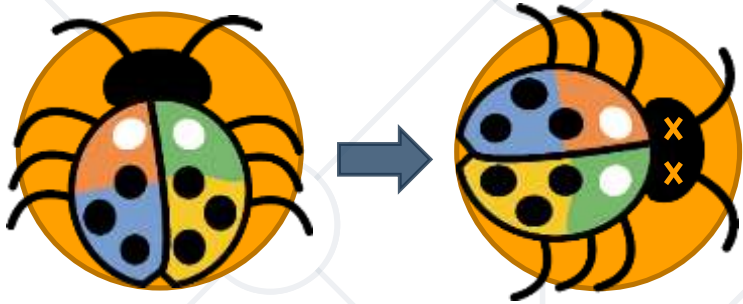


Unit Testing

Building Rock-Solid Software



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

- Seven Testing Principles
- What is Unit Testing?
- Unit Testing Frameworks
- NUnit
 - NUnit Setup
 - Test Classes and Test Methods
 - 3A-s Pattern
- Good Practices

sli.do

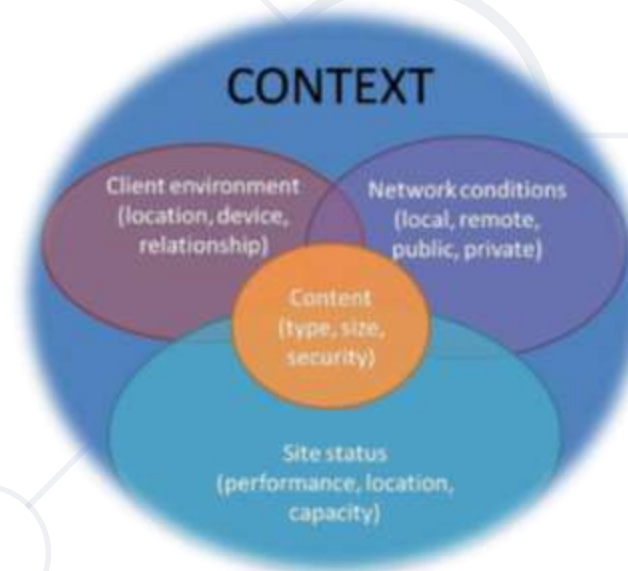
#csharp-advanced



Seven Testing Principles

Seven Testing Principles (1)

- Testing is context dependent
 - Testing is done differently in **different contexts**
- Example:
 - Safety-critical software is tested **differently** from an e-commerce site



Seven Testing Principles (2)

- Exhaustive testing is **impossible**
 - All combinations of inputs and preconditions are usually almost **infinite number**
 - Testing everything is not feasible
 - Except for trivial cases
 - Risk analysis and priorities should be used to focus testing efforts
- E.g.: Big list of naughty strings

A QA Tester walks into a bar:

He orders a beer.

He orders 3 beers.

He orders 2976412836 beers.

He orders 0 beers.

He orders -1 beer.

He orders q beers.

He orders nothing.

Él ordena una cerveza.

He orders a deer.

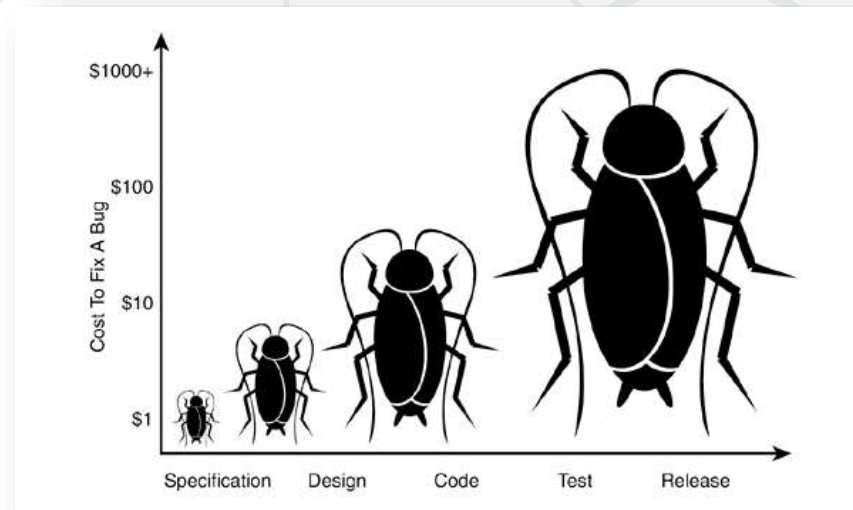
He tries to leave without paying.

He starts ordering a beer, then throws himself through the window half way through.

He orders a beer, gets his receipt, then tries to go back.

Seven Testing Principles (3)

- Early testing is **always preferred**
 - Testing activities shall be started as early as possible
 - And shall be focused on defined objectives
 - The later a bug is found – the more it costs!



Seven Testing Principles (4)

- Defect clustering
 - Testing effort shall be focused **proportionally**
 - To the expected and later observed defect density of modules
 - A **small number** of modules usually contains **most of the defects** discovered (80/20 principle)
 - Responsible for most of the operational failures

Seven Testing Principles (5)

- Pesticide paradox
 - Same tests repeated **over and over again** tend to **lose their effectiveness**
 - Previously **undetected** defects remain **undiscovered**
 - New and modified test cases should be developed

Seven Testing Principles (6)

- Testing shows presence of defects
 - Testing can **show that defects are present**
 - Cannot prove that there are no defects
 - Appropriate testing **reduces** the probability for defects

Seven Testing Principles (7)

- Absence-of-errors fallacy
 - **Finding** and **fixing** defects itself does not help in these cases:
 - The system built is unusable
 - Does not fulfill the users' needs and expectations



Software Used to Test Software

What is Unit Testing?

- **Unit test** == a piece of code that **tests specific functionality** in certain software component (unit)

```
sum(arr)
✓ sum([1,2]) == 3
✓ sum([-2]) == -2
1) sum([]) == 0
2 passing (10ms)
1 failing
```

```
int Sum(int[] arr)
{
    int sum = arr[0];
    for (int i=1; i<arr
        .Length; i++)
        sum += arr[i];
    return sum;
}
```

```
void Test_SumTwoNumbers() {
    if (Sum(new int[]{1, 2}) != 3)
        throw new Exception("1+2 != 3");
}
```

```
void Test_SumEmptyArray() {
    if (Sum(new int[]{ }) != 0)
        throw new Exception("sum [] != 0");
}
```

