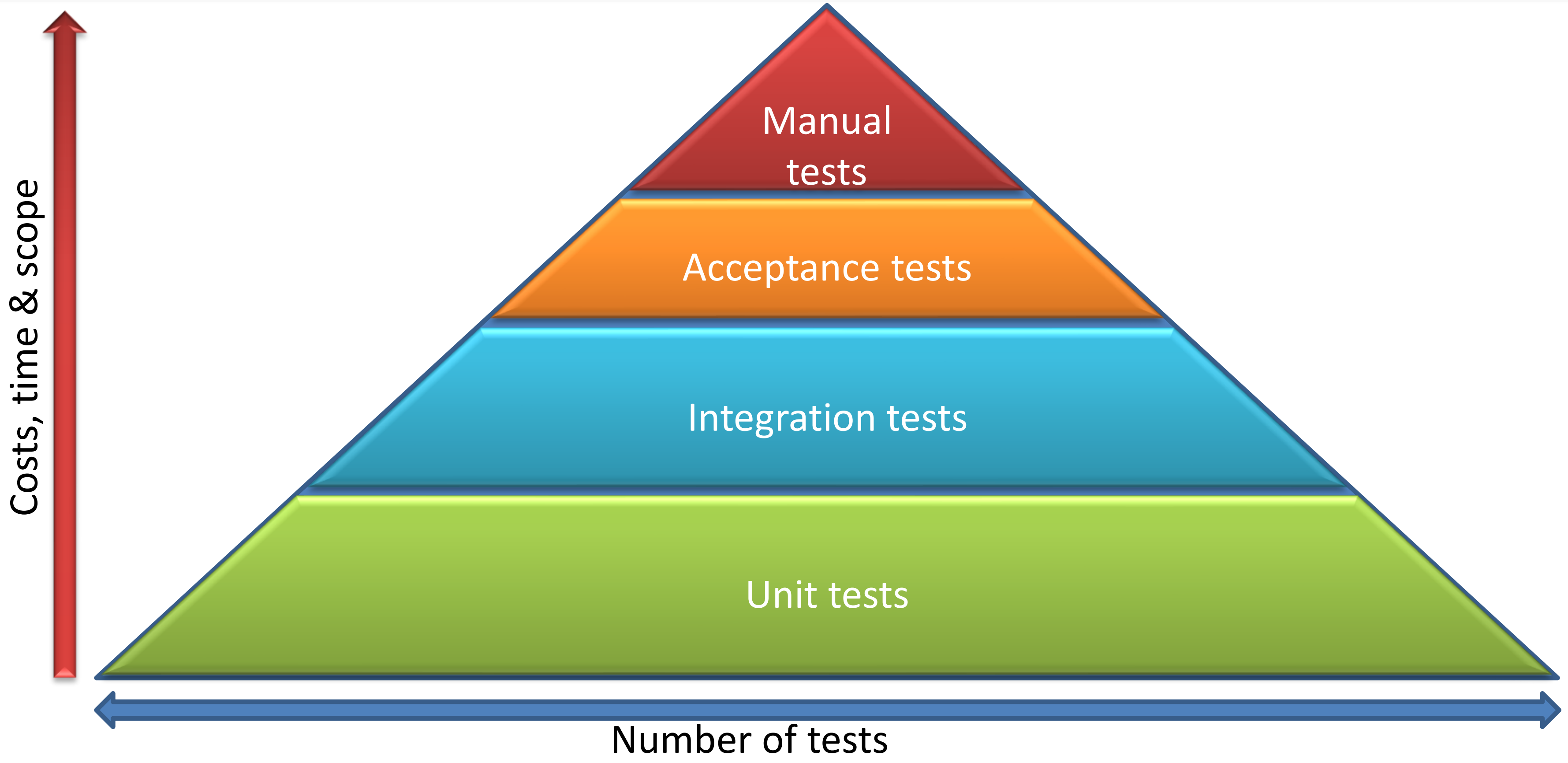
Types of Testing

Unit tests	are very low level, close to the source of your application. They consist in testing individual methods and functions of the classes, components or modules used by your software. Unit tests are in general quite cheap to automate and can be run very quickly by a continuous integration server.
Integration tests	verify that different modules or services used by your application work well together. These types of tests are more expensive to run as they require multiple parts of the application to be up and running.
Acceptance testing	are formal tests executed to verify if a system satisfies its business requirements. They require the entire application to be up and running and focus on replicating user behaviors. But they can also go further and measure the performance of the system and reject changes if certain goals are not met.
Manual tests	

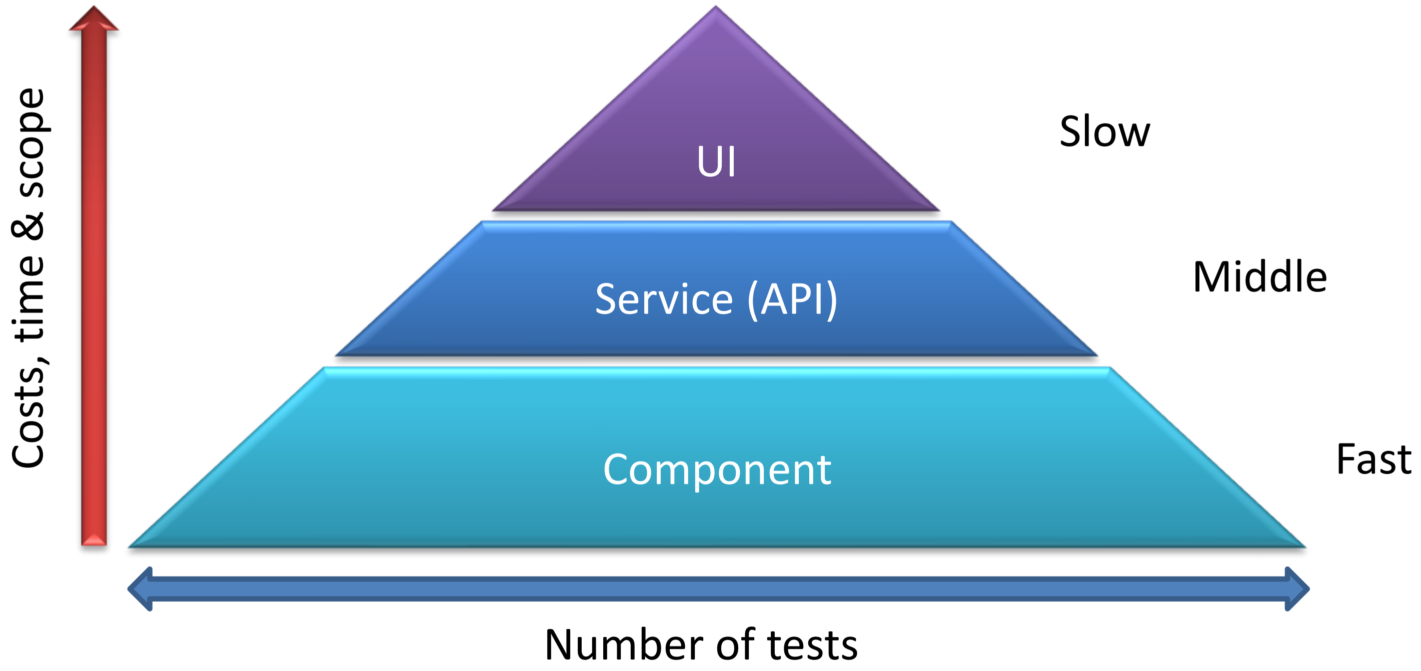
Types of Testing



Types of Testing: Testing pyramid

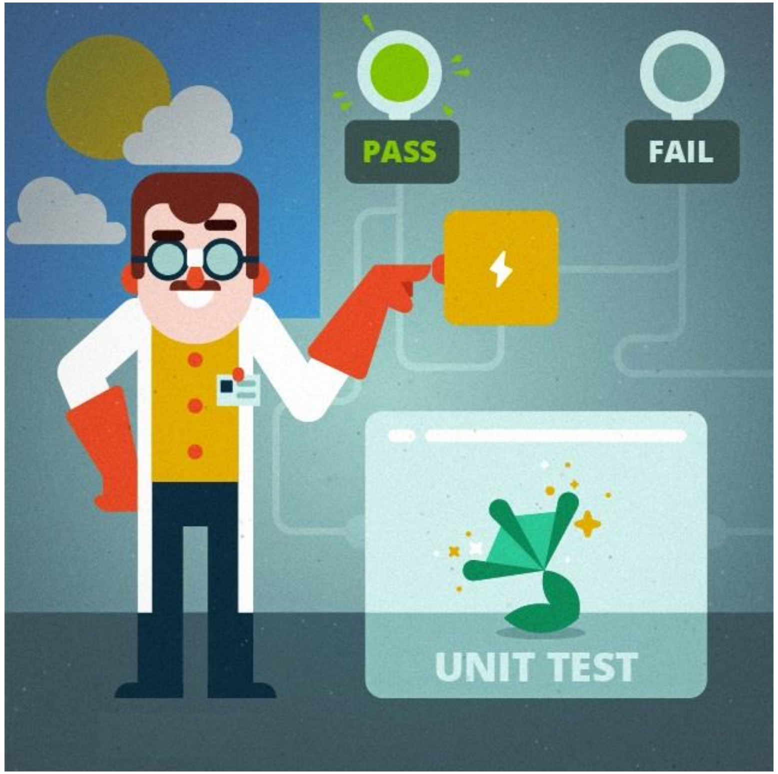


Types of Testing: Testing pyramid by layer



Types of Testing: What a unit test tests?

Unit test is a code that can check whether another code works as expected.



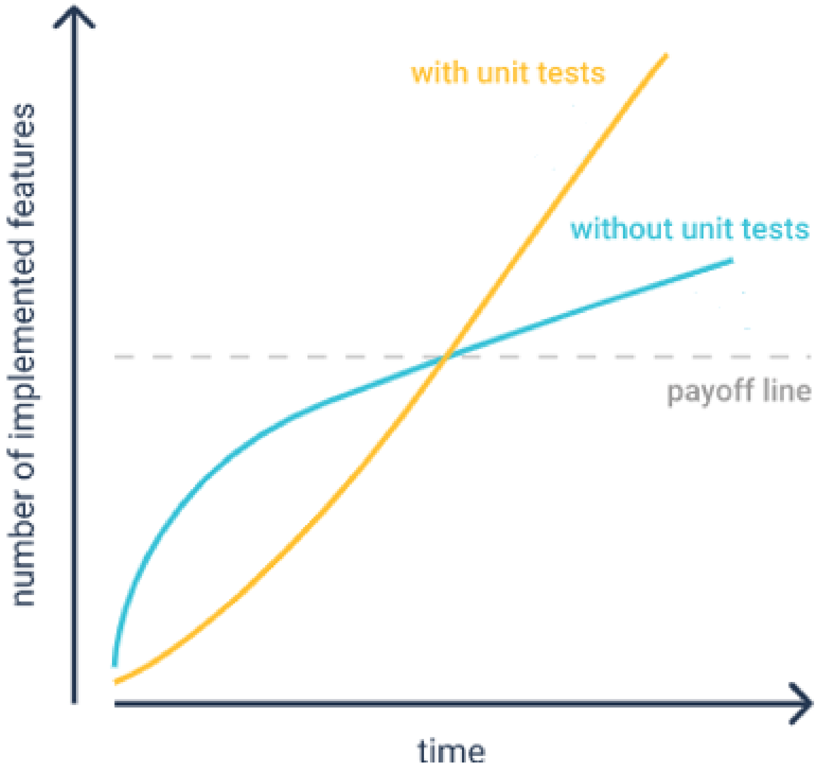
Among the important qualities of a Unit Test are the following:

- Tests functionality of smallest application elements
- Written by developers
- Easy to run in IDE
- Take a few minutes or seconds to run
- Easily integrated with CI

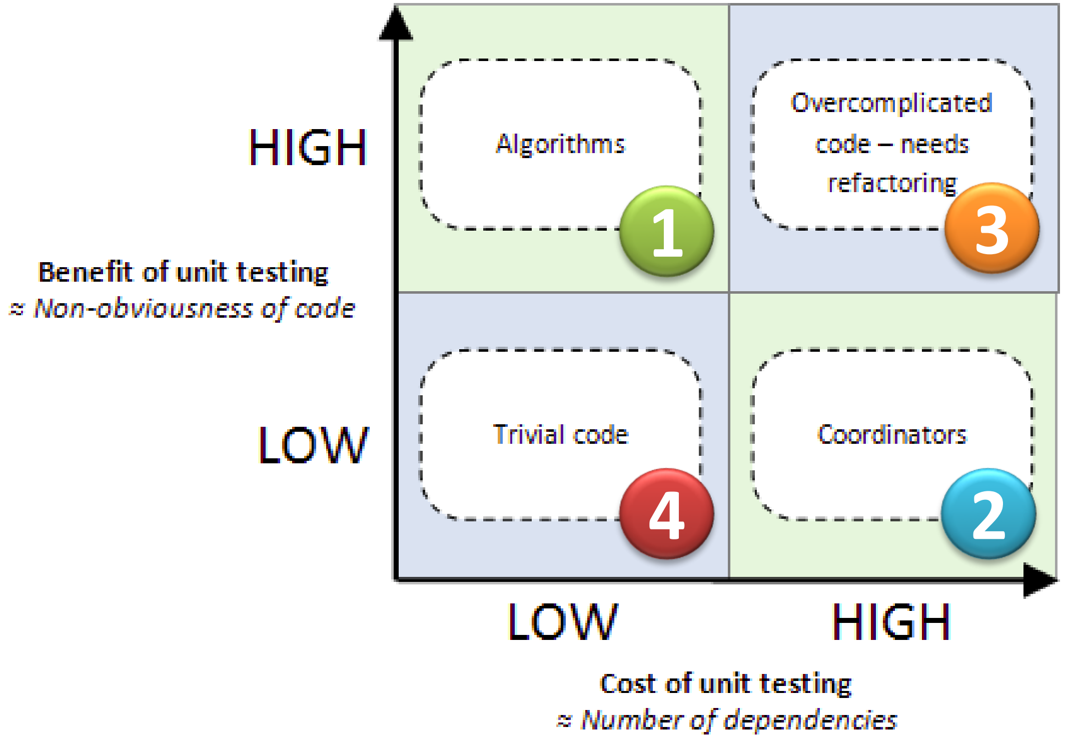
Types of Testing: Why?

- Unit Tests reduces the level of bugs in production code
- Unit Tests save you development time
- Unit Tests save time in debugging later
- Automated Unit Tests can be run as frequently as required with different set of input
- A good Unit Tests are a form of documentation
- Unit Tests allow you to make big changes to code quickly
- Unit Tests help you really understand the design of the code you are working on. Instead of writing code to do somethings, you are starting by outlining all the conditions you are subjecting the code to and what outputs you'd expect from that
- It's first reduces the cost of bugs

Types of Testing



Types of Testing: Unit test quadrant



Types of Testing: FIRST principle

F	**FAST**	run (subset of) tests quickly (since you'll be running them all the time)
I	**INDEPENDENT (or ISOLATED)**	no tests depend on others, so can run any subset in any order
R	**Repeatable**	run N times, get same result (to help isolate bugs and enable automation)
S	**SELF-VALIDATING**	test can automatically detect if passed (no human checking of output)
T	**TIMELY**	written about the same time as code under test (with TDD, written first!)

Types of Testing

When you should and shouldn't write unit tests?

Triple A
(arrange, act, assert)

TDD

The AAA («Triple A», Arrange-Act-Assert) pattern has become almost a standard across the industry.
It suggests that you should divide your test method into three sections:

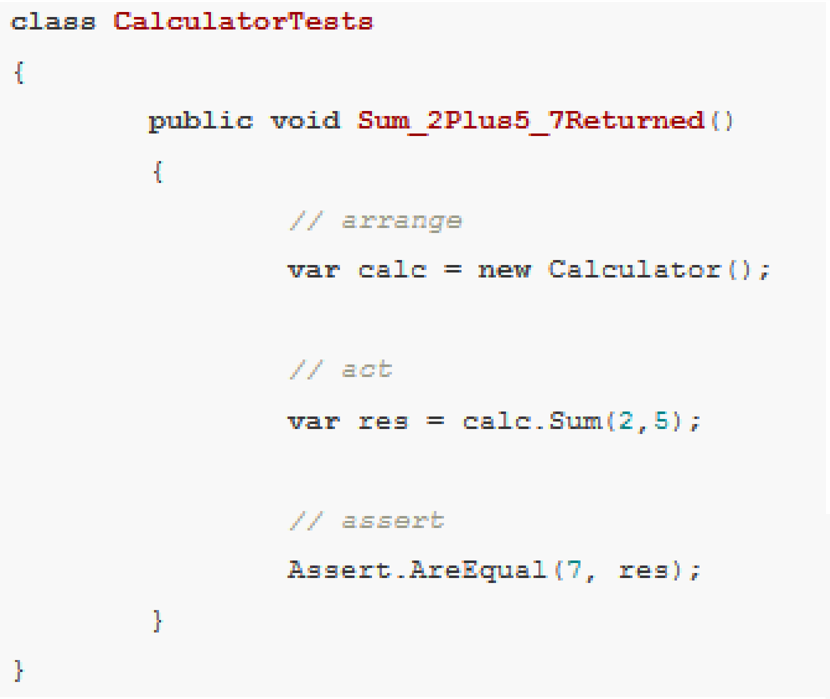
- **A**rrange – setup the testing objects and prepare the prerequisites for your test
- **A**ct – perform the actual work of the test
- **A**ssert – verify the result

Each one of them only responsible for the part in which they are named after.