- **Unit tests**

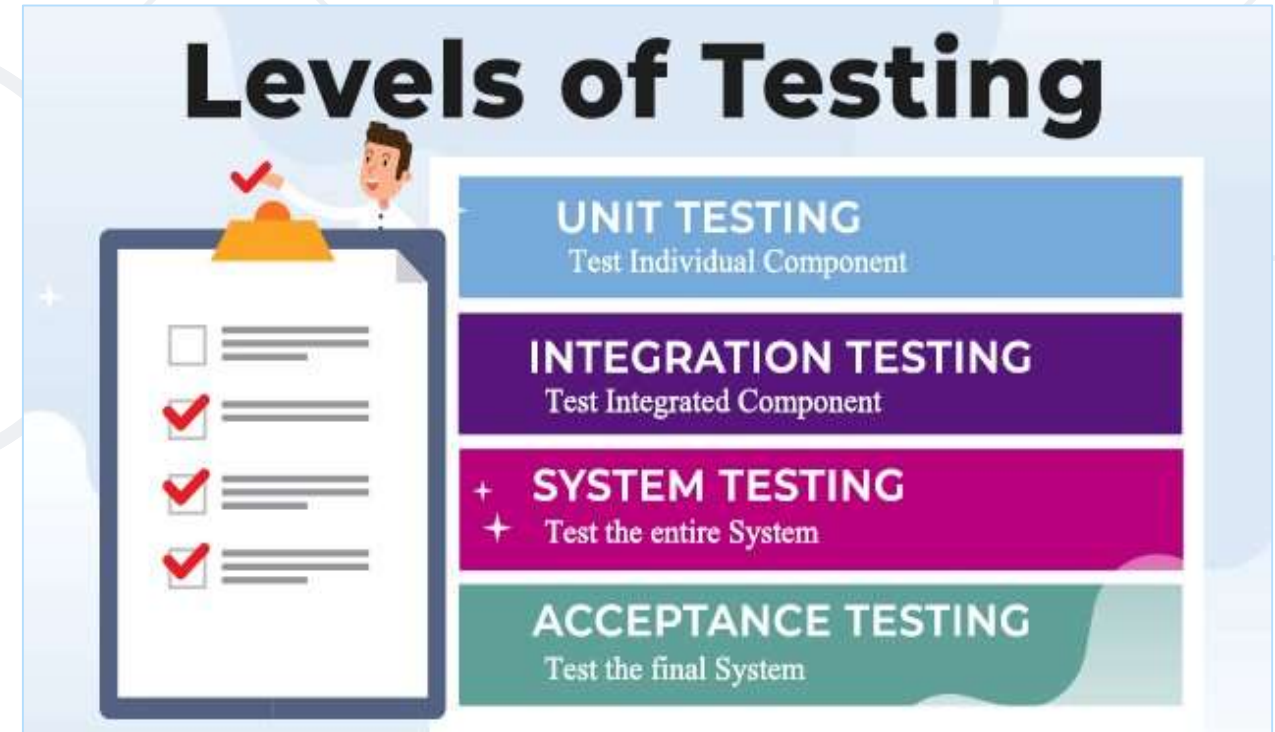
- Test a **single component** (mocking the dependencies)
- NUnit, JUnit, PyUnit, Mocha

- **Integration tests**

- Test an **interaction** between components, e. g. **API tests**

- **System tests / acceptance tests / end-to-end tests**

- Test the **entire system**, e. g. Selenium, Appium, Cypress, Playwright





Unit Testing Frameworks

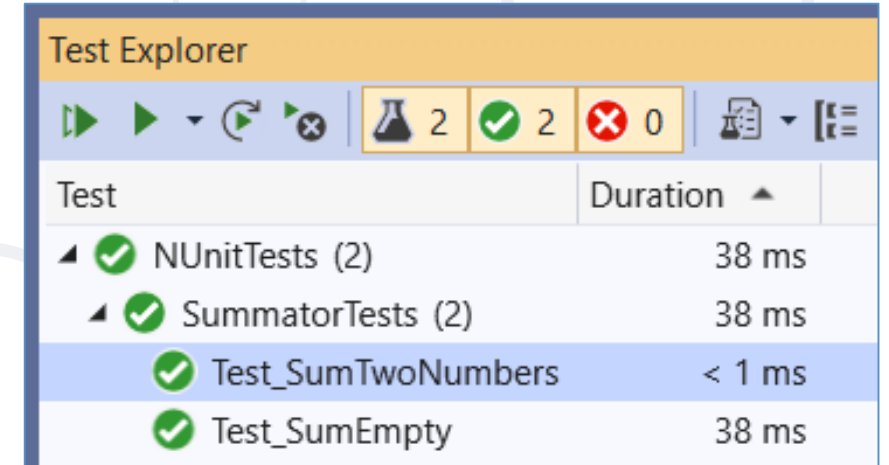
- **Testing frameworks** provide foundation for test automation
 - Consists of **libraries**, code **modules** and **tools** for test automation
 - **Structure the tests** into hierarchical or other form
 - **Implement** test cases, **execute the tests** and **generate reports**
 - **Assert** the execution results and exit conditions
 - Perform initialization at **startup** and cleanup at **shut down**
- **Examples** of testing frameworks:
 - NUnit, xUnit, MSTest (C#), JUnit (Java), Mocha (JS), PyUnit (Python)

Testing Framework – Example

- **Testing frameworks** simplify automated testing and reporting
 - Example: **NUnit** testing framework for C#

```
using NUnit.Framework;

public class SummatorTests
{
    [Test]
    public void Test_SumTwoNumbers() {
        var sum = Sum(new int[] { 1, 2 });
        Assert.AreEqual(3, sum);
    }
}
```



| ▶ ▶ ◀ ↺ ✖ 2 2 0 📄 ⌵ ⌵ | |
|---------------------------|------------|
| Test | Duration ▲ |
| ▶ ✓ NUnitTests (2) | 38 ms |
| ▶ ✓ SummatorTests (2) | 38 ms |
| ✓ Test_SumTwoNumbers | < 1 ms |
| ✓ Test_SumEmpty | 38 ms |

- **Unit testing framework** == automated testing framework == testing framework == test framework
 - Many names for similar concepts → why?
- Testing frameworks like **JUnit** and **NUnit** were initially designed for **unit testing**, but nothing limits them to wider use
- With additional libraries, NUnit and JUnit are used for:
 - **Integration testing, API testing, Web service testing**
 - **End-to-end testing, Web UI testing, mobile testing, etc.**



NUnit: First Steps

Setup and First Test

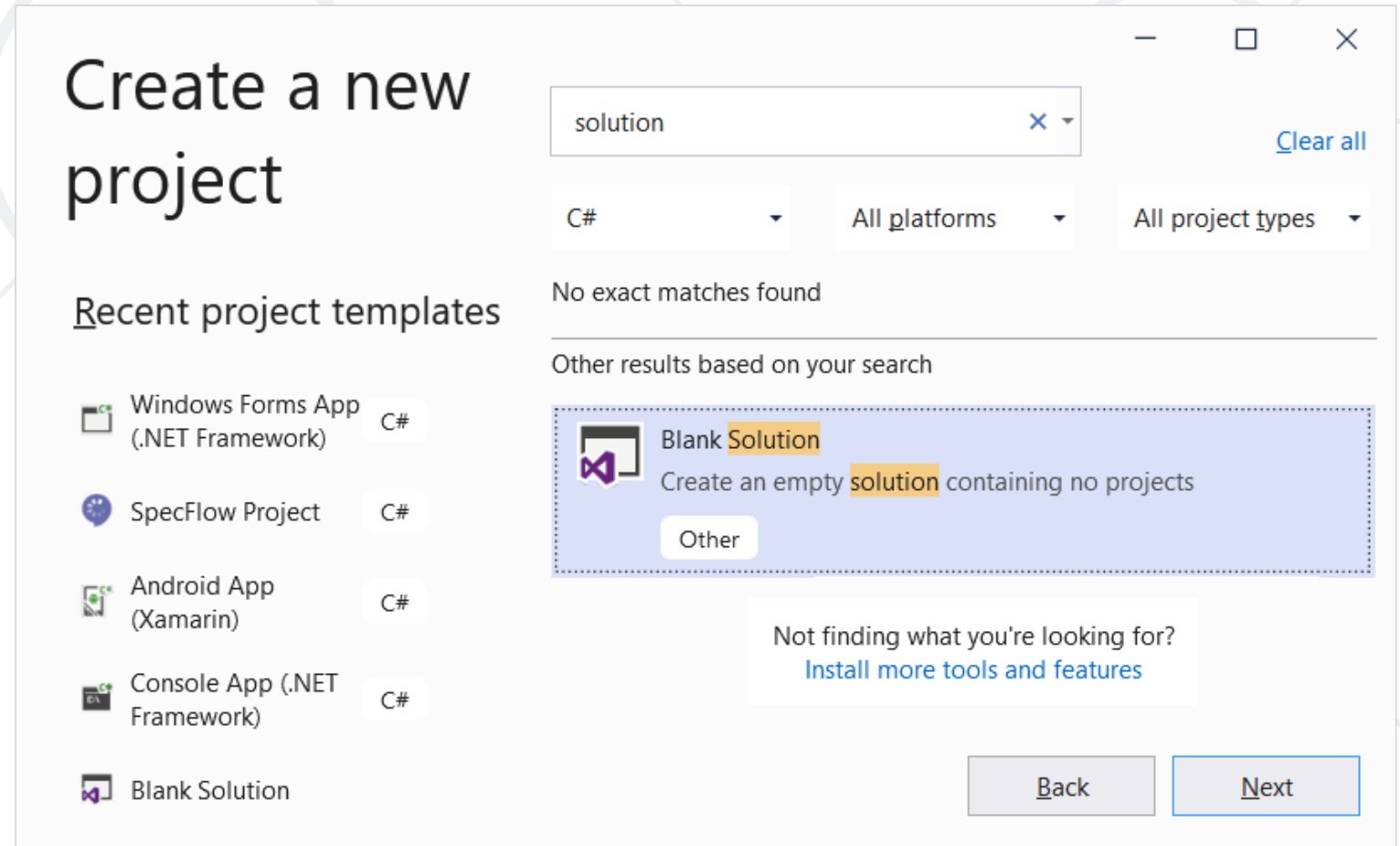
NUnit: Overview

- **NUnit** == popular C# testing framework
 - Supports test suites, test cases, before & after code, startup & cleanup code, timeouts, expected errors, ...
 - Like **JUnit** (for Java)
 - Free, open-source
 - Powerful and mature
 - Wide community
 - Built-in support in Visual Studio
 - Official site: nunit.org



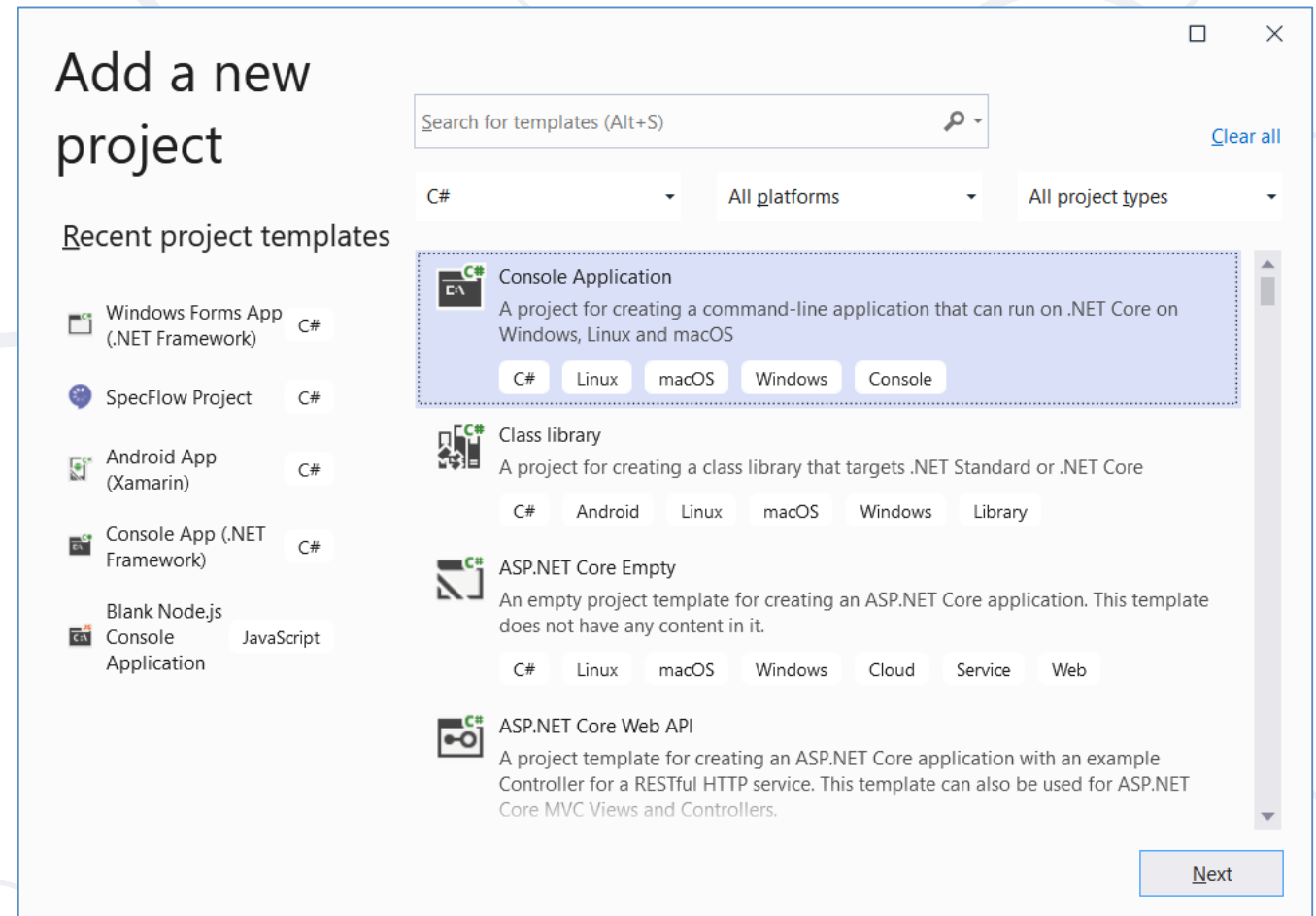
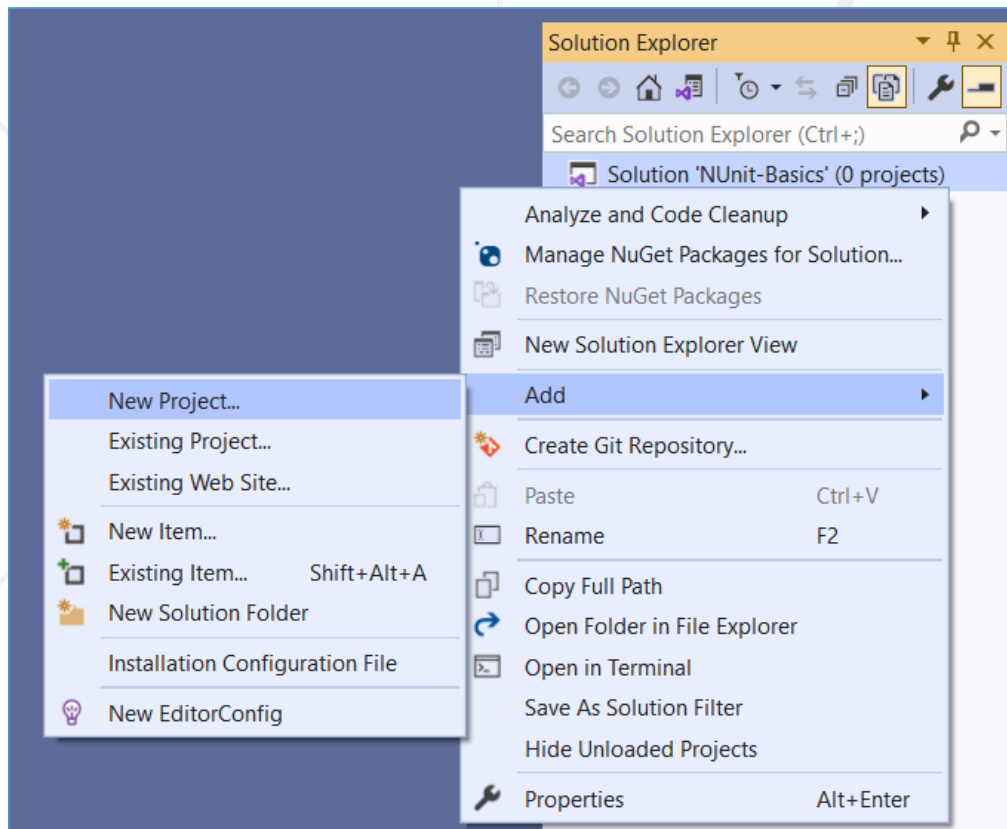
Creating a Blank Solution