Triple A

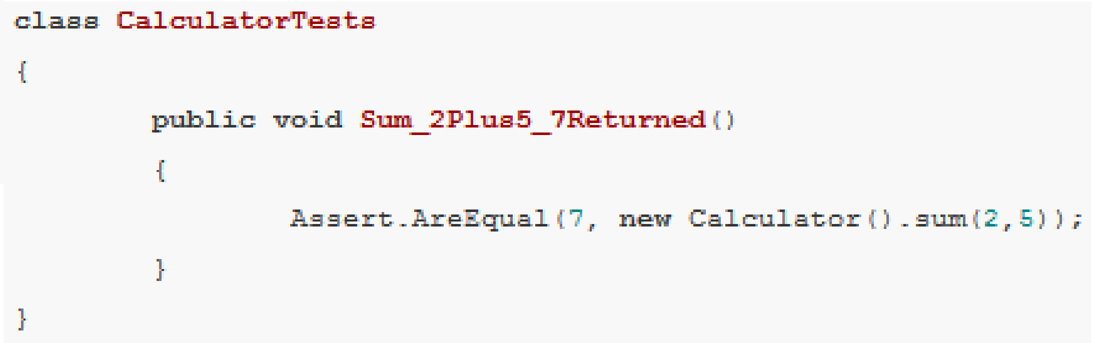
```
// arrange
var repository = Substitute.For<IClientRepository>();
var client = new Client(repository);

// act
client.Save();

// assert
mock.Received.SomeMethod();
```

Triple A



```
class CalculatorTests
{
        public void Sum_2Plus5_7Returned()
        {
                // arrange
                var calc = new Calculator();

                // act
                var res = calc.Sum(2,5);

                // assert
                Assert.AreEqual(7, res);
        }
}
```

VS



```
class CalculatorTests
{
        public void Sum_2Plus5_7Returned()
        {
                Assert.AreEqual(7, new Calculator().sum(2,5));
        }
}
```

Best practices: Naming your tests

The name of your test should consist of three parts:

- The name of the method being tested.
- The scenario under which it's being tested.
- The expected behavior when the scenario is invoked.

Best practices: Naming your tests

```
[Fact]
public void Test_Single()
{
    var stringCalculator = new StringCalculator();
    var actual = stringCalculator.Add("0");
    Assert.Equal(0, actual);
}
```

X

+

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();
    var actual = stringCalculator.Add("0");
    Assert.Equal(0, actual);
}
```

Best practices: Arranging your tests

Arrange, Act, Assert is a common pattern when unit testing.

As the name implies, it consists of three main actions:

- *Arrange* your objects, creating and setting them up as necessary.
- *Act* on an object.
- *Assert* that something is as expected.

Best practices: Arranging your tests

```
[Fact]
public void Add_EmptyString_ReturnsZero()
{
    // Arrange
    var stringCalculator = new StringCalculator();

    // Assert
    Assert.Equal(0, stringCalculator.Add(""));
}
```

X

```
[Fact]
public void Add_EmptyString_ReturnsZero()
{
    // Arrange
    var stringCalculator = new StringCalculator();

    // Act
    var actual = stringCalculator.Add("");

    // Assert
    Assert.Equal(0, actual);
}
```

+

Best practices: Write minimally passing tests

The input to be used in a unit test should be the simplest possible in order to verify the behavior that you are currently testing.

Best practices: Write minimally passing tests

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();
    var actual = stringCalculator.Add("42");
    Assert.Equal(42, actual);
}
```

🟥 X

🟩 +

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();
    var actual = stringCalculator.Add("0");
    Assert.Equal(0, actual);
}
```

Best practices: Avoid magic strings

Naming variables in unit tests is as important, if not more important, than naming variables in production code. Unit tests should not contain magic strings.

Best practices: Avoid magic strings

```
[Fact]
public void Add_BigNumber_ThrowsException()
{
      var stringCalculator = new StringCalculator();
      Action actual = () =>
      stringCalculator.Add("1001");
      Assert.Throws<OverflowException>(actual);
}
```

X

```
[Fact]
void Add_MaximumSumResult_ThrowsOverflowException()
{
      var stringCalculator = new StringCalculator();
      const string MAXIMUM_RESULT = "1001";
      Action actual = () => stringCalculator.Add(MAXIMUM_RESULT);
      Assert.Throws<OverflowException>(actual);
}
```

+

Best practices: Avoid logic in tests

When writing your unit tests avoid manual string concatenation and logical conditions such as *if*, *while*, *for*, *switch*, etc.

Best practices: Avoid logic in tests

```
[Fact]
public void Add_MultipleNumbers_ReturnsCorrectResults()
{
    var stringCalculator = new StringCalculator();
    var expected = 0; var testCases = new[]
    {
        "0,0,0",
        "0,1,2",
        "1,2,3"
    };
    foreach (var test in testCases)
    {
        Assert.Equal(expected, stringCalculator.Add(test));
        expected += 3;
    }
}
```

X

```
[Theory]
[InlineData("0,0,0", 0)]
[InlineData("0,1,2", 3)]
[InlineData("1,2,3", 6)]
public void Add_MultipleNumbers_ReturnsSumOfNumbers(string input, int expected)
{
    var stringCalculator = new StringCalculator();
    var actual = stringCalculator.Add(input);
    Assert.Equal(expected, actual);
}
```

+

Best practices: Prefer helper methods to setup and teardown

If you require a similar object or state for your tests, prefer a helper method than leveraging Setup and Teardown attributes if they exist.

Best practices: Prefer helper methods to setup and teardown

```
private readonly StringCalculator ;
public StringCalculatorTests()
{
    stringCalculator = new StringCalculator();
}

// more tests...

[Fact]
public void Add_TwoNumbers_ReturnsSumOfNumbers()
{
    var result = stringCalculator.Add("0,1");
    Assert.Equal(1, result);
}
```

X

```
[Fact]
public void Add_TwoNumbers_ReturnsSumOfNumbers()
{
    var stringCalculator = CreateDefaultStringCalculator();
    var actual = stringCalculator.Add("0,1");
    Assert.Equal(1, actual);
}

 // more tests...

private StringCalculator CreateDefaultStringCalculator()
{
    return new StringCalculator();
}
```

+

Best practices: Avoid multiple asserts

When writing your tests, try to only include one Assert per test.

Common approaches to using only one assert include:

- ➢ Create a separate test for each assert.
- ➢ Use parameterized tests.

Best practices: Avoid multiple asserts

```
[Fact]
public void Add_EdgeCases_ThrowsArgumentExceptions()
{
    Assert.Throws<ArgumentException>(() => stringCalculator.Add(null));
    Assert.Throws<ArgumentException>(() => stringCalculator.Add("a"));
}
```

X

```
[Theory]
[InlineData(null)]
[InlineData("a")]
public void Add_InputNullOrAlphabetic_ThrowsArgumentException(string input)
{
    var stringCalculator = new StringCalculator();
    Action actual = () => stringCalculator.Add(input);
    Assert.Throws<ArgumentException>(actual);
}
```

+

Best practices: Validate private methods by unit testing public methods

In most cases, there should not be a need to test a private method. Private methods are an implementation detail. You can think of it this way: private methods never exist in isolation. At some point, there is going to be a public facing method that calls the private method as part of its implementation. What you should care about is the end result of the public method that calls into the private one.

Best practices: Validate private methods by unit testing public methods

```
public string ParseLogLine(string input)
 {
    var sanitizedInput = TrimInput(input);
    return sanitizedInput;
}
private string TrimInput(string input)
{
    return input.Trim();
}
```

```
public void
ParseLogLine_ByDefault_ReturnsTrimmedResult()
{
    var parser = new Parser();
    var result = parser.ParseLogLine(" a ");
    Assert.Equals("a", result);
}
```

Best practices: Stub static references

One of the principles of a unit test is that it must have full control of the system under test. This can be problematic when production code includes calls to static references (for example, DateTime.Now).

```
public int GetDiscountedPrice(int price)
{
    if (DateTime.Now.DayOfWeek == DayOfWeek.Tuesday)
    {
        return price / 2;
    }
    else
    {
        return price;
    }
}
```

Best practices: Stub static references

```
public void GetDiscountedPrice_NotTuesday_ReturnsFullPrice()
{
    var priceCalculator = new PriceCalculator();
    var actual = priceCalculator.GetDiscountedPrice(2);
    Assert.Equals(2, actual)
}

public void GetDiscountedPrice_OnTuesday_ReturnsHalfPrice()
{
    var priceCalculator = new PriceCalculator();
    var actual = priceCalculator.GetDiscountedPrice(2);
    Assert.Equals(1, actual);
}
```

Best practices: Stub static references

```
public interface IDateTimeProvider
{
    DayOfWeek DayOfWeek();
}

public int GetDiscountedPrice(int price, IDateTimeProvider dateTimeProvider)
{
    if (dateTimeProvider.DayOfWeek() == DayOfWeek.Tuesday)
    {
        return price / 2;
    }
    else
    {
        return price;
    }
}
```

Best practices: Stub static references

```
public void GetDiscountedPrice_NotTuesday_ReturnsFullPrice()
{
    var priceCalculator = new PriceCalculator();
    var dateTimeProviderStub = new Mock<IDateTimeProvider>();
    dateTimeProviderStub.Setup(dtp => dtp.DayOfWeek()).Returns(DayOfWeek.Monday);
    var actual = priceCalculator.GetDiscountedPrice(2, dateTimeProviderStub); Assert.Equals(2, actual);
}

public void GetDiscountedPrice_OnTuesday_ReturnsHalfPrice()
{
    var priceCalculator = new PriceCalculator();
    var dateTimeProviderStub = new Mock<IDateTimeProvider>();
    dateTimeProviderStub.Setup(dtp => dtp.DayOfWeek()).Returns(DayOfWeek.Tuesday);
    var actual = priceCalculator.GetDiscountedPrice(2, dateTimeProviderStub);
    Assert.Equals(1, actual);
}
```

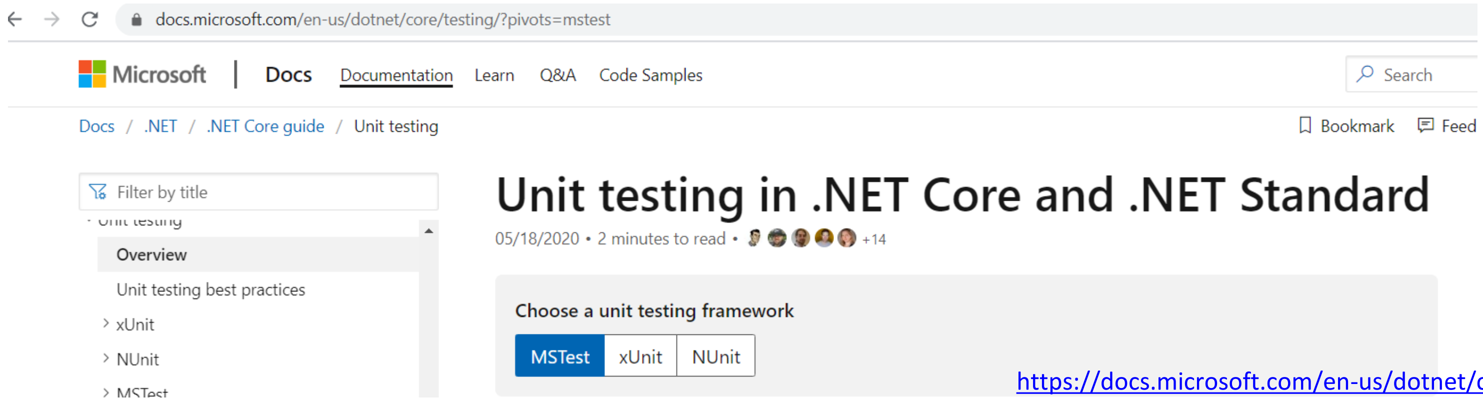
Unit test frameworks and extensions (MSTest, NUnit, XUnit etc.)

Unit test frameworks and extensions

- **NUnit** and **Mb-unit** are unit-testing frameworks for .NET languages
- **MoQ** is a popular and friendly mocking framework
- **Specflow** provides a pragmatic approach to Specification-By-Example for .NET projects
- **MSTest** is a command line utility from Microsoft that executes unit tests created in Visual Studio
- **DotCover** is a coverage tool
- **xUnit.net** is a free, open source, community-focused unit testing tool for the .NET Framework.

Unit test frameworks and extensions

- **MS Unit Testing Framework (Microsoft)** https://github.com/Microsoft/testfx-docs
  Reference: Microsoft.VisualStudio.TestTools.UnitTesting
- **NUnit Framework,** https://nunit.org/
  Reference: Nunit .Framework
  Features: it's opensource. It has no built-in support in Visual Studio
- **xUnit Framework,** https://xunit.net/
  Reference: xunit .Framework
  Features: it's opensource. It has no built-in support in Visual Studio



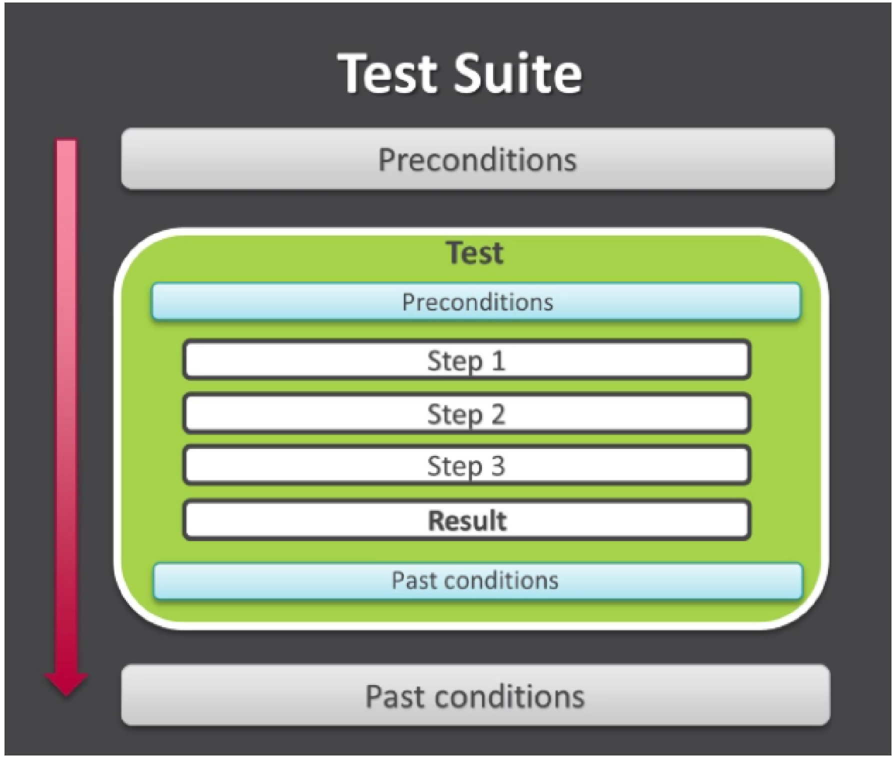
Unit testing in .NET Core and .NET Standard

05/18/2020 • 2 minutes to read • +14

Choose a unit testing framework

https://docs.microsoft.com/en-us/dotnet/core/testing/?pivots=mstest

Test workflow

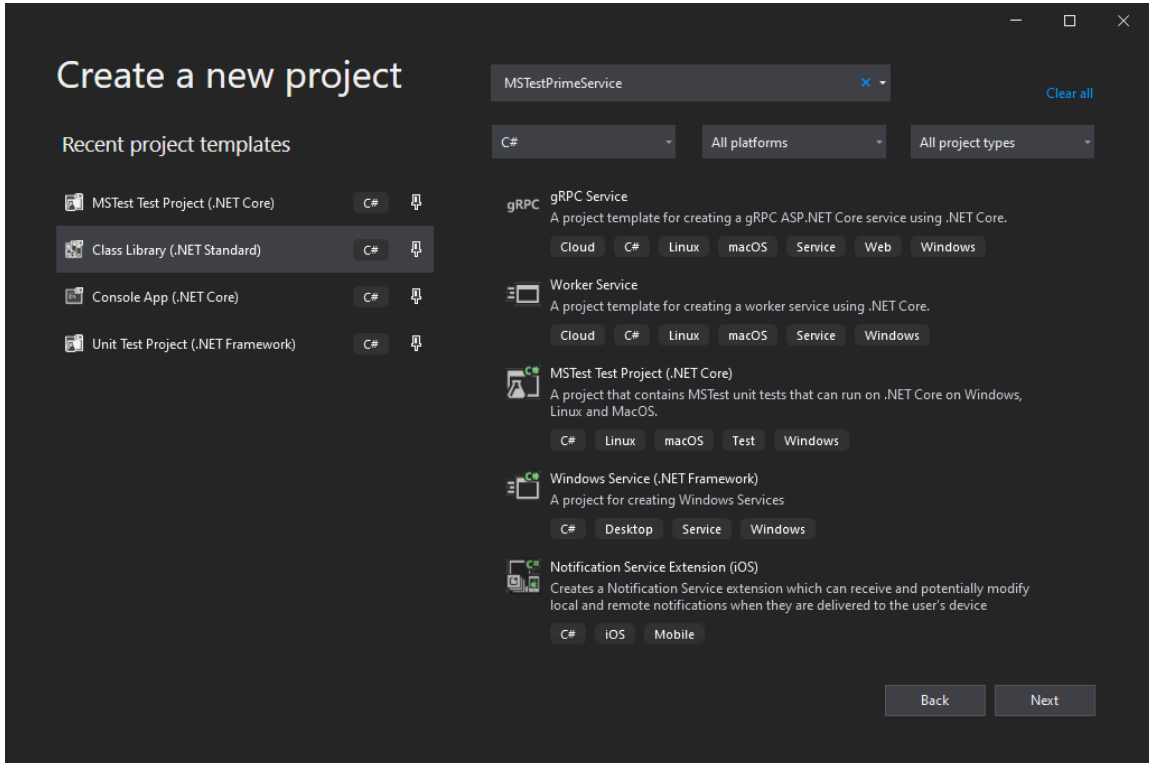


classlib

Unit test frameworks and extensions: MSTest

Unit testing C# with MSTest and .NET Core

1. **Solution:** MSTestPrimeService
2. **Project**: MSTestPrimeService.Services
3. **Template:** Class Library (.NET Standard)


Create a new project

MSTestPrimeService

Clear all

C# | All platforms | All project types

Recent project templates

- MSTest Test Project (.NET Core) C#
- Class Library (.NET Standard) C#
- Console App (.NET Core) C#
- Unit Test Project (.NET Framework) C#

gRPC Service
A project template for creating a gRPC ASP.NET Core service using .NET Core.
Cloud C# Linux macOS Service Web Windows

Worker Service
A project template for creating a worker service using .NET Core.
Cloud C# Linux macOS Service Windows

MSTest Test Project (.NET Core)
A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Test Windows

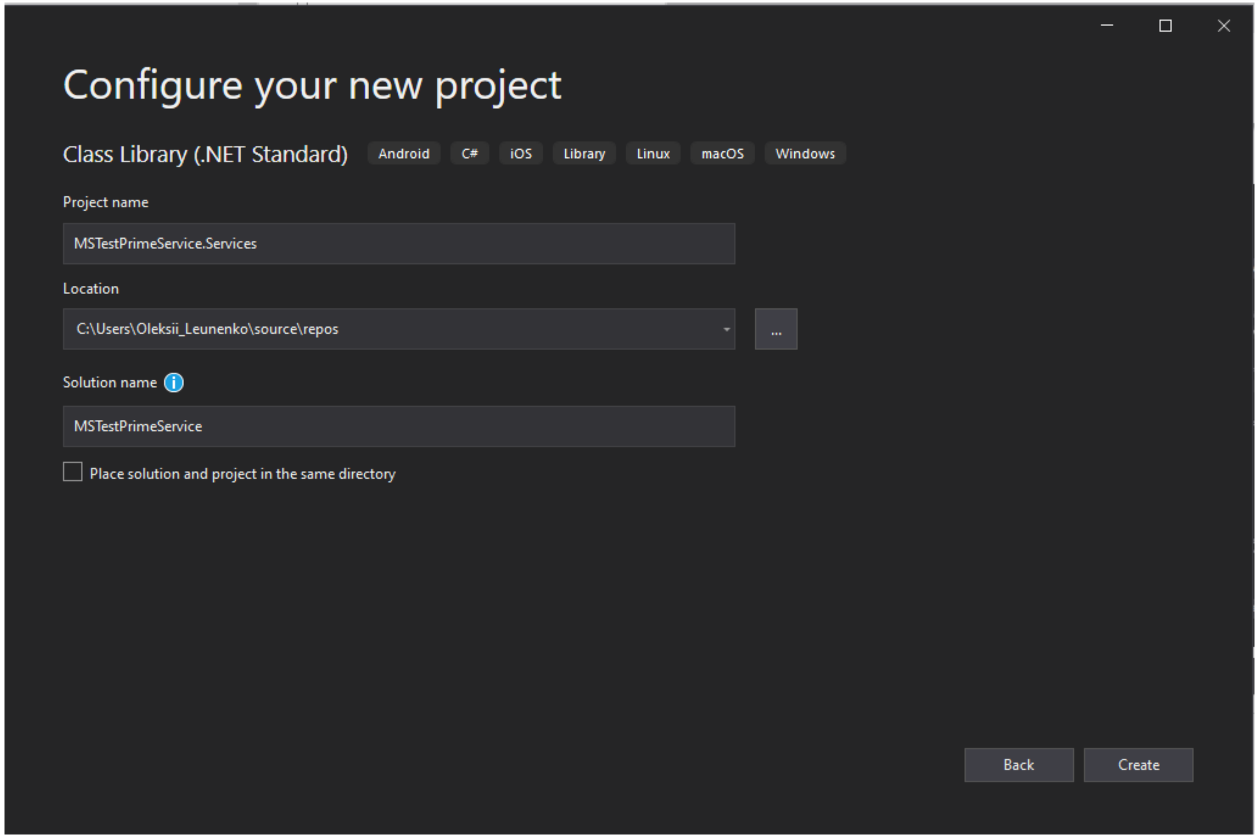
Windows Service (.NET Framework)
A project for creating Windows Services
C# Desktop Service Windows

Notification Service Extension (iOS)
Creates a Notification Service extension which can receive and potentially modify local and remote notifications when they are delivered to the user's device
C# iOS Mobile

Back Next

classlib

Unit test frameworks and extensions: MSTest


Configure your new project

Class Library (.NET Standard) Android C# iOS Library Linux macOS Windows

Project name

MSTestPrimeService.Services

Location

C:\Users\Oleksii_Leunenko\source\repos

Solution name

MSTestPrimeService

☐ Place solution and project in the same directory

Back Create

Unit test frameworks and extensions: MSTest

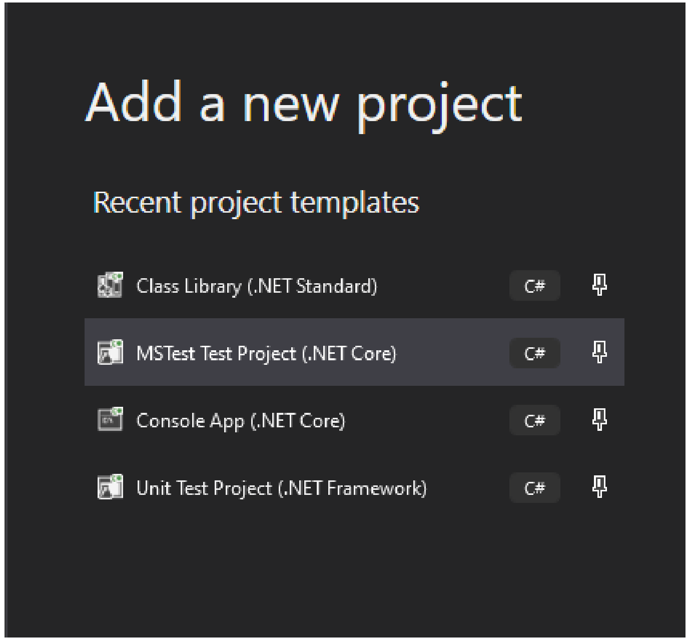
Rename *Class1.cs* to *PrimeService.cs*.