- Create a **blank solution** in Visual Studio
 - It will hold the **project for testing**
 - And the **unit test project (tests)**

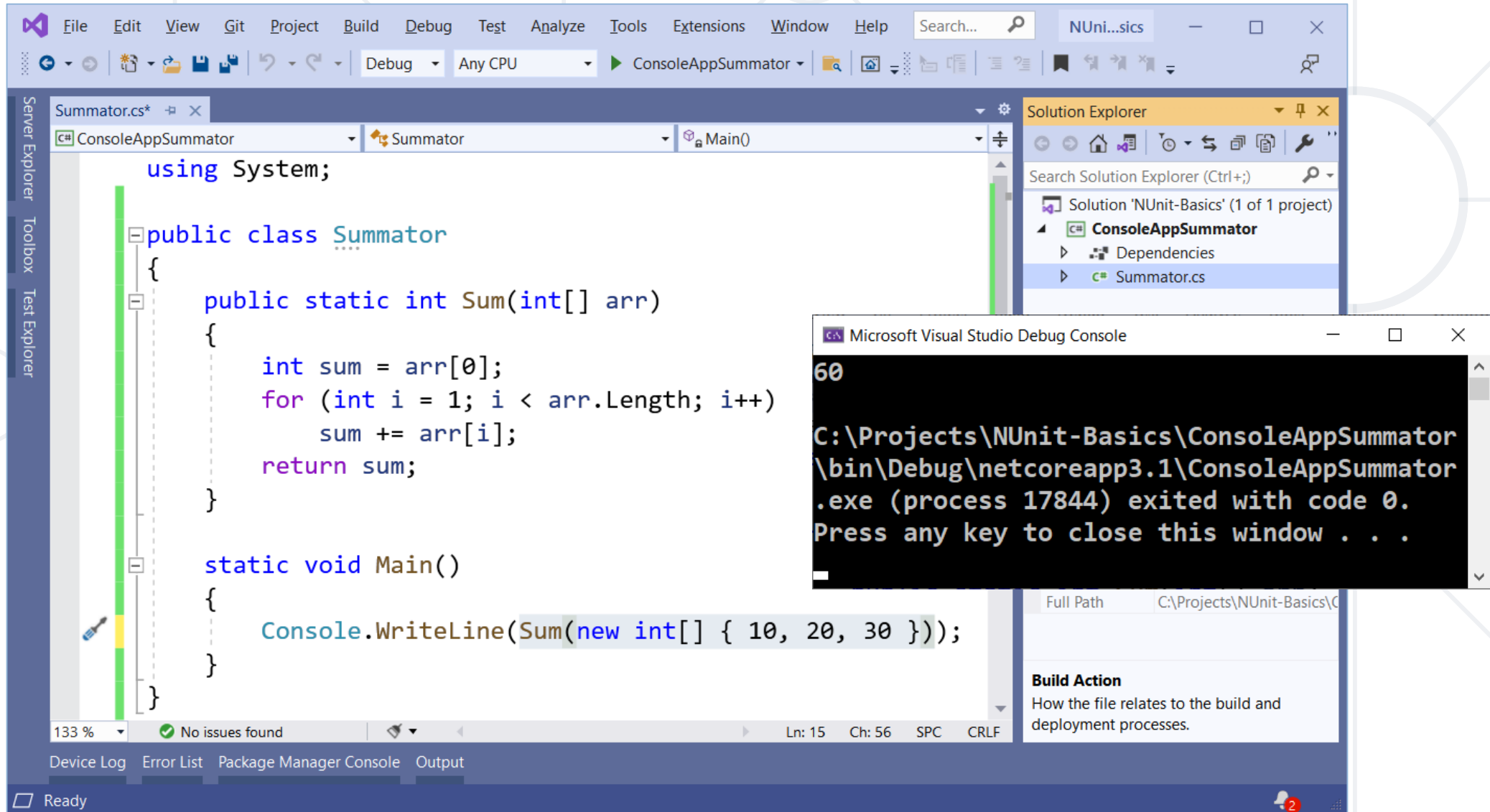


Creating a Project for Testing

- Create a **console-based app**, to hold the **code for testing**



Creating a Project for Testing (2)



Summator.cs*

```
using System;

public class Summator
{
    public static int Sum(int[] arr)
    {
        int sum = arr[0];
        for (int i = 1; i < arr.Length; i++)
            sum += arr[i];
        return sum;
    }

    static void Main()
    {
        Console.WriteLine(Sum(new int[] { 10, 20, 30 }));
    }
}
```

Solution Explorer

- Solution 'NUnit-Basics' (1 of 1 project)
 - ConsoleAppSummator
 - Dependencies
 - Summator.cs

Microsoft Visual Studio Debug Console

```
60

C:\Projects\NUnit-Basics\ConsoleAppSummator\bin\Debug\netcoreapp3.1\ConsoleAppSummator.exe (process 17844) exited with code 0.
Press any key to close this window . . .
```

Full Path C:\Projects\NUnit-Basics\...