```
using System;

namespace MSTestPrimeService.Services
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

classlib

Unit test frameworks and extensions: MSTest

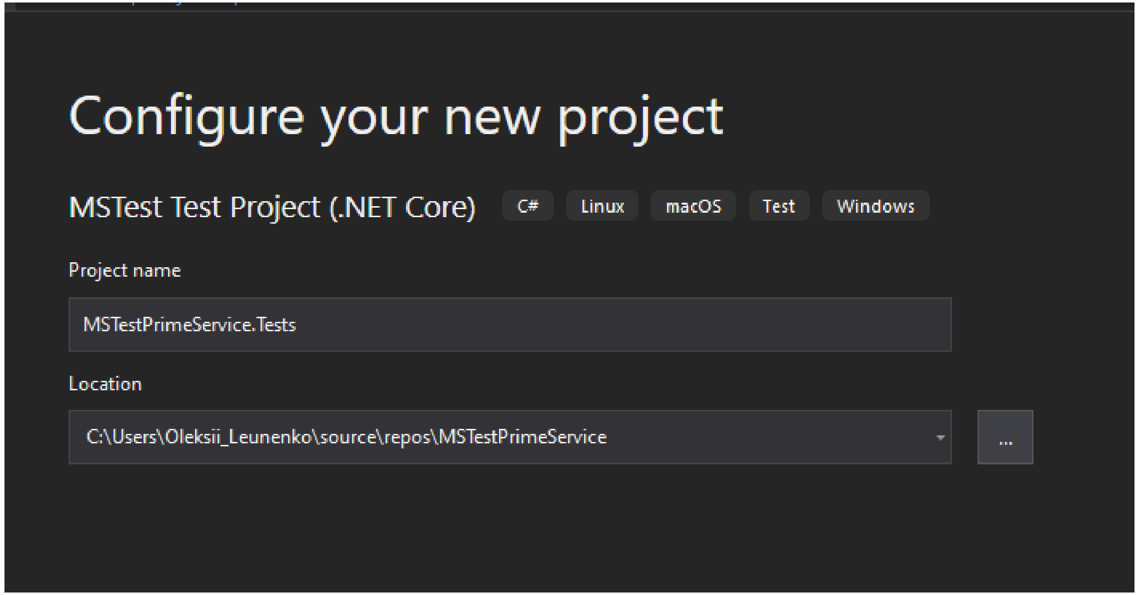
Create the test project



Add a new project

Recent project templates

- Class Library (.NET Standard) C#
- MSTest Test Project (.NET Core) C#
- Console App (.NET Core) C#
- Unit Test Project (.NET Framework) C#



Configure your new project

MSTest Test Project (.NET Core) C# Linux macOS Test Windows

Project name

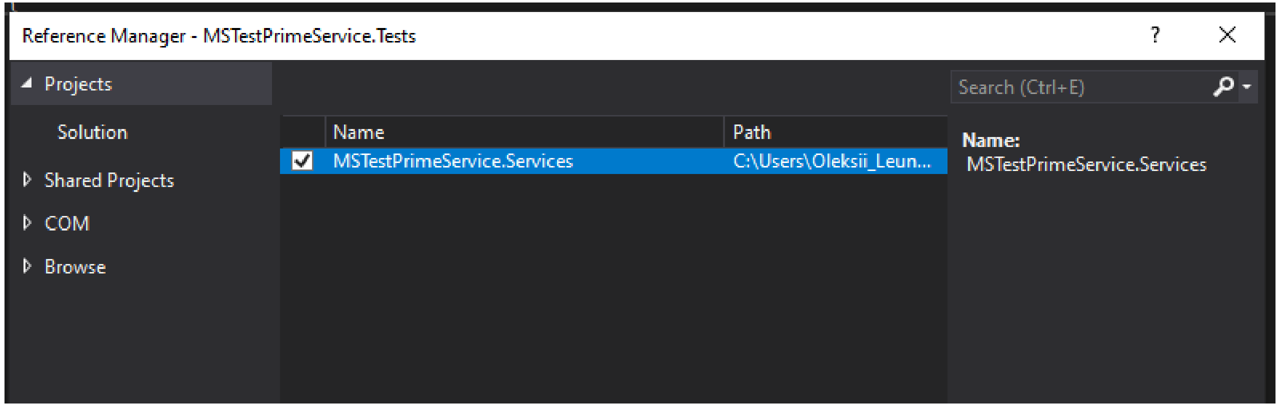
MSTestPrimeService.Tests

Location

C:\Users\Oleksii_Leunenko\source\repos\MSTestPrimeService

classlib

Unit test frameworks and extensions: MSTest


Reference Manager - MSTestPrimeService.Tests

	Name	Path
✓	MSTestPrimeService.Services	C:\Users\Oleksii_Leun...

Name:
MSTestPrimeService.Services

Rename test class *UnitTest1* to *PrimeService_IsPrimeShould*

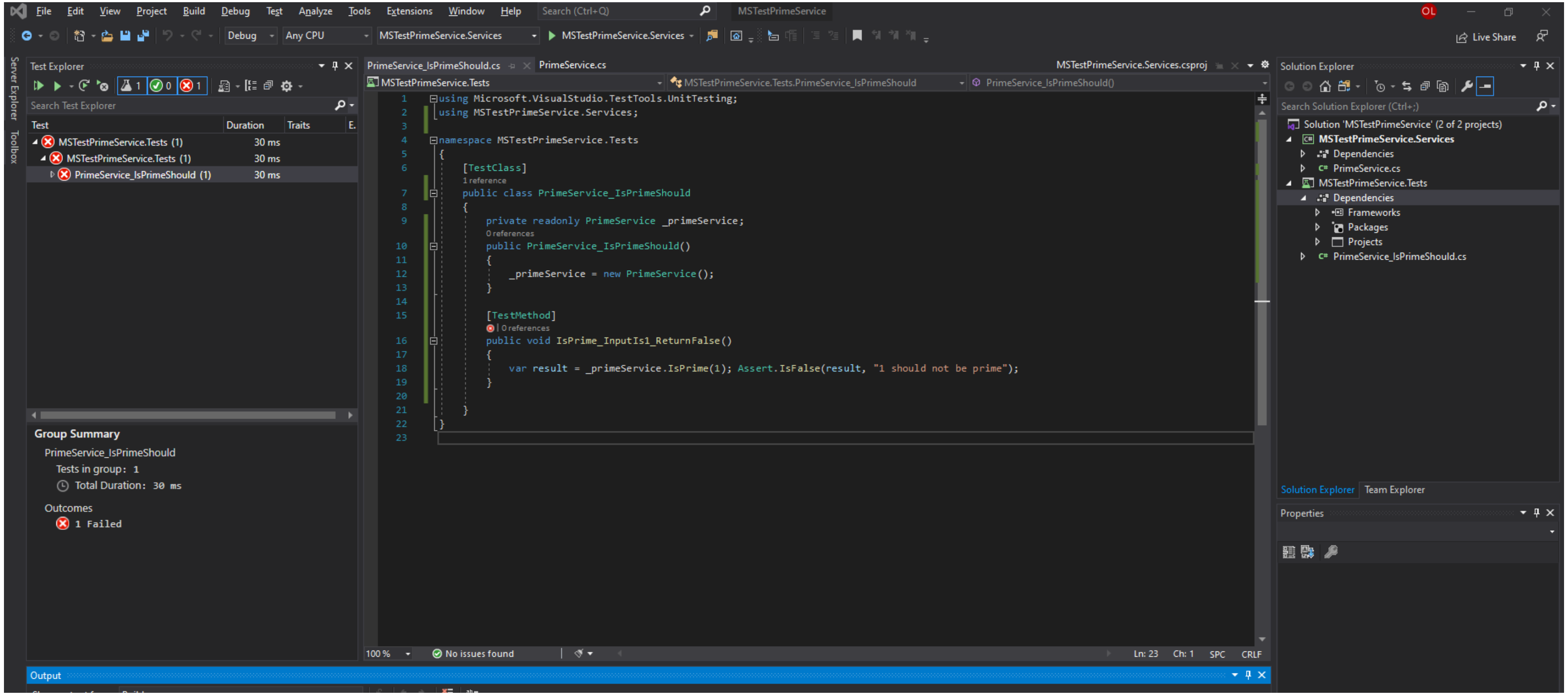
classlib

Unit test frameworks and extensions: MSTest

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MSTestPrimeService.Services;

namespace MSTestPrimeService.Tests
{
    [TestClass]
    public class PrimeService_IsPrimeShould
    {
        private readonly PrimeService _primeService;
        public PrimeService_IsPrimeShould()
        {
            _primeService = new PrimeService();
        }

        [TestMethod]
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var result = _primeService.IsPrime(1); Assert.IsFalse(result, "1 should not be prime");
        }

    }
}
```

classlib

Unit test frameworks and extensions: MSTest



Test	Duration	Traits
MSTestPrimeService.Tests (1)	30 ms	
MSTestPrimeService.Tests (1)	30 ms	
PrimeService_IsPrimeShould (1)	30 ms	

Group Summary

PrimeService_IsPrimeShould
Tests in group: 1
Total Duration: 30 ms

Outcomes
1 Failed

PrimeService_IsPrimeShould.cs | PrimeService.cs | MSTestPrimeService.Services.csproj

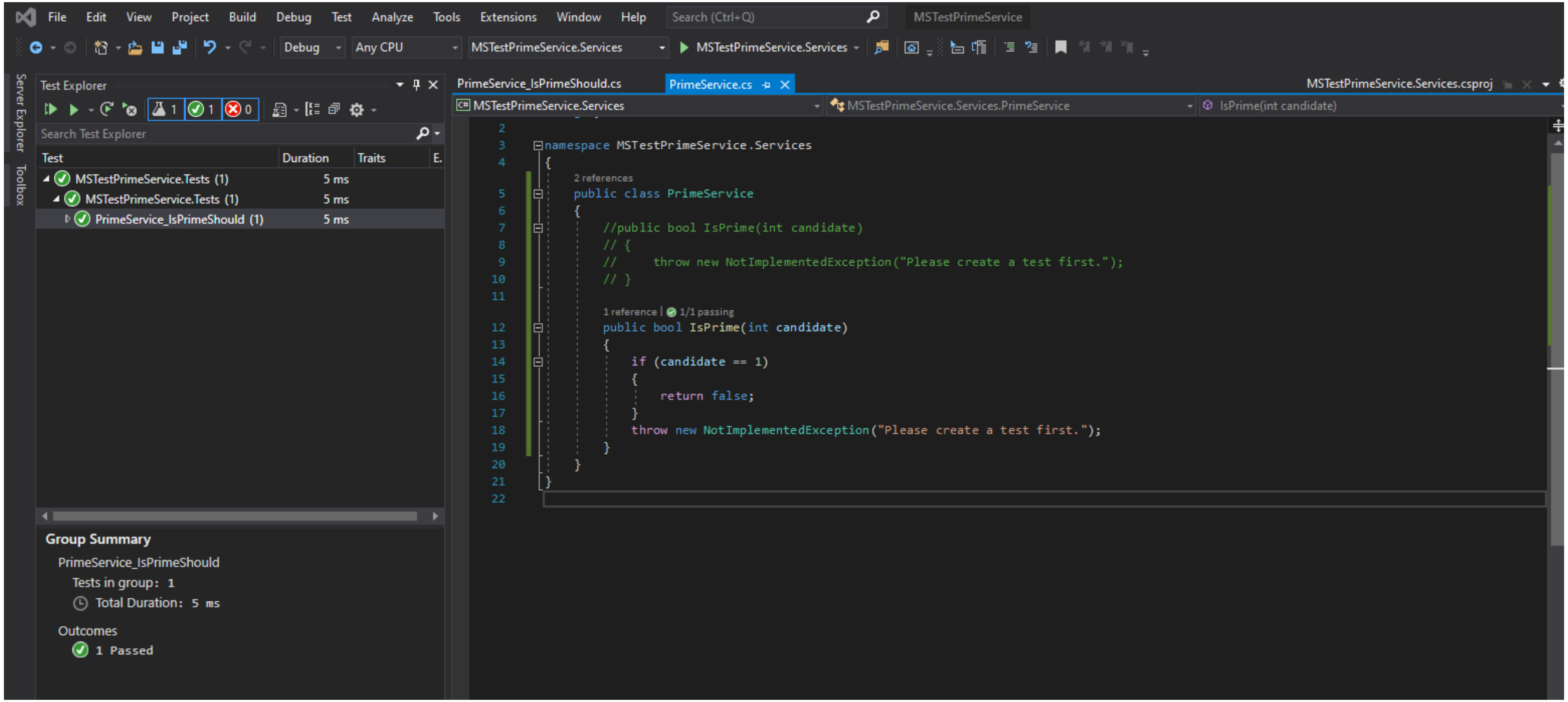
MSTestPrimeService.Tests | MSTestPrimeService.Tests.PrimeService_IsPrimeShould | PrimeService_IsPrimeShould()

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MSTestPrimeService.Services;

namespace MSTestPrimeService.Tests
{
    [TestClass]
    1 reference
    public class PrimeService_IsPrimeShould
    {
        private readonly PrimeService _primeService;
        0 references
        public PrimeService_IsPrimeShould()
        {
            _primeService = new PrimeService();
        }

        [TestMethod]
        | 0 references
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var result = _primeService.IsPrime(1); Assert.IsFalse(result, "1 should not be prime");
        }

    }
}
```