Build Action
How the file relates to the build and deployment processes.

133 % No issues found Ln: 15 Ch: 56 SPC CRLF

Device Log Error List Package Manager Console Output

Ready

Creating an NUnit Project

Create a new project

Recent project templates

- ASP.NET Core Web Application C#
- Console App (.NET Core) C#
- Windows Forms App (.NET) C#
- NUnit Test Project (.NET Core) C#
- Console App (.NET Framework) C#
- Windows Forms App (.NET Framework) C#

NUnit

[Clear all](#)

C#

All platforms

All project types



xUnit Test Project (.NET Core)

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

C#

Windows

Linux

macOS

Test



NUnit Test Project (.NET Core)

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

C#

Linux

macOS

Windows

Desktop

Test

Web



xUnit Test Project (.NET Core)

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

C#

Windows

Linux

macOS

Test

[Next](#)

New Project...

Existing Project...

Existing Web Site...

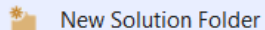


New Item...



Existing Item...

Shift+Alt+A



New Solution Folder

Installation Configuration File



New EditorConfig

Save As Solution Filter

Hide Unloaded Projects

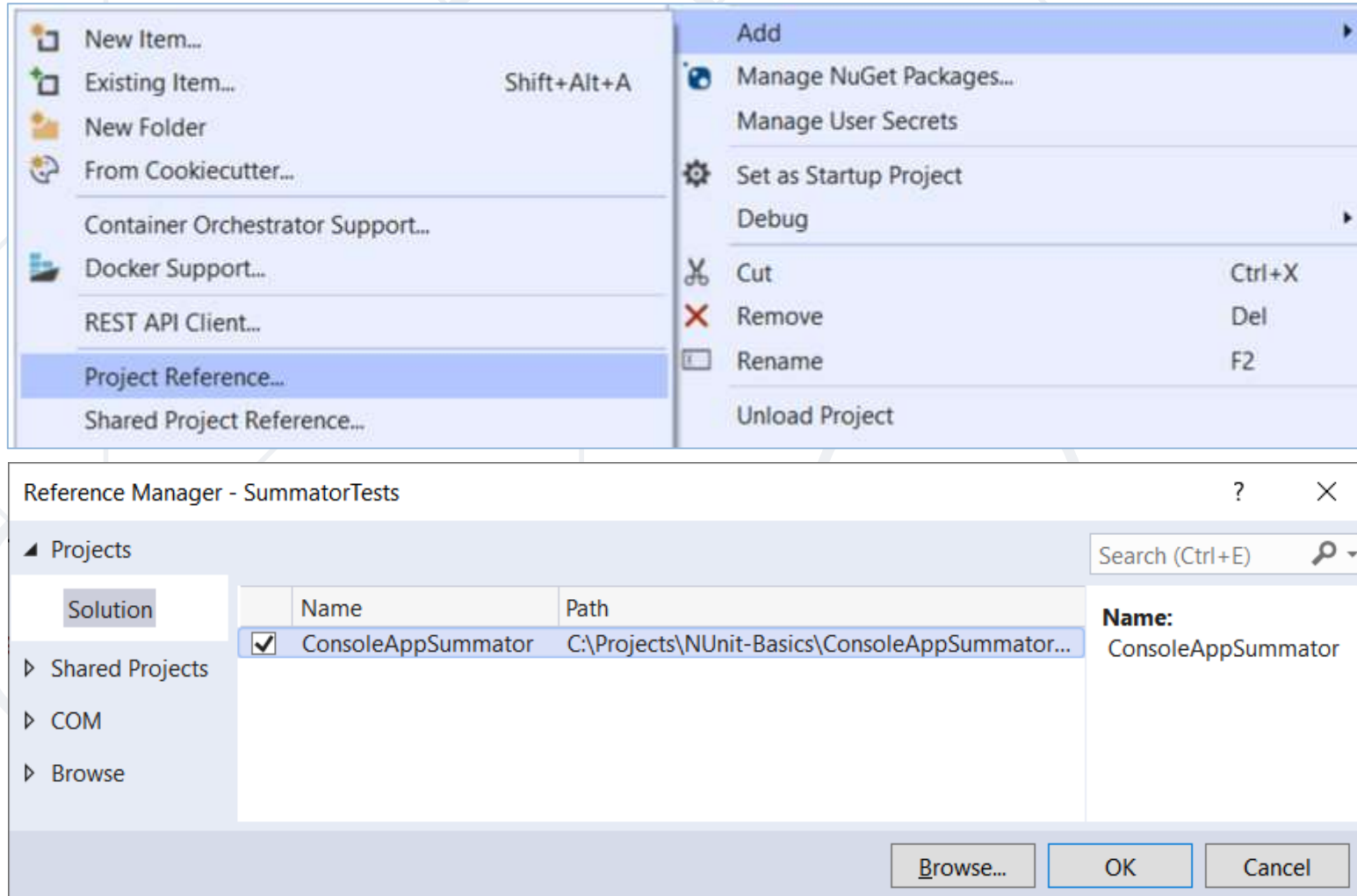


Properties

Alt+Enter

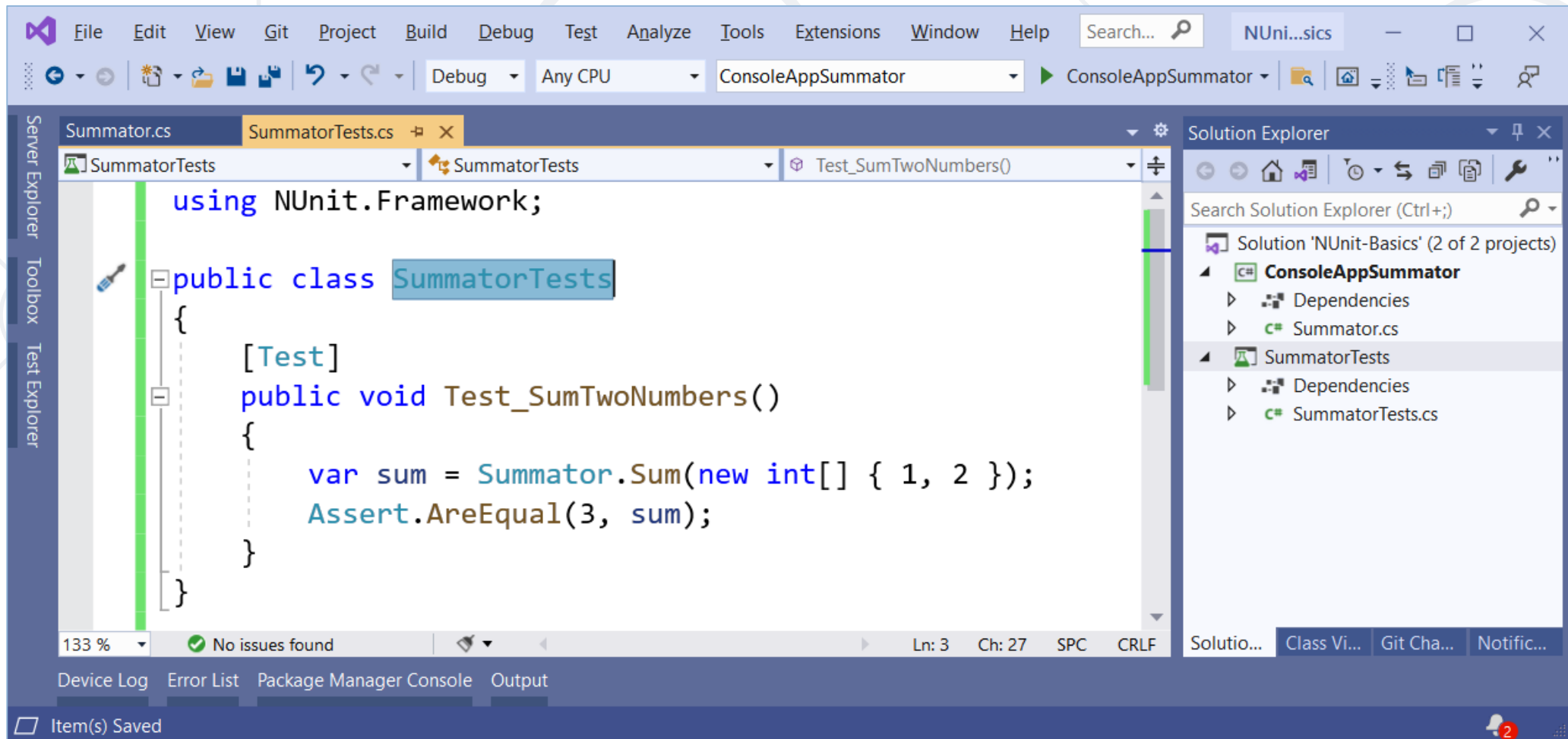
Adding Project Reference

- Add **Project Reference** to the target project for testing:



Writing the First Test

- Writing the first **NUnit** test method:



The screenshot shows the Visual Studio IDE with the following details:

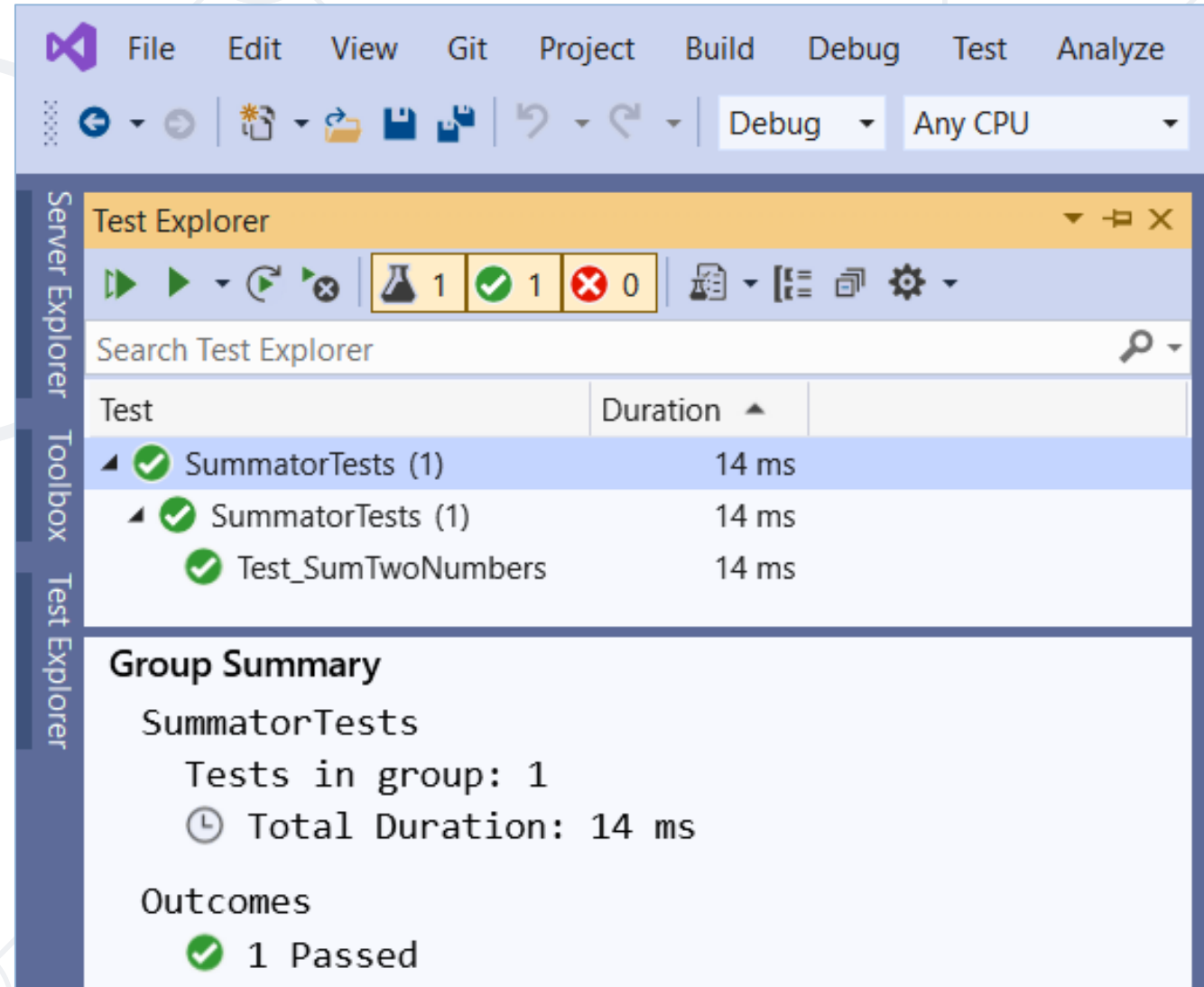
- File Explorer:** Shows the project structure with 'ConsoleAppSummator' and 'SummatorTests' projects.
- Code Editor:** Displays the `SummatorTests.cs` file. The code is as follows:

```
using NUnit.Framework;

public class SummatorTests
{
    [Test]
    public void Test_SumTwoNumbers()
    {
        var sum = Summator.Sum(new int[] { 1, 2 });
        Assert.AreEqual(3, sum);
    }
}
```
- Test Explorer:** Shows the test method `Test_SumTwoNumbers()` with a green status bar indicating it passed.
- Output Window:** Shows the message 'Item(s) Saved'.