Solution 'MSTestPrimeService' (2 of 2 projects)
- MSTestPrimeService.Services
  - Dependencies
  - PrimeService.cs
- MSTestPrimeService.Tests
  - Dependencies
    - Frameworks
    - Packages
    - Projects
  - PrimeService_IsPrimeShould.cs

classlib

Unit test frameworks and extensions: MSTest



Test	Duration	Traits	E.
MSTestPrimeService.Tests (1)	5 ms		
MSTestPrimeService.Tests (1)	5 ms		
PrimeService_IsPrimeShould (1)	5 ms		

Group Summary

PrimeService_IsPrimeShould

Tests in group: 1

Total Duration: 5 ms

Outcomes

1 Passed

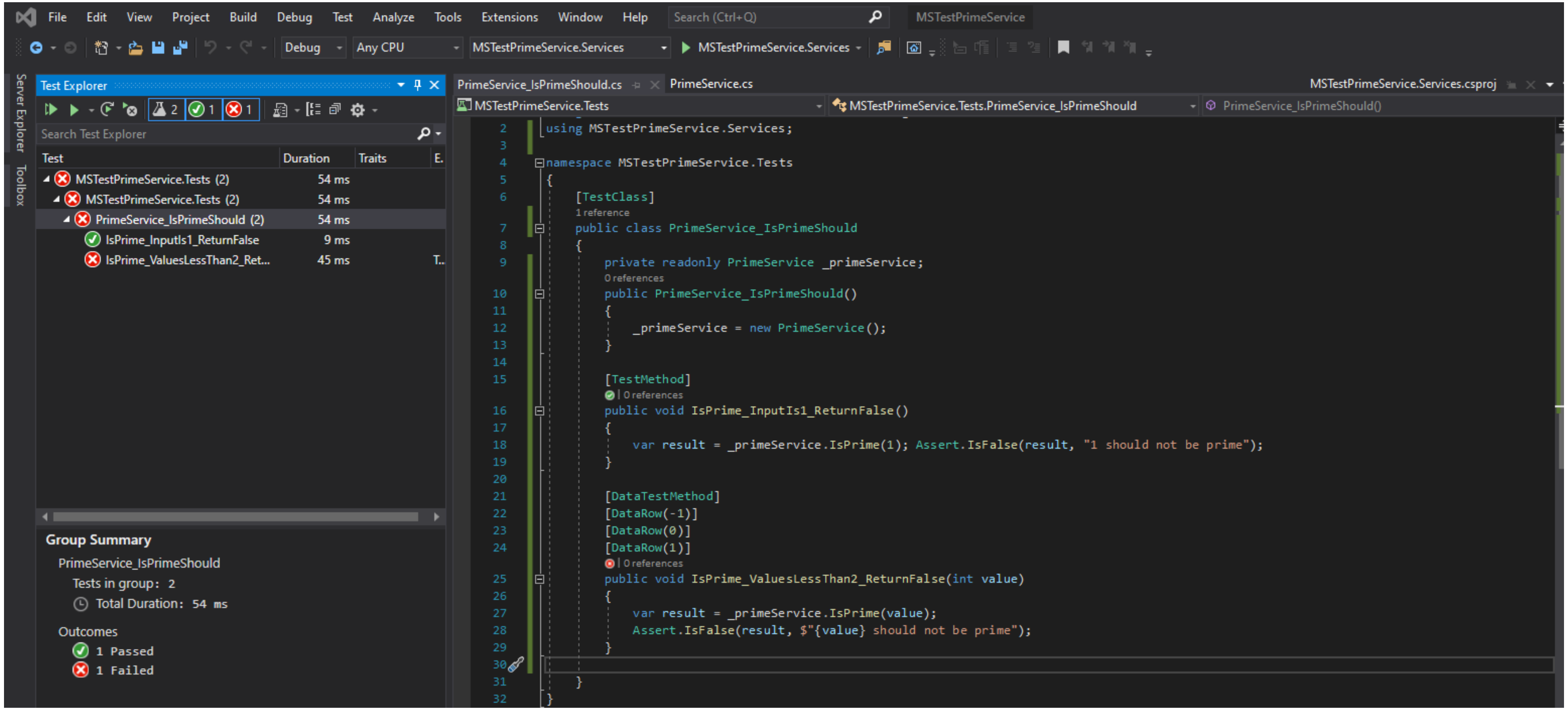
```
namespace MSTestPrimeService.Services
{
    2 references
    public class PrimeService
    {
        //public bool IsPrime(int candidate)
        // {
        //     throw new NotImplementedException("Please create a test first.");
        // }

        1 reference | 1/1 passing
        public bool IsPrime(int candidate)
        {
            if (candidate == 1)
            {
                return false;
            }
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

classlib

Unit test frameworks and extensions: MSTest

```
[DataTestMethod]
[DataRow(-1)]
[DataRow(0)]
[DataRow(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
    var result = _primeService.IsPrime(value);
    Assert.IsFalse(result, $"{value} should not be prime");
}
```

Unit test frameworks and extensions: MSTest


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
MSTestPrimeService
Debug
Any CPU
MSTestPrimeService.Services
Test Explorer
Search Test Explorer
PrimeService_IsPrimeShould.cs
PrimeService.cs
MSTestPrimeService.Services.csproj
MSTestPrimeService.Tests
MSTestPrimeService.Tests.PrimeService_IsPrimeShould
PrimeService_IsPrimeShould()

Test	Duration	Traits	E.
MSTestPrimeService.Tests (2)	54 ms		
MSTestPrimeService.Tests (2)	54 ms		
PrimeService_IsPrimeShould (2)	54 ms		
IsPrime_InputIs1_ReturnFalse	9 ms		
IsPrime_ValuesLessThan2_Ret...	45 ms		T...

Group Summary
PrimeService_IsPrimeShould
Tests in group: 2
Total Duration: 54 ms

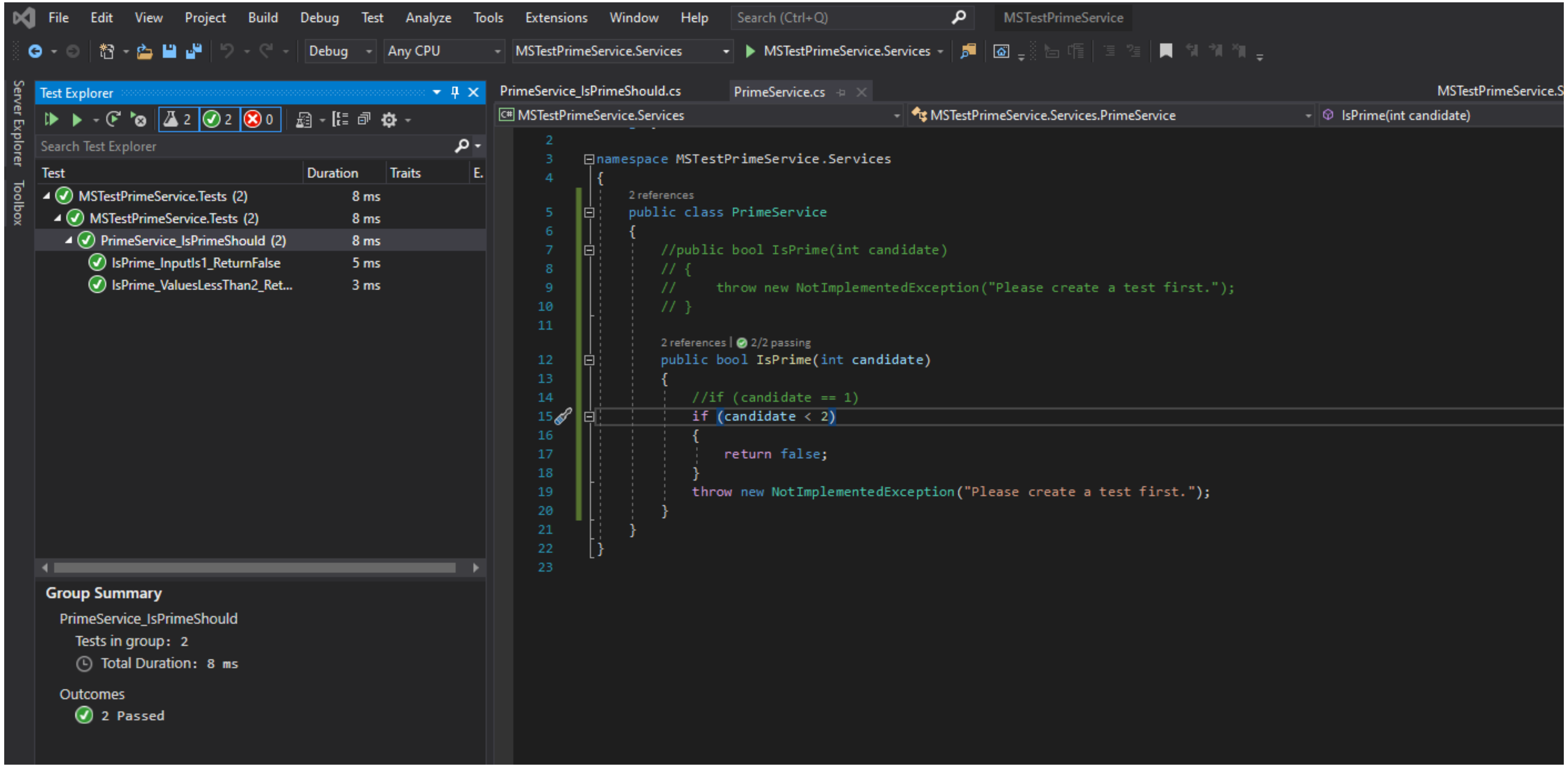
Outcomes
1 Passed
1 Failed

```csharp
using MSTestPrimeService.Services;

namespace MSTestPrimeService.Tests
{
    [TestClass]
    1 reference
    public class PrimeService_IsPrimeShould
    {
        private readonly PrimeService _primeService;
        0 references
        public PrimeService_IsPrimeShould()
        {
            _primeService = new PrimeService();
        }

        [TestMethod]
        0 references
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var result = _primeService.IsPrime(1); Assert.IsFalse(result, "1 should not be prime");
        }

        [DataTestMethod]
        [DataRow(-1)]
        [DataRow(0)]
        [DataRow(1)]
        0 references
        public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
        {
            var result = _primeService.IsPrime(value);
            Assert.IsFalse(result, $"{value} should not be prime");
        }

    }
}
```

if (candidate < 2)

classlib

Unit test frameworks and extensions: MSTest



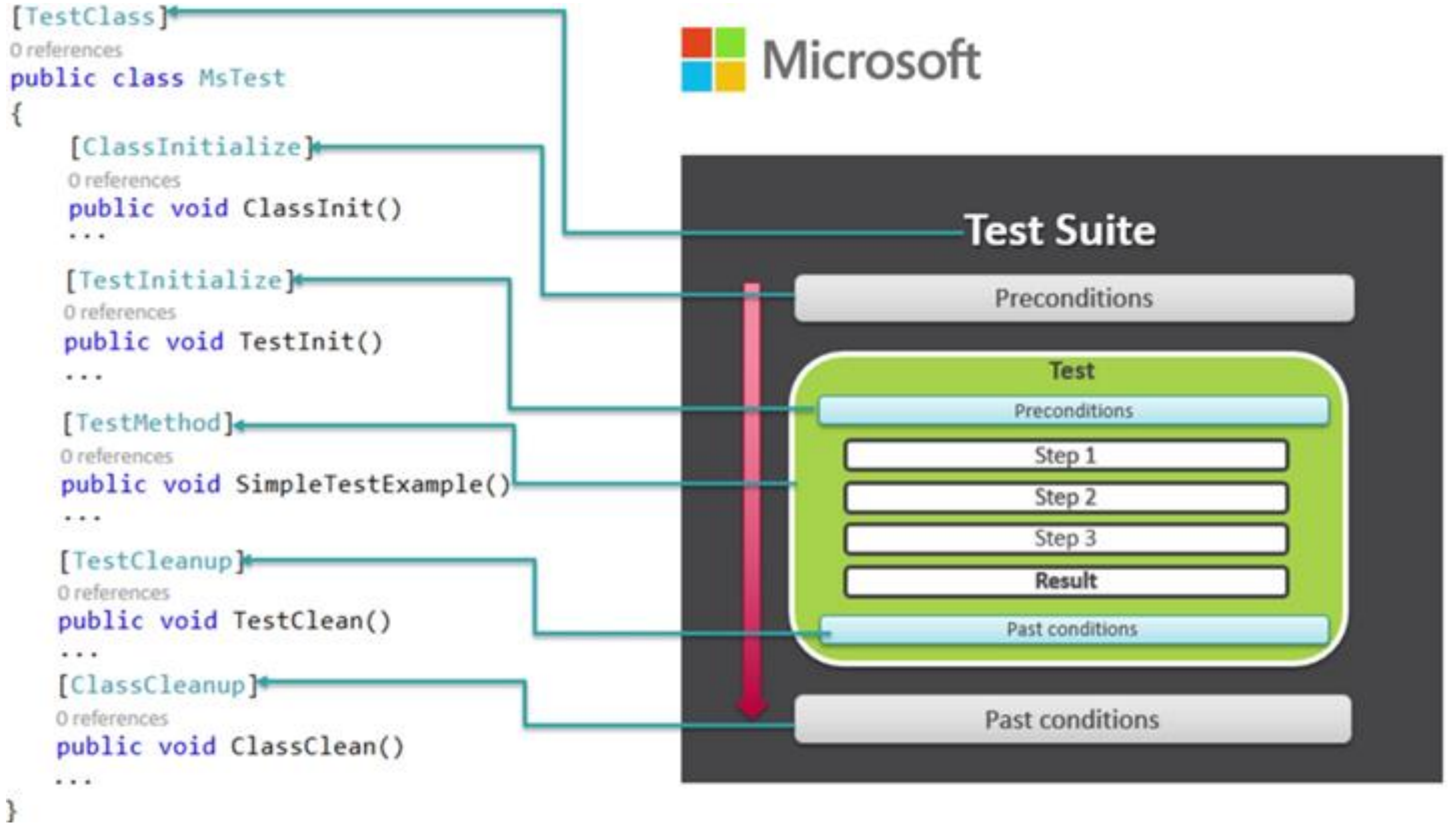
Test	Duration	Traits
MSTestPrimeService.Tests (2)	8 ms	
MSTestPrimeService.Tests (2)	8 ms	
PrimeService_IsPrimeShould (2)	8 ms	
IsPrime_InputIs1_ReturnFalse	5 ms	
IsPrime_ValuesLessThan2_Ret...	3 ms	

Group Summary
PrimeService_IsPrimeShould
Tests in group: 2
Total Duration: 8 ms
Outcomes
2 Passed

```
namespace MSTestPrimeService.Services
{
    2 references
    public class PrimeService
    {
        //public bool IsPrime(int candidate)
        // {
        //     throw new NotImplementedException("Please create a test first.");
        // }

        2 references | 2/2 passing
        public bool IsPrime(int candidate)
        {
            //if (candidate == 1)
            if (candidate < 2)
            {
                return false;
            }
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

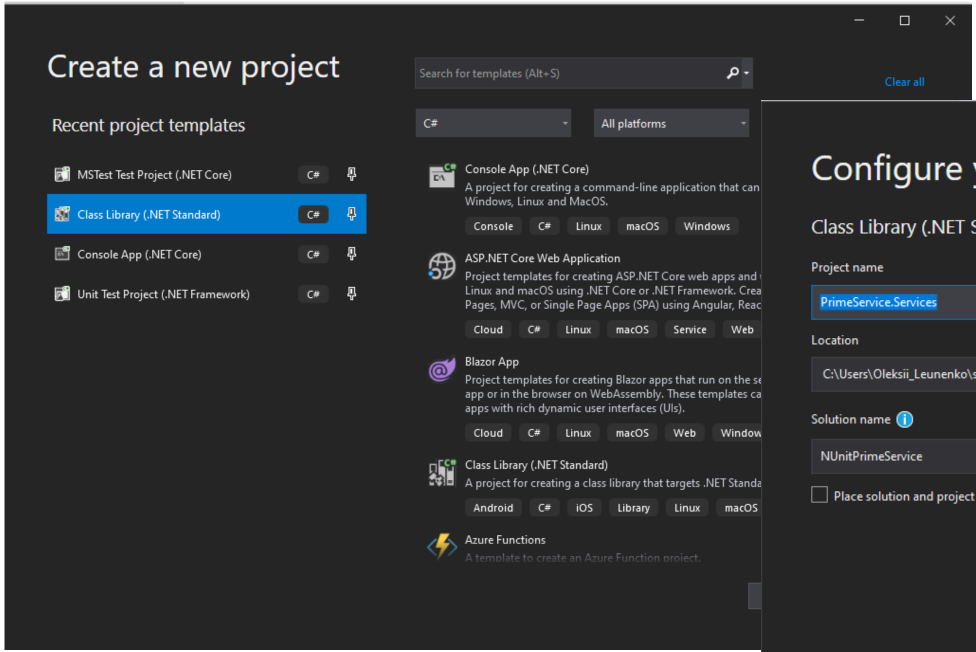
classlib

Unit test frameworks and extensions: MSTest



```
[TestClass]
0 references
public class MsTest
{
    [ClassInitialize]
    0 references
    public void ClassInit()
    ...
    [TestInitialize]
    0 references
    public void TestInit()
    ...
    [TestMethod]
    0 references
    public void SimpleTestExample()
    ...
    [TestCleanup]
    0 references
    public void TestClean()
    ...
    [ClassCleanup]
    0 references
    public void ClassClean()
    ...
}
```

Unit test frameworks and extensions: NUnit



Create a new project

Search for templates (Alt+S)

Clear all

Recent project templates

- MSTest Test Project (.NET Core) C#
- Class Library (.NET Standard) C#
- Console App (.NET Core) C#
- Unit Test Project (.NET Framework) C#

C# | All platforms

Console App (.NET Core)
A project for creating a command-line application that can
Windows, Linux and MacOS.
Console C# Linux macOS Windows

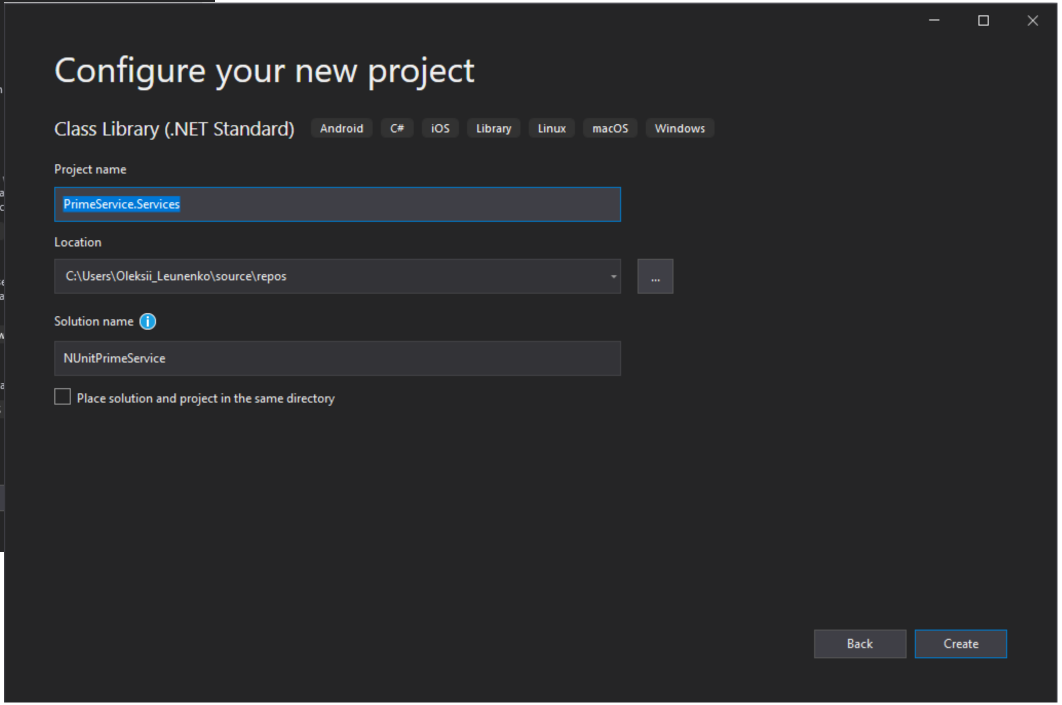
ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and
Linux and macOS using .NET Core or .NET Framework. Crea
Pages, MVC, or Single Page Apps (SPA) using Angular, Reac
Cloud C# Linux macOS Service Web

Blazor App
Project templates for creating Blazor apps that run on the se
app or in the browser on WebAssembly. These templates ca
apps with rich dynamic user interfaces (UIs).
Cloud C# Linux macOS Web Window

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standa
Android C# iOS Library Linux macOS

Azure Functions
A template to create an Azure Function project.

Unit testing C# with NUnit



Configure your new project

Class Library (.NET Standard) Android C# iOS Library Linux macOS Windows

Project name: PrimeService.Services

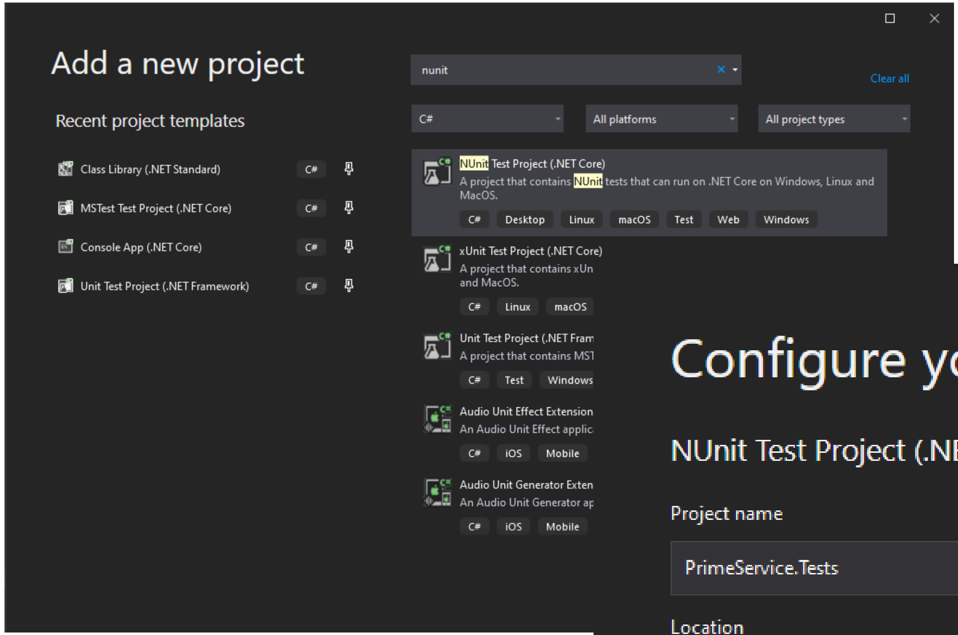
Location: C:\Users\Oleksii_Leunenko\source\repos

Solution name: NUnitPrimeService

☐ Place solution and project in the same directory

Back | Create

Unit test frameworks and extensions: NUnit



Add a new project

nunit

Clear all

Recent project templates

Class Library (.NET Standard) C#

MSTest Test Project (.NET Core) C#

Console App (.NET Core) C#

Unit Test Project (.NET Framework) C#

C# | All platforms | All project types

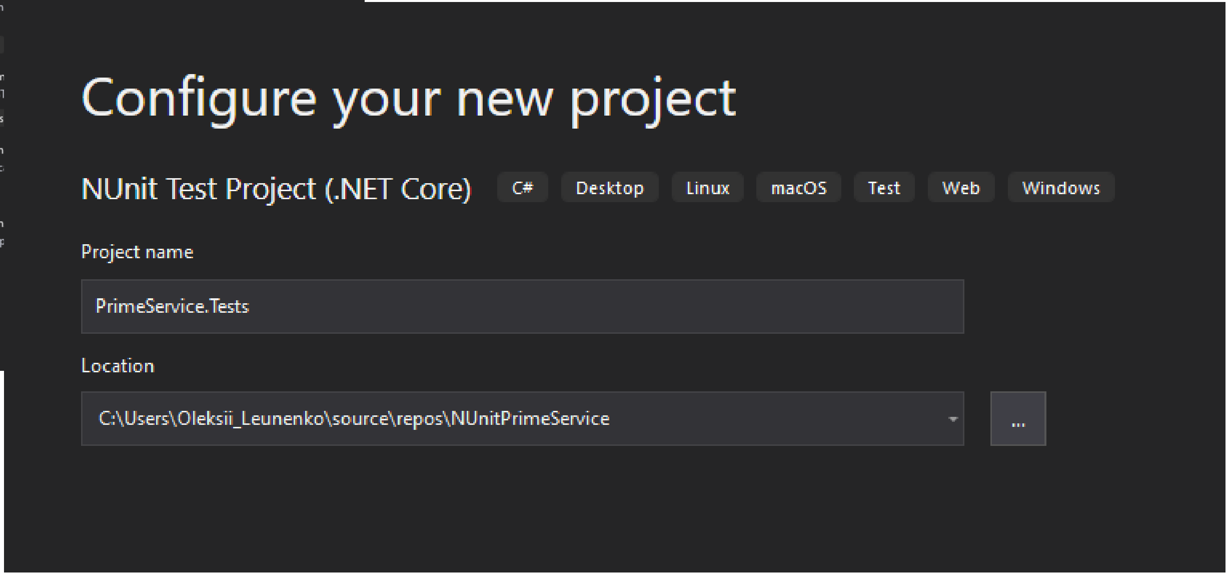
NUnit Test Project (.NET Core)
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Desktop Linux macOS Test Web Windows

xUnit Test Project (.NET Core)
A project that contains xUn
and MacOS.
C# Linux macOS

Unit Test Project (.NET Fram
A project that contains MST
C# Test Windows

Audio Unit Effect Extension
An Audio Unit Effect applic
C# iOS Mobile

Audio Unit Generator Exten
An Audio Unit Generator ap
C# iOS Mobile



Configure your new project

NUnit Test Project (.NET Core) C# Desktop Linux macOS Test Web Windows

Project name

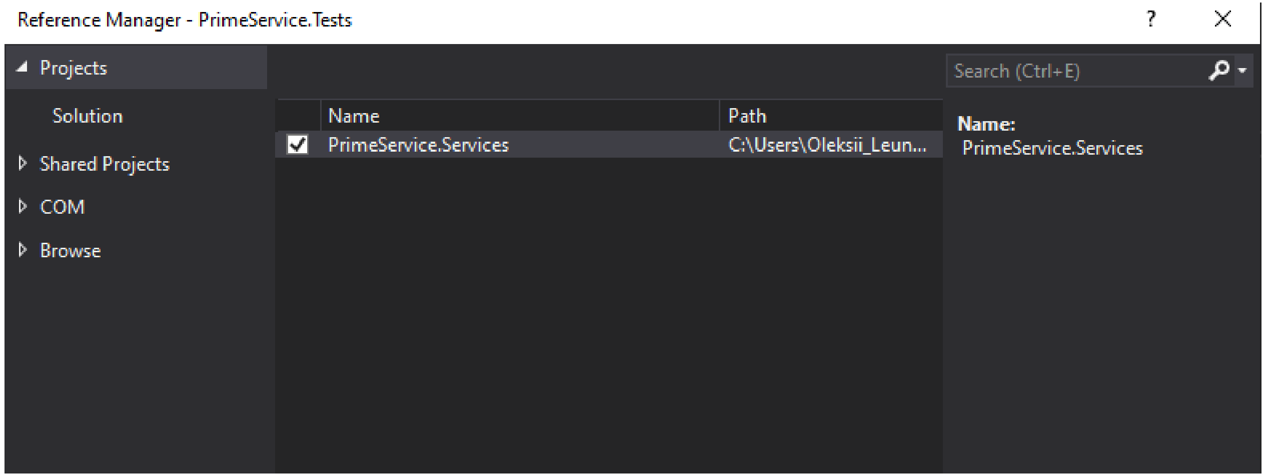
PrimeService.Tests

Location

C:\Users\Oleksii_Leunenko\source\repos\NUnitPrimeService

...

Unit test frameworks and extensions: NUnit



	Name	Path
✓	PrimeService.Services	C:\Users\Oleksii_Leun...

Name: PrimeService.Services

Rename the *UnitTest1.cs* file to *PrimeService_IsPrimeShould.cs*

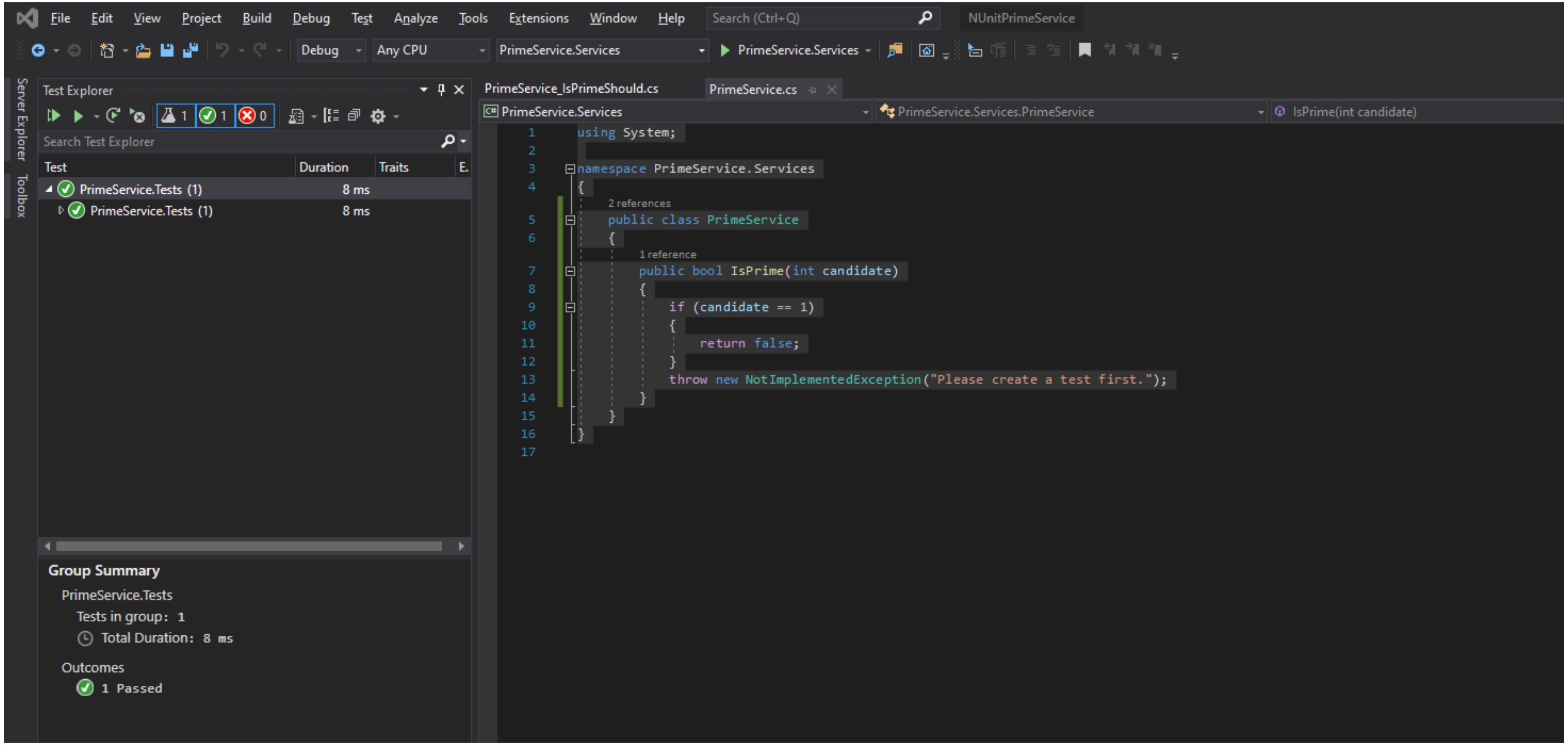
Unit test frameworks and extensions: NUnit

```csharp
using NUnit.Framework;
using PrimeService.Services;


namespace PrimeService.Tests
{
    public class Tests
    {
        [TestFixture]
        public class PrimeService_IsPrimeShould
        {
            private PrimeService.Services.PrimeService _primeService;

            [SetUp]
            public void SetUp()
            {
                _primeService = new PrimeService.Services.PrimeService();
            }

            [Test]
            public void IsPrime_InputIs1_ReturnFalse()
            {
                var result = _primeService.IsPrime(1);

                Assert.IsFalse(result, "1 should not be prime");
            }
        }
    }
}
```

Unit test frameworks and extensions: NUnit

```
using System;

namespace PrimeService.Services
{
    public class PrimeService
    {
        public bool IsPrime(int candidate)
        {
            if (candidate == 1)
            {
                return false;
            }
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

Unit test frameworks and extensions: NUnit



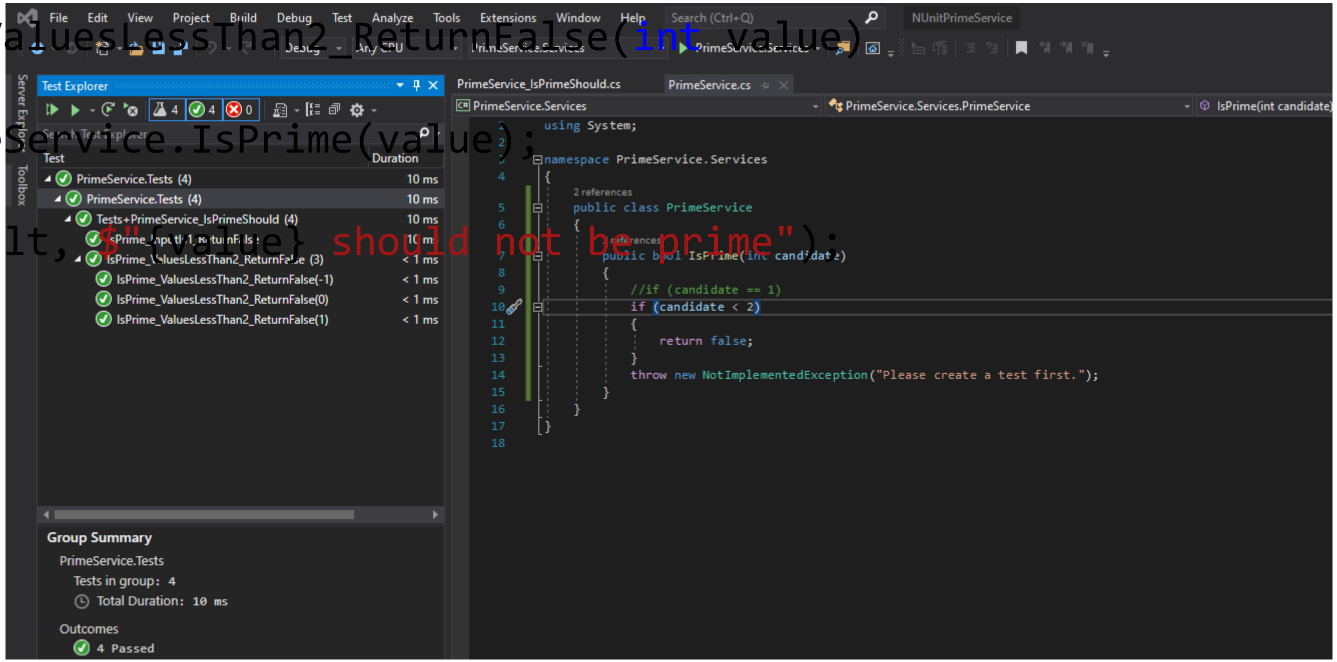
Test	Duration	Traits	E.
PrimeService.Tests (1)	8 ms		
PrimeService.Tests (1)	8 ms		

Group Summary
PrimeService.Tests
Tests in group: 1
Total Duration: 8 ms
Outcomes
1 Passed