Running the Tests

- The **[Test Explorer]** tool in Visual Studio
 - Show the **[Test Explorer]**:
 - **[Ctrl + E] + T**
 - Visualizes the **hierarchy** of tests
 - **Executes** tests
 - **Reports** results

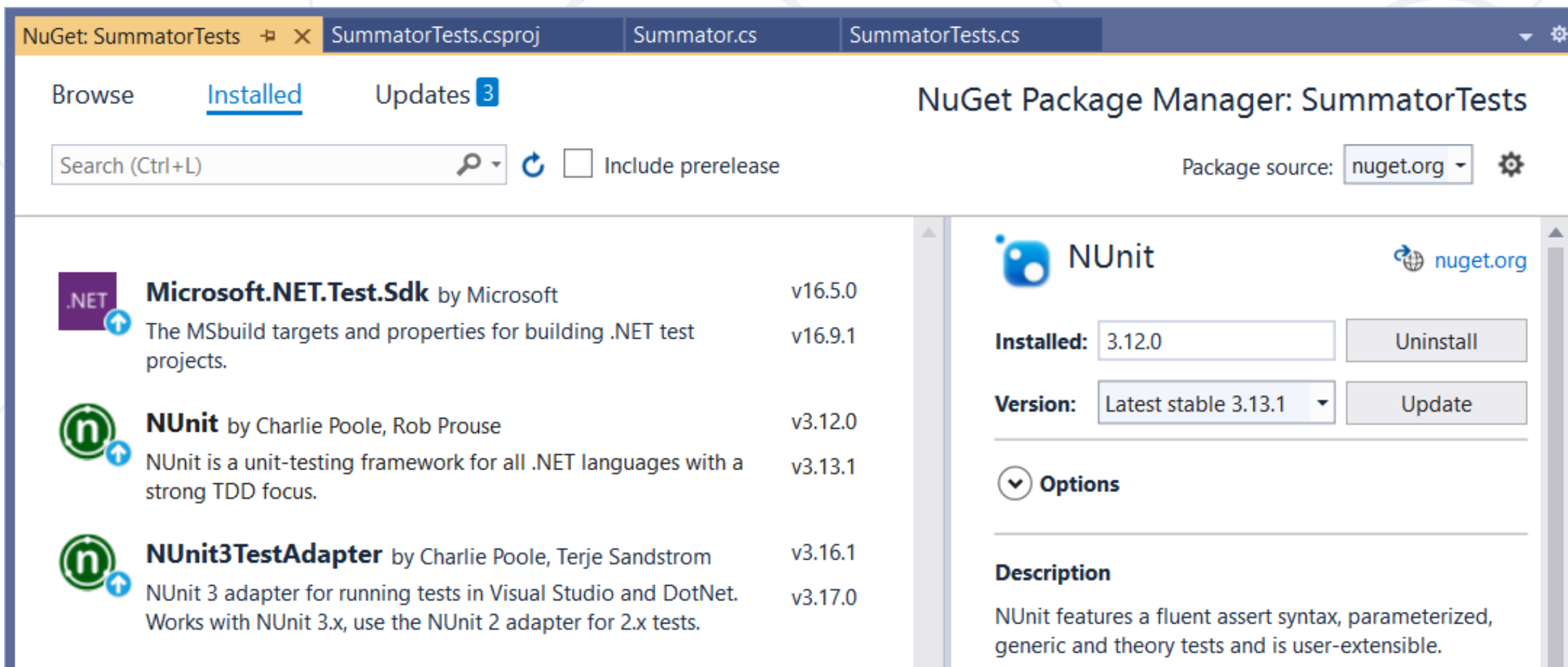




NUnit: Basics

Test Classes and Test Methods

- **NuGet packages**, required to run NUnit tests in Visual Studio



The screenshot shows the NuGet Package Manager window for the project 'SummatorTests'. The 'Installed' tab is selected, showing a list of installed packages. The 'NUnit' package is highlighted, and its details are shown on the right.

| Package Name | Author | Version |
|------------------------|--------------------------------|---------|
| Microsoft.NET.Test.Sdk | Microsoft | v16.5.0 |
| NUnit | Charlie Poole, Rob Prouse | v3.12.0 |
| NUnit3TestAdapter | Charlie Poole, Terje Sandstrom | v3.16.1 |

NUnit by Charlie Poole, Rob Prouse
NUnit is a unit-testing framework for all .NET languages with a strong TDD focus.

Installed: 3.12.0 **Uninstall**

Version: Latest stable 3.13.1 **Update**

Options

Description
NUnit features a fluent assert syntax, parameterized, generic and theory tests and is user-extensible.

Test Classes and Test Methods

- Test classes hold test methods:

```
using NUnit.Framework;
```

Import NUnit

```
[TestFixture]
```

Optional notation

```
public class SummatorTests  
{
```

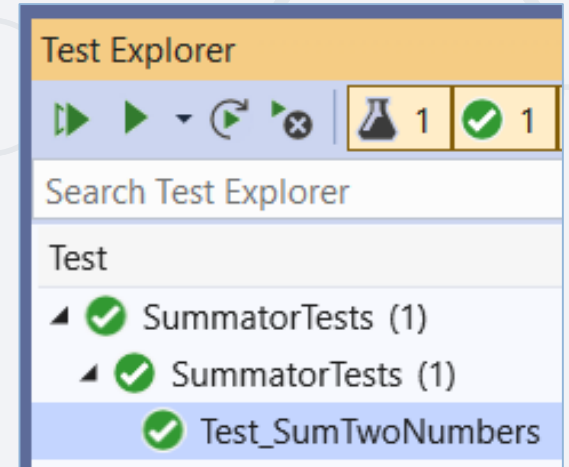
Test class

```
[Test]
```

Test method

```
public void Test_SumTwoNumbers() {  
    var sum = Sum(new int[] { 1, 2 });  
    Assert.AreEqual(3, sum);  
}
```

Assertion



Initialization and Cleanup Methods

```
private Summator summator;
```

```
[SetUp] // or [OneTimeSetUp]
```

```
public void TestInitialize()
```

```
{
```

```
    this.summator = new Summator();
```

```
}
```

```
[TearDown] // or [OneTimeTearDown]
```

```
public void TestCleanup()
```

```
{
```

```
    // ...
```

```
}
```

Executes **before**
each test

Executes **after**
each test

Problem: NUnit Test

- Create console application project
- Add BankAccount class
- Create NUnit Project
- Test the BankAccount class

Solution: NUnit Test (1)

- Create a console **application**
 - Add BankAccount class for us to test

| BankAccount |
|-----------------------|
| +Amount : decimal |
| +BankAccount(decimal) |

- Create a new **NUnit Test Project**
 - Name it like the project you are testing, but with **".Tests"** suffix
- Open **Test Explorer** (Ctrl + E, T or Test->Windows->TestExplorer)

Solution: NUnit Test (2)

- Write your first test

[TestFixture]

Attribute that marks a class that contains tests

```
public class BankAccountTests  
{
```

[Test]

Test Method

```
    public void AccountInitializeWithPositiveValue() {  
        BankAccount account = new BankAccount(2000m);  
        Assert.That(account.Amount, Is.EqualTo(2000m));  
    }  
}
```