```
using System;

namespace PrimeService.Services
{
    2 references
    public class PrimeService
    {
        1 reference
        public bool IsPrime(int candidate)
        {
            if (candidate == 1)
            {
                return false;
            }
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

Unit test frameworks and extensions: NUnit

```
[TestCase(-1)]
[TestCase(0)]
[TestCase(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
  var result = _primeService.IsPrime(value);

  Assert.IsFalse(result, $"{value} should not be prime");
}
```

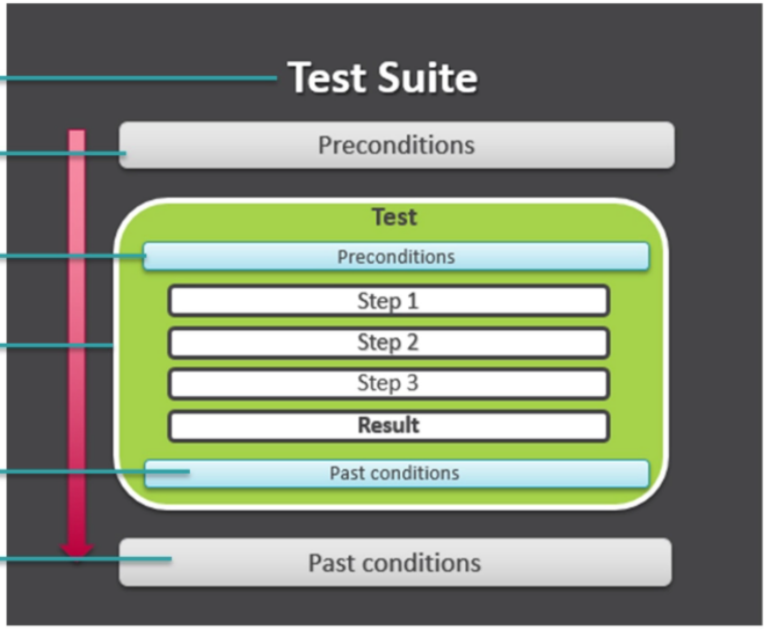


```
Test                                            Duration
PrimeService.Tests (4)                          10 ms
  PrimeService.Tests (4)                        10 ms
    Tests+PrimeService_IsPrimeShould (4)        10 ms
      IsPrime_InputIs1_ReturnFalse              10 ms
      IsPrime_ValuesLessThan2_ReturnFalse (3)   < 1 ms
        IsPrime_ValuesLessThan2_ReturnFalse(-1) < 1 ms
        IsPrime_ValuesLessThan2_ReturnFalse(0)  < 1 ms
        IsPrime_ValuesLessThan2_ReturnFalse(1)  < 1 ms

Group Summary
  PrimeService.Tests
    Tests in group: 4
    Total Duration: 10 ms
  Outcomes
    4 Passed
```

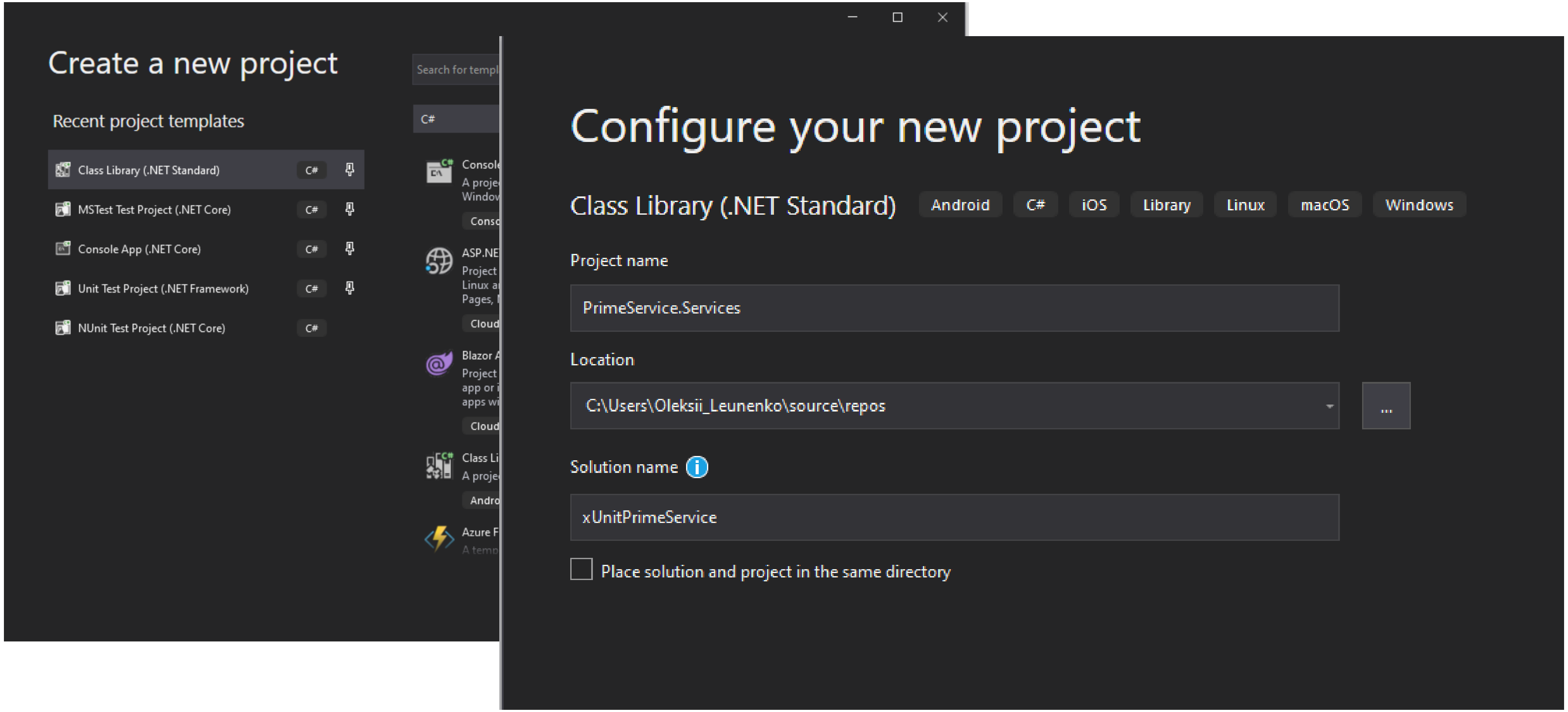
```
using System;

namespace PrimeService.Services
{
    2 references
    public class PrimeService
    {
        references
        public bool IsPrime(int candidate)
        {
            //if (candidate == 1)
            if (candidate < 2)
            {
                return false;
            }
            throw new NotImplementedException("Please create a test first.");
        }
    }
}
```

Unit test frameworks and extensions: NUnit

```
[TestFixture]
0 references
public class NUnitExample
{
    [TestFixtureSetUp]
    0 references
    public static void ClassInit()
    ...

    [SetUp]
    0 references
    public static void TestInit()
    ...

    [Test]
    0 references
    public void SimpleTestExample()
    ...

    [TearDown]
    0 references
    public static void TestClean()
    ...

    [TestFixtureTearDown]
    0 references
    public static void ClassClean()
    ...
}
```



Unit test frameworks and extensions: xUnit



Create a new project

Recent project templates

- Class Library (.NET Standard) C#
- MSTest Test Project (.NET Core) C#
- Console App (.NET Core) C#
- Unit Test Project (.NET Framework) C#
- NUnit Test Project (.NET Core) C#

Configure your new project

Class Library (.NET Standard) Android C# iOS Library Linux macOS Windows

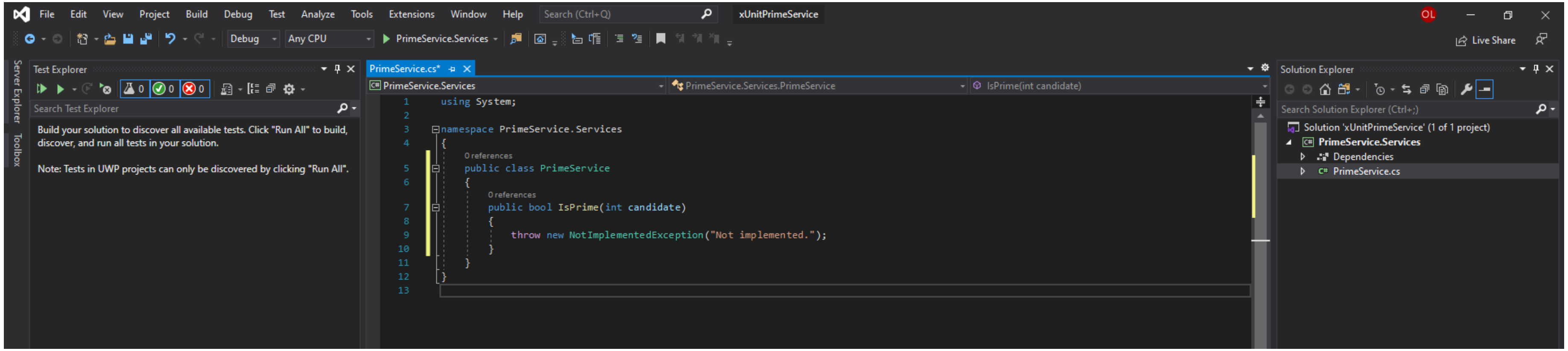
Project name: PrimeService.Services

Location: C:\Users\Oleksii_Leunenko\source\repos

Solution name: xUnitPrimeService

☐ Place solution and project in the same directory

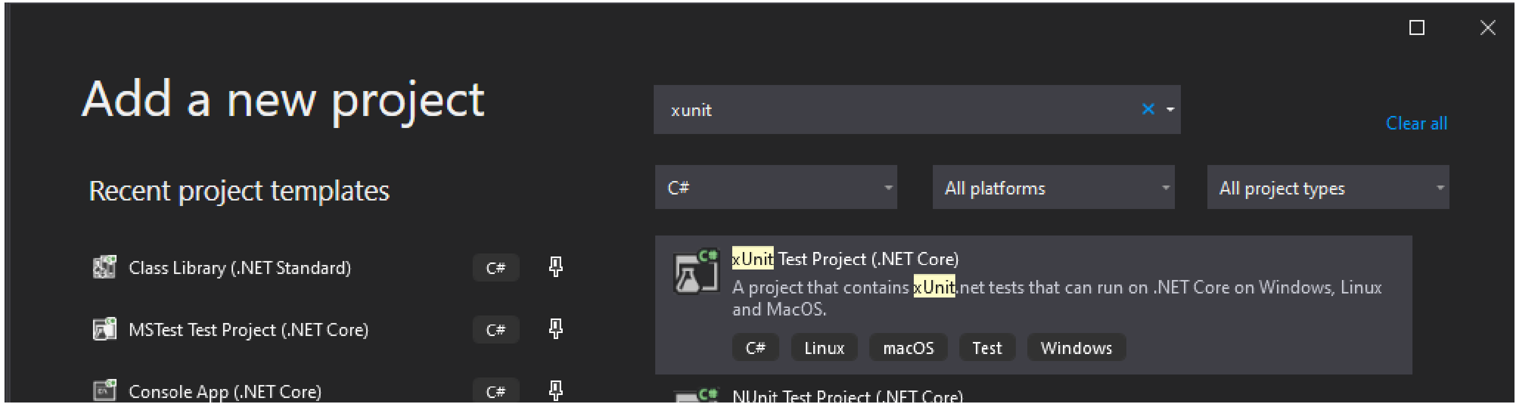
Unit test frameworks and extensions: xUnit



Build your solution to discover all available tests. Click "Run All" to build, discover, and run all tests in your solution.

Note: Tests in UWP projects can only be discovered by clicking "Run All".

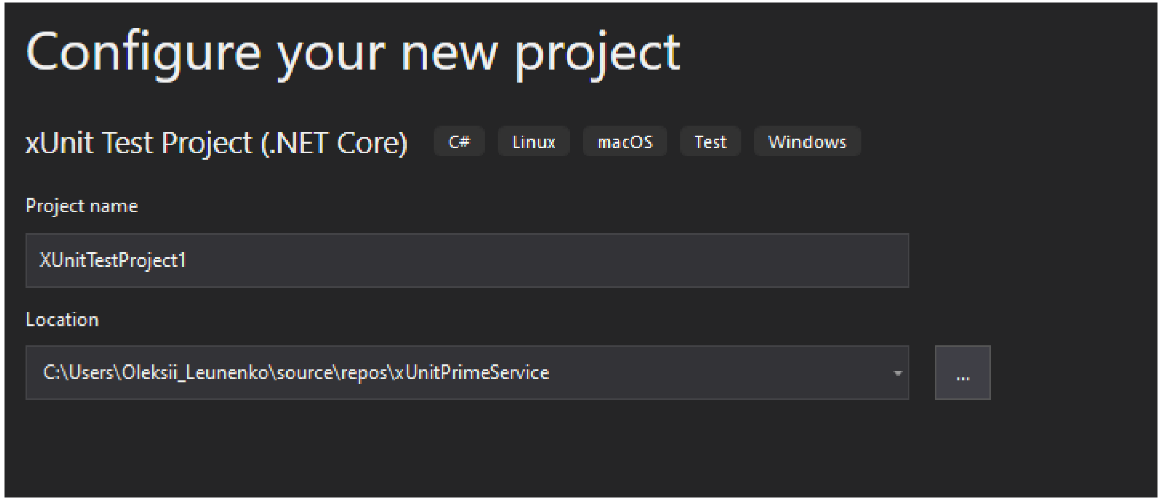
```
using System;

namespace PrimeService.Services
{
    0 references
    public class PrimeService
    {
        0 references
        public bool IsPrime(int candidate)
        {
            throw new NotImplementedException("Not implemented.");
        }
    }
}
```



xUnit Test Project (.NET Core)

A project that contains xUnit.net tests that can run on .NET Core on Windows, Linux and MacOS.

Unit test frameworks and extensions: xUnit



Configure your new project

xUnit Test Project (.NET Core)

Project name

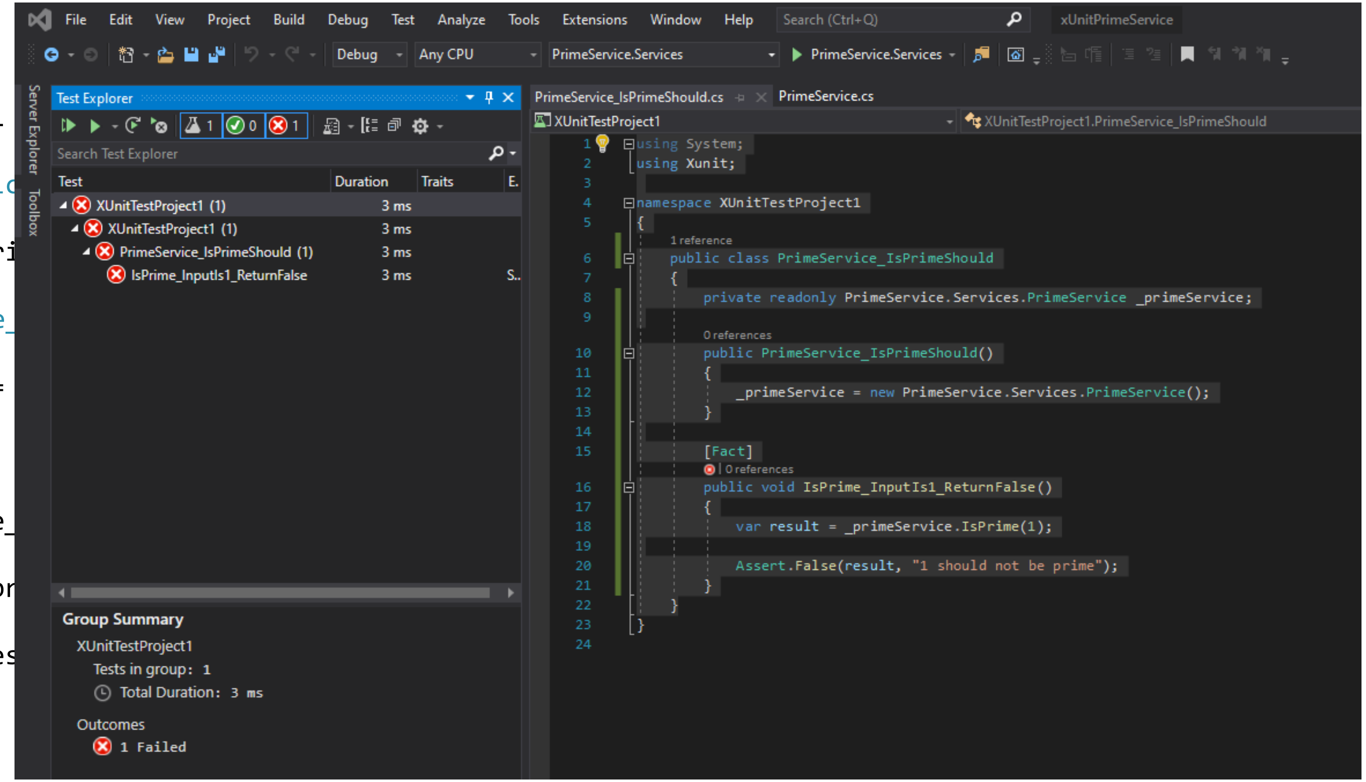
XUnitTestProject1

Location

C:\Users\Oleksii_Leunenko\source\repos\xUnitPrimeService

Unit test frameworks and extensions: xUnit

```
using System;
using Xunit;

namespace XUnitTestProject1
{
    public class PrimeServic
    {
        private readonly Pri

        public PrimeService_
        {
            _primeService =
        }

        [Fact]
        public void IsPrime_
        {
            var result = _pr

            Assert.False(res
        }
    }
```



Test	Duration	Traits	E.
XUnitTestProject1 (1)	3 ms		
XUnitTestProject1 (1)	3 ms		
PrimeService_IsPrimeShould (1)	3 ms		
IsPrime_InputIs1_ReturnFalse	3 ms		S..