Assert class comes with NUnit

The "AAA" Testing Pattern

- Automated tests usually follow the **"AAA" pattern**
 - **Arrange**: prepare the **input** data and entrance conditions
 - **Act**: invoke the **action** for testing
 - **Assert**: check the **output** and exit conditions

```
[Test]
public void Test_SumNumbers()
{
    // Arrange
    var nums = new int[] {3, 5};

    // Act
    var sum = Sum(nums);

    // Assert
    Assert.AreEqual(8, sum);
}
```



How to Write Good Tests

Unit Testing Best Practices

- **Test names** should answer the question "*what's inside?*"
 - Should use **business domain terminology**
 - Should be **descriptive** and **readable**

```
IncrementNumber() {}  
Test1() {}  
TestTransfer() {}
```



```
Test_DepositAddsMoneyToBalance() {}  
Test_DepositNegativeShouldNotAddMoney() {}  
Test_TransferSubtractsFromSourceAddsToDestAccount() {}
```



- Test cases must be **repeatable**
 - Tests should behave the same if you run them many times
 - The expected results must be **consistent** and easily verified
- Test cases should **have no dependencies**
 - The order of test execution should never be important
 - Input data and entrance conditions should be set in the test
 - Test cases may depend on the test initialization only: **[SetUp]**
 - Tests should **cleanup** properly any resources used

Automated Tests: Good Practices (2)

- **Single scenario** per test case, not multiple

```
[Test]
public void Test_Collections_RemoveAtStart()
{
    var names = new Collection<string>("Peter", "Maria", "Steve", "Mia");
    var removed = names.RemoveAt(1);
    Assert.That(removed, Is.EqualTo("Maria"));
    Assert.That(names.ToString(), Is.EqualTo("[Peter, Steve, Mia]"));
}
```

```
[Test]
public void Test_Collections_RemoveAtStart()
{
    var names = new Collection<string>("Peter", "Maria", "Steve", "Mia");
    var removed = names.RemoveAt(1);
    Assert.That(removed, Is.EqualTo("Maria"));
    Assert.That(names.ToString(), Is.EqualTo("[Peter, Steve, Mia]"));
}
```

```
[Test]
public void Test_Collections_RemoveAtEnd()
{
    var names = new Collection<string>("Peter", "Maria", "Steve", "Mia");
    var removed = names.RemoveAt(3);
    Assert.That(removed, Is.EqualTo("Mia"));
    Assert.That(names.ToString(), Is.EqualTo("[Peter, Maria, Steve]"));
}
```

```
[Test]
public void Test_Collections_RemoveAtMiddle()
{
    var names = new Collection<string>("Peter", "Maria", "Steve", "Mia");
    var removed = names.RemoveAt(1);
    Assert.That(removed, Is.EqualTo("Maria"));
    Assert.That(names.ToString(), Is.EqualTo("[Peter, Steve, Mia]"));
}
```

- **Private methods** should be tested indirectly
 - By testing the **public methods** with certain inputs and entrance conditions, that will invoke the target private methods
 - Check the **code coverage** to ensure all code is tested!
- Example:

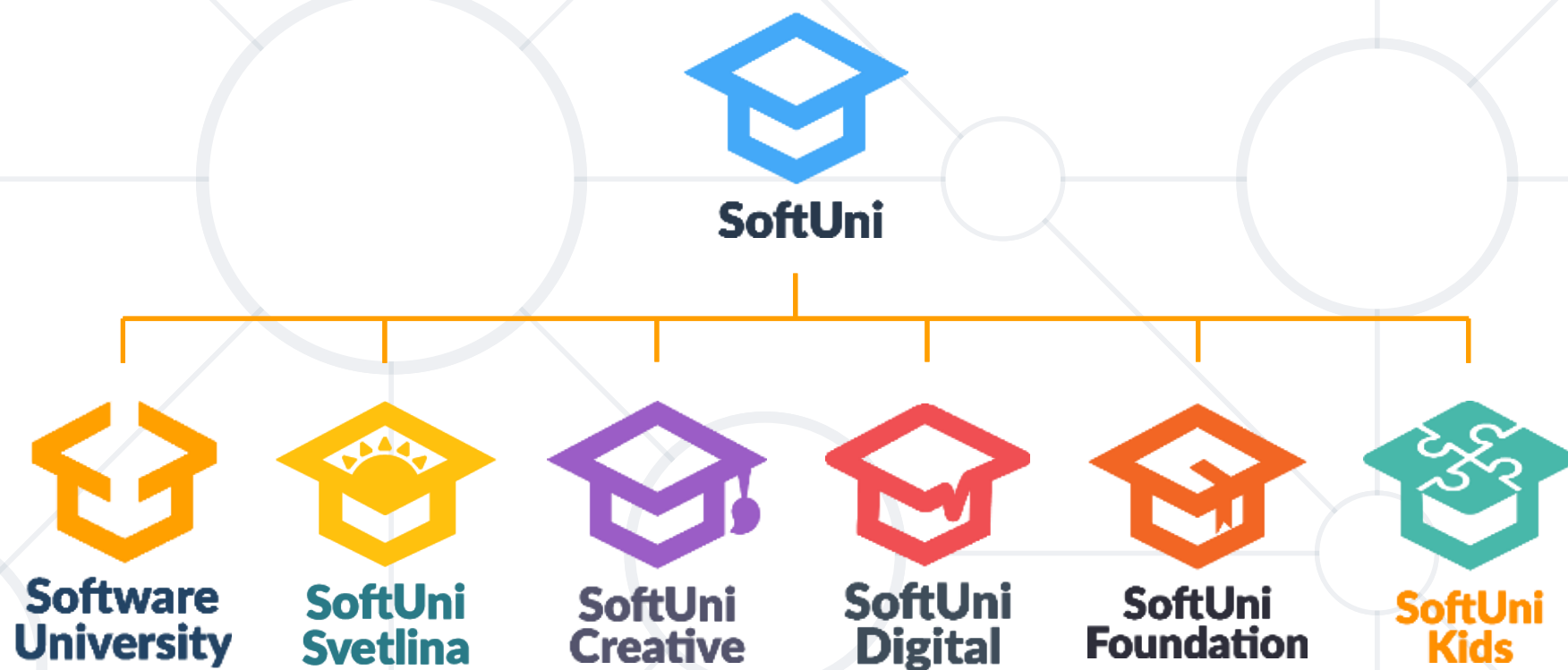
```
public void Add(T item)
{
    this.EnsureCapacity();
    this.items[this.Count] = item;
    this.Count++;
}
```

```
private void EnsureCapacity()
{
    if (this.Count == this.Capacity)
    {
        // Grow the capacity 2 times and move the items
        T[] oldItems = this.items;
        this.items = new T[2 * oldItems.Length];
        for (int i = 0; i < this.Count; i++)
            this.items[i] = oldItems[i];
    }
}
```


- **Unit testing** == automated testing of single component (unit)
- **Testing framework** == foundation for writing tests
- **NUnit** == automated testing framework for C#
- The **AAA pattern**: Arrange, Act, Assert



Questions?



SoftUni Diamond Partners



SCHWARZ



**SUPER
HOSTING
.BG**



INDEAVR
Serving the high achievers

Bosch.io





- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