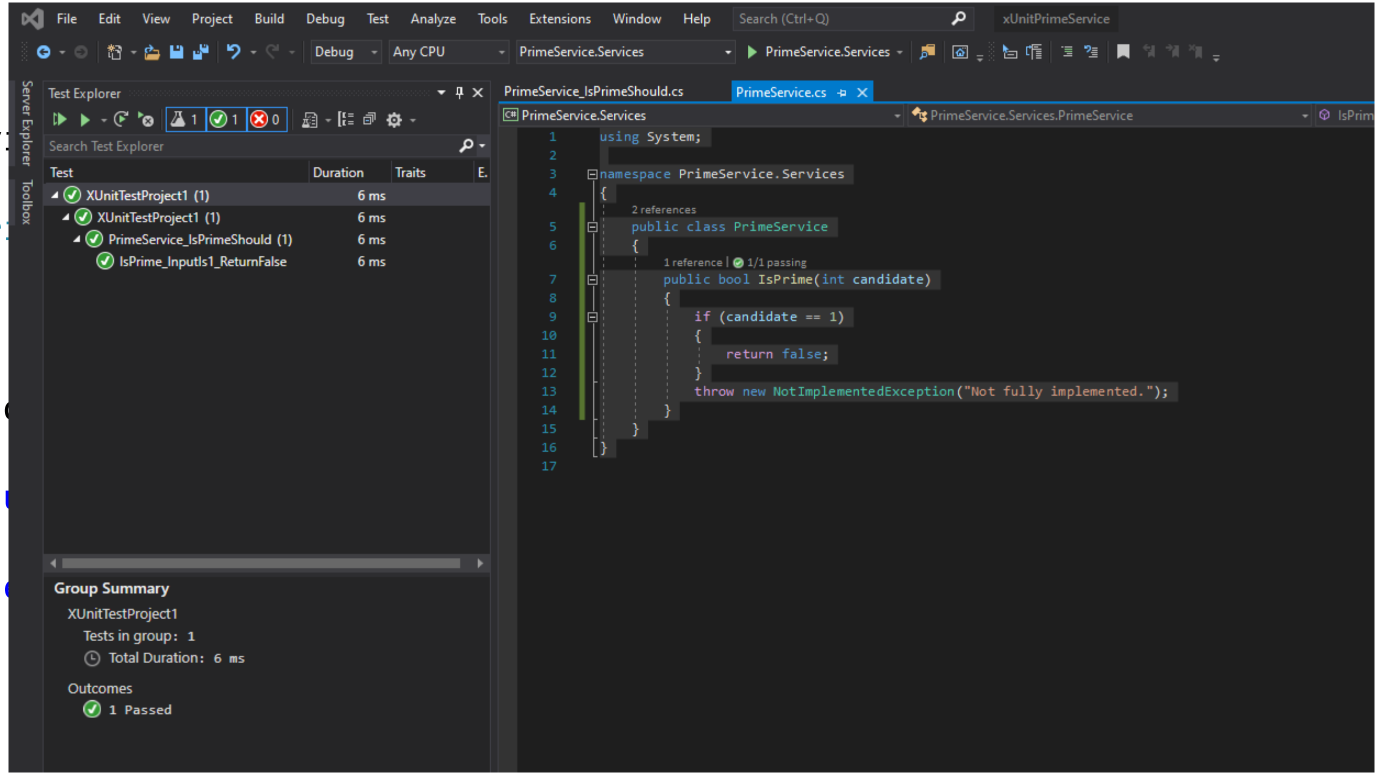
Group Summary
XUnitTestProject1
Tests in group: 1
Total Duration: 3 ms

Outcomes
1 Failed

```
using System;
using Xunit;

namespace XUnitTestProject1
{
    1 reference
    public class PrimeService_IsPrimeShould
    {
        private readonly PrimeService.Services.PrimeService _primeService;

        0 references
        public PrimeService_IsPrimeShould()
        {
            _primeService = new PrimeService.Services.PrimeService();
        }

        [Fact]
        0 references
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var result = _primeService.IsPrime(1);

            Assert.False(result, "1 should not be prime");
        }
    }
}
```

Unit test frameworks and extensions: xUnit

```
using System;

namespace PrimeServ
{
    public class Pr
    {
        public bool
        {
            if (can
            {
                ret
            }
            throw ne
        }
    }
}
```



Test	Duration	Traits
XUnitTestProject1 (1)	6 ms	
XUnitTestProject1 (1)	6 ms	
PrimeService_IsPrimeShould (1)	6 ms	
IsPrime_InputIs1_ReturnFalse	6 ms	

```
Group Summary
  XUnitTestProject1
    Tests in group: 1
    Total Duration: 6 ms
  Outcomes
    1 Passed
```

```
using System;

namespace PrimeService.Services
{
    2 references
    public class PrimeService
    {
        1 reference | 1/1 passing
        public bool IsPrime(int candidate)
        {
            if (candidate == 1)
            {
                return false;
            }
            throw new NotImplementedException("Not fully implemented.");
        }
    }
}
```

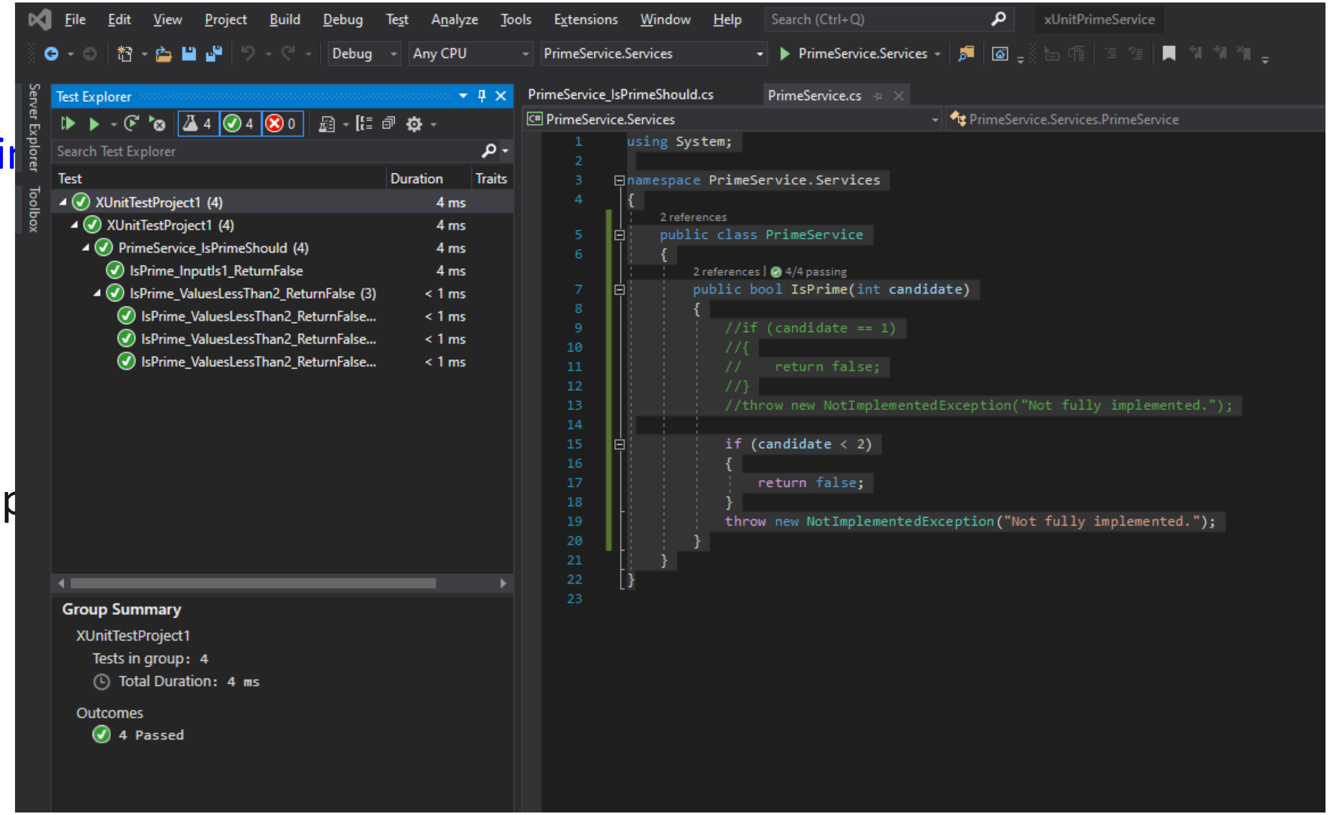
Unit test frameworks and extensions: xUnit

```
[Fact]
public void IsPrime_InputIs1_ReturnFalse()
{
    var result = _primeService.IsPrime(1);
    Assert.False(result, "1 should not be
    prime");
}
```

```
[Theory]
[InlineData(-1)]
[InlineData(0)]
[InlineData(1)]
public void IsPrime_ValuesLessThan2_ReturnFalse(int value)
{
    var result = _primeService.IsPrime(value);
    Assert.False(result, $"{value} should not be prime");
}
```

Unit test frameworks and extensions: xUnit

```
public bool IsPrime(i
{
    if (candidate < 2)
    {
        return false;
    }
    throw new NotImp
}
```



Test Explorer

4 ✓4 ⊗0

Search Test Explorer

Test	Duration	Traits
✓ XUnitTestProject1 (4)	4 ms	
✓ XUnitTestProject1 (4)	4 ms	
✓ PrimeService_IsPrimeShould (4)	4 ms	
✓ IsPrime_InputIs1_ReturnFalse	4 ms	
✓ IsPrime_ValuesLessThan2_ReturnFalse (3)	< 1 ms	
✓ IsPrime_ValuesLessThan2_ReturnFalse...	< 1 ms	
✓ IsPrime_ValuesLessThan2_ReturnFalse...	< 1 ms	
✓ IsPrime_ValuesLessThan2_ReturnFalse...	< 1 ms	

Group Summary
XUnitTestProject1
Tests in group: 4
Total Duration: 4 ms

Outcomes
✓ 4 Passed

PrimeService_IsPrimeShould.cs | PrimeService.cs

PrimeService.Services | PrimeService.Services.PrimeService

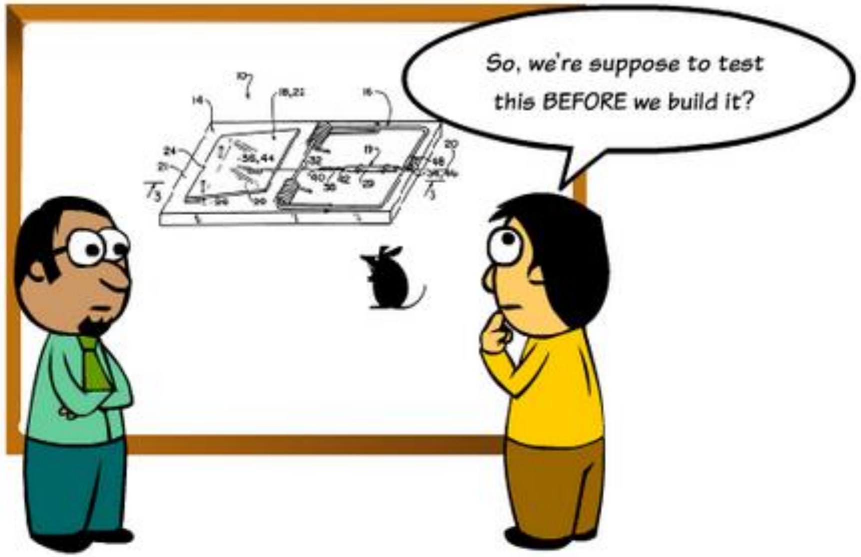
```
using System;

namespace PrimeService.Services
{
    2 references
    public class PrimeService
    {
        2 references | ✓ 4/4 passing
        public bool IsPrime(int candidate)
        {
            //if (candidate == 1)
            //{
            //    return false;
            //}
            //throw new NotImplementedException("Not fully implemented.");

            if (candidate < 2)
            {
                return false;
            }
            throw new NotImplementedException("Not fully implemented.");
        }
    }
}
```

Unit test frameworks and extensions

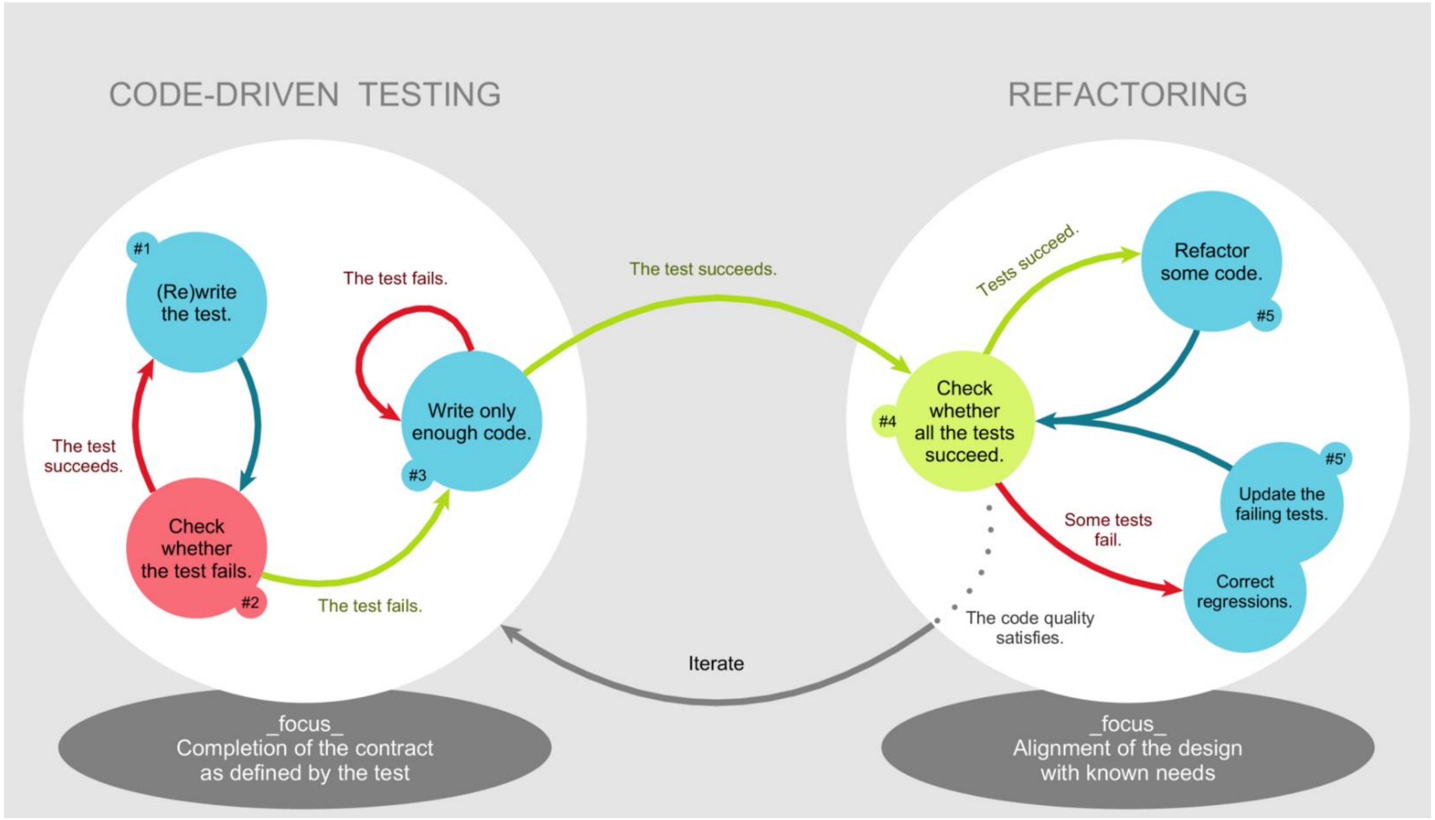
Test-Driven Development (TDD)

TDD

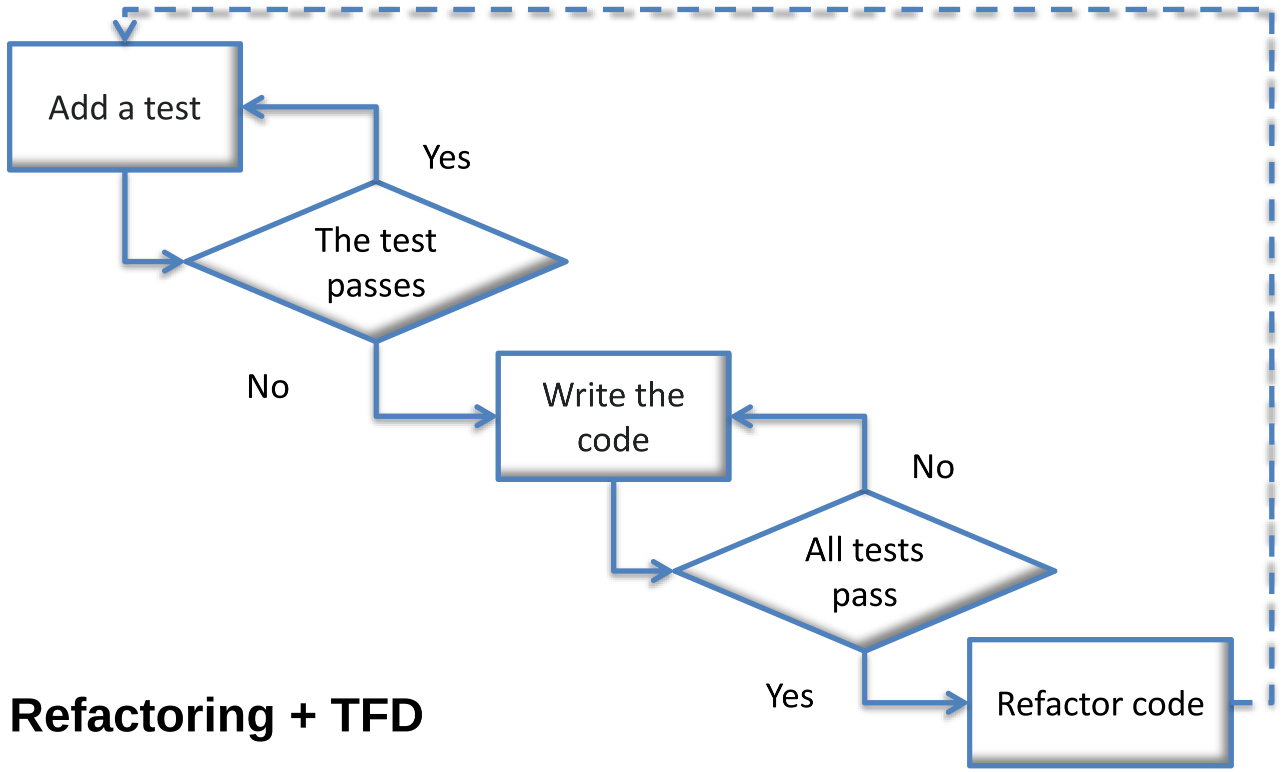
Test-driven development (**TDD**) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the code is improved so that the tests pass. This is opposed to software development that allows code to be added that is not proven to meet requirements.



TDD


CODE-DRIVEN TESTING
REFACTORING
#1
(Re)write the test.
The test fails.
The test succeeds.
Write only enough code.
#3
The test succeeds.
Check whether the test fails.
#2
The test fails.
Tests succeed.
Refactor some code.
#5
#4
Check whether all the tests succeed.
#5'
Update the failing tests.
Some tests fail.
Correct regressions.
The code quality satisfies.
Iterate
focus
Completion of the contract as defined by the test
focus
Alignment of the design with known needs

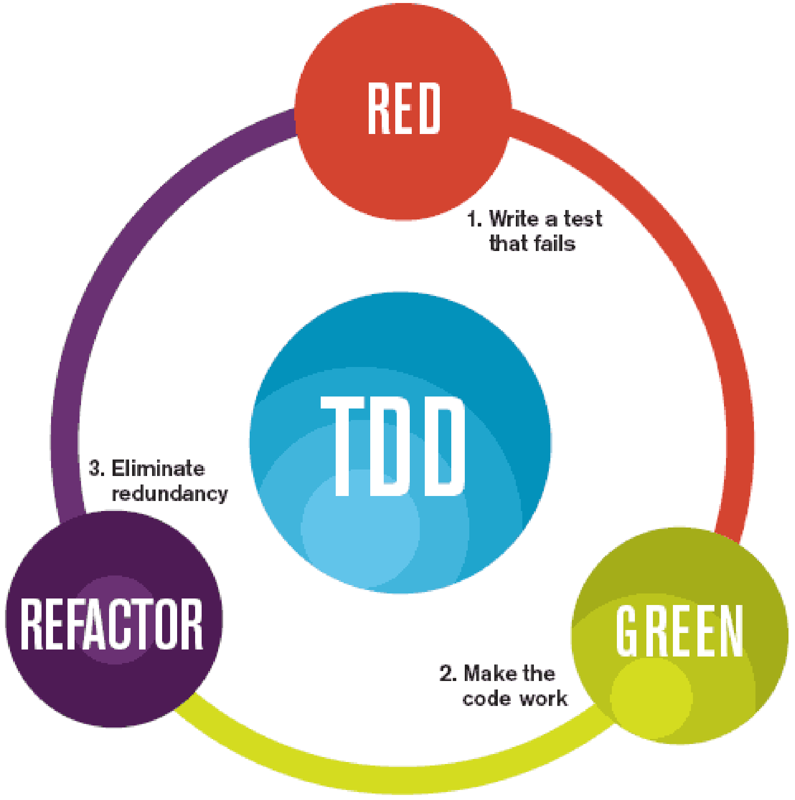
TDD: The Steps of test-first development (TFD)



TDD = Refactoring + TFD

TDD: Development style

- "keep it simple, stupid" (KISS)
- "You aren't gonna need it" (YAGNI)
- "Fake it till you make it"



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

TDD: Fakes, mocks and integration tests

Unit tests are so named because they each test *one unit* of code. A complex module may have a thousand unit tests and a simple module may have only ten. The unit tests used for TDD should never cross process boundaries in a program, let alone network connections. Doing so introduces delays that make tests run slowly and discourage developers from running the whole suite. Introducing dependencies on external modules or data also turns *unit tests* into ***integration tests***. If one module misbehaves in a chain of interrelated modules, it is not so immediately clear where to look for the cause of the failure.

TDD: Fakes, mocks and integration tests

When code under development relies on a database, a web service, or any other external process or service, enforcing a unit-testable separation is also an opportunity and a driving force to design more modular, more testable and more reusable code.

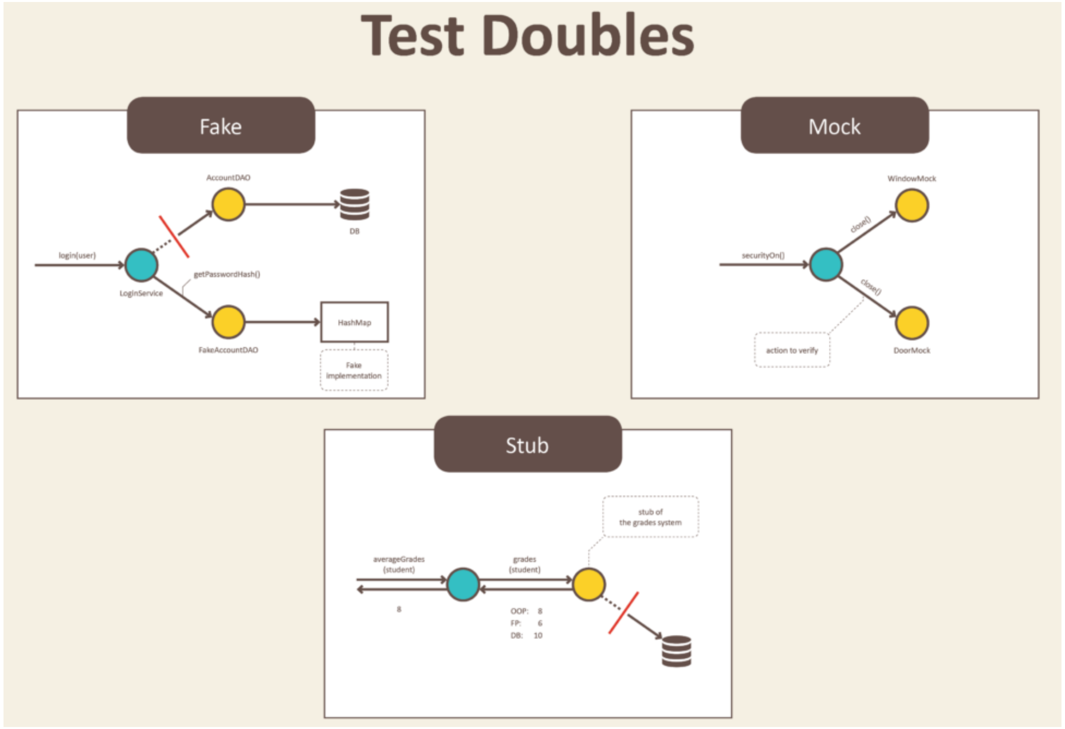
Two steps are necessary:

1. Whenever external access is needed in the final design, an interface should be defined that describes the access available. See the dependency inversion principle for a discussion of the benefits of doing this regardless of TDD.
2. The interface should be implemented in two ways, one of which really accesses the external process, and the other of which is a fake or mock. Fake objects need do little more than add a message such as "Person object saved" to a trace log, against which a test assertion can be run to verify correct behaviour. Mock objects differ in that they themselves contain test assertions that can make the test fail, for example, if the person's name and other data are not as expected.

Stub, Mock, Fake

Test Double

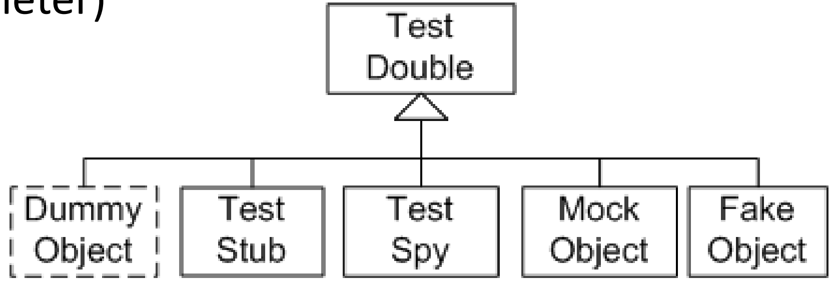
A Test Double is a test-specific capability that substitutes for a system capability, typically a class or function, that the UUT (unit under test) depends on.



http://xunitpatterns.com/Test%20Double.html

Test Double

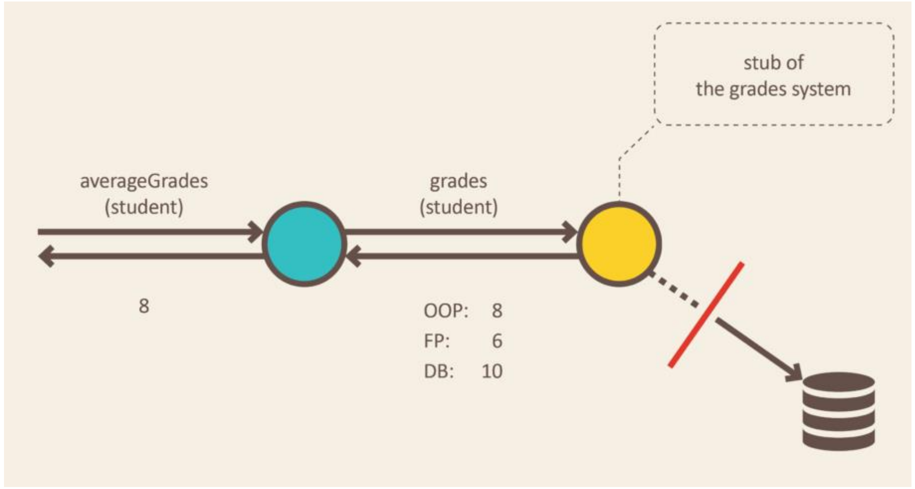
- **Test stub** (used for providing the tested code with "indirect input")
- **Mock object** (used for verifying "indirect output" of the tested code, by first defining the expectations before the tested code is executed)
- **Test spy** (used for verifying "indirect output" of the tested code, by asserting the expectations afterwards, without having defined the expectations before the tested code is executed. It helps in recording information about the indirect object created)
- **Fake object** (used as a simpler implementation, e.g. using an in-memory database in the tests instead of doing real database access)
- **Dummy object** (used when a parameter is needed for the tested method but without actually needing to use the parameter)



Stub

Stub is an object that holds predefined data and uses it to answer calls during tests. It is used when we cannot or don't want to involve objects that would answer with real data or have undesirable side effects.

- A Stub is the lightest and most static version of this chain.
- Stub always returns the predefined output regardless of the input.
- We can't control the behavior of the stub.
- A stub can be useful to mimic the database objects.



Stub

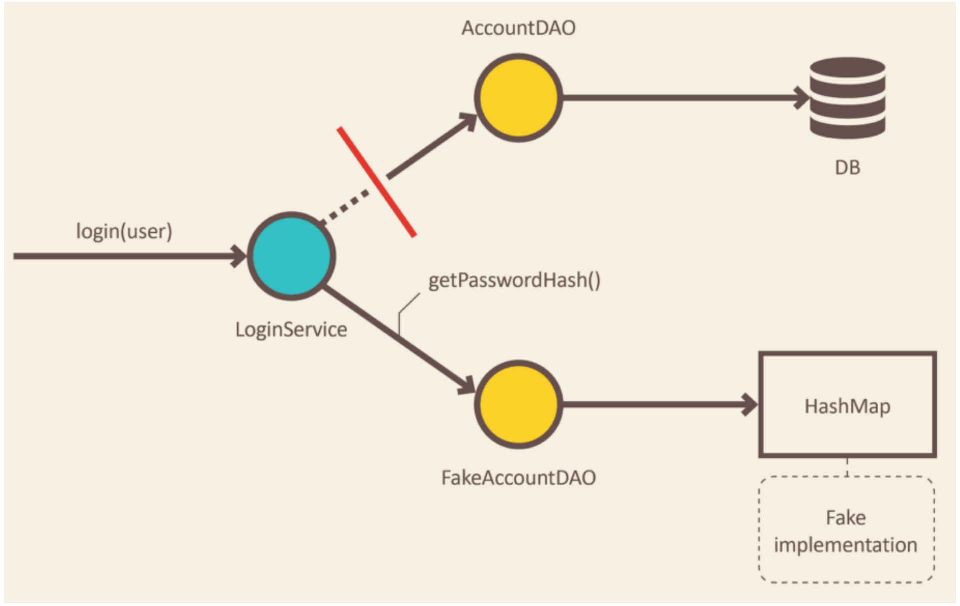
```
public class StubUserStore : IUserStore
{
   public string GetUserRole(string username)
   {
      return "contributor";
   }

   public List<UserDetail> GetAllUsers()
   {
      return new List<UserDetail>()
      {
         new UserDetail{ Role = "administrator", Name = "admin"},

         new UserDetail(){ Role = "contributor", Name = "User 1"}
      };
   }
}
```

```
public interface IUserStore
{
   string GetUserRole(string username);
}

public class UserDetail
{
   public string Name { get; set; }
   public string Role { get; set; }
}
```

Fake

Fakes are objects that have working implementations, but not same as production one. Usually they take some shortcut and have simplified version of production code.

- A Fake is more powerful than Stub.
- Fake classes can change the behavior based on input.
- Fake class functions can return different output for different inputs unlike that of stub.
- Fakes can help us to mimic all the possible behavior of the interfaces.



Fake

```
public class FakeUserStore : IUserStore
{
    public string GetUserRole(string username)
    {
        if (username == "admin")
            return "administrator";
        else
        return "contributor";
    }
}
public interface IUserStore
{
    string GetUserRole(string username);
}
```

Spy

- A Spy is an advanced version of the Fake which can store the previous state of the object.
- The spy can be useful to mimic the retry services or to check scenarios like 'if the function called at least once'.
- You can also create a spy for loggers to store and validate all the logs logged while running the test case.

Spy

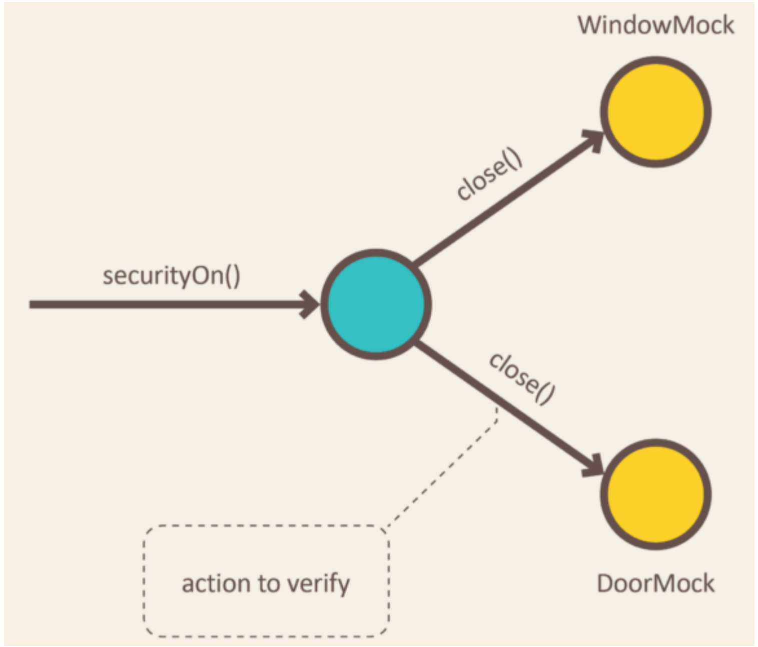
```
public class SpyUserStore : IUserStore
{
   private static int Counter { get; set; }

   public SpyUserStore()
   {
      Counter = 0;
   }

   public string GetUserRole(string username)
   {

      if (Counter >= 1)
         throw new Exception("Function called more than once");

Counter++;

   if (username == "admin")
         return "administrator";
      else
         return "contributor";
    }
}
```

Mock

Mocks are objects that register calls they receive.

In test assertion we can verify on Mocks that all expected actions were performed.

- A Mock is the most powerful and flexible version in the chain.
- The behavior of the mocked interface can be changed dynamically based on scenarios.
- We can apply a variety of assertions by creating Mocked objects using mock frameworks, for example - **Moq**.
- Mock gives the full control over the behavior of mocked objects.



Mock

```
Mock<IUserStore> mockedUserStore=new Mock<IUserStore>();
                    mockedUserStore.Setup(func => func.GetUserRole("admin")).Returns("administrator");

                    mockedUserStore.Setup(func => func.GetUserRole("user1")).Returns("contributor");
                    mockedUserStore.Setup(func => func.GetUserRole("user2")).Returns("basic");
```

Stub, Mock, Fake: Which one shall I use?

- Try to avoid *mocks* if the same scenarios can be reproduced with simple *stubs* and *fakes*.
- Use *Stub* to represent database objects and use *Fake* and *Spy* to mimic the behavior of business interfaces or services like retry, logging, etc.
- *Mocks* sometimes make test cases difficult to read and difficult to understand.
- Improper use of *Mock* may impact test strategy in a negative way

.NET Online UA Training Course Feedback

I hope that you will find this material useful.

If you find errors or inaccuracies in this material or know how to improve it, please report on to the electronic address:

Oleksii_Leunenko@epam.com

With the note [.NET Online UA Training Course Feedback]

Thank you.

Q&A

DRIVEN CANDID CREATIVE ORIGINAL INTELLIGENT EXPERT

UA .NET Online LAB