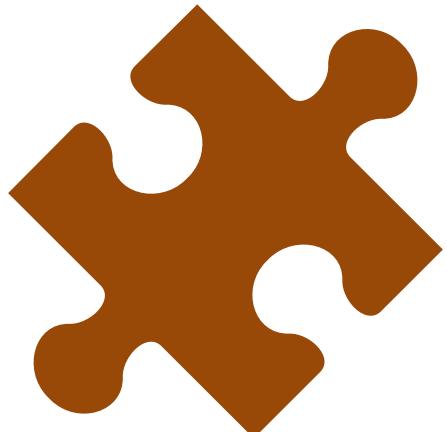


ASP.NET Core

Unit Testing

Hans-Petter Halvorsen



Contents

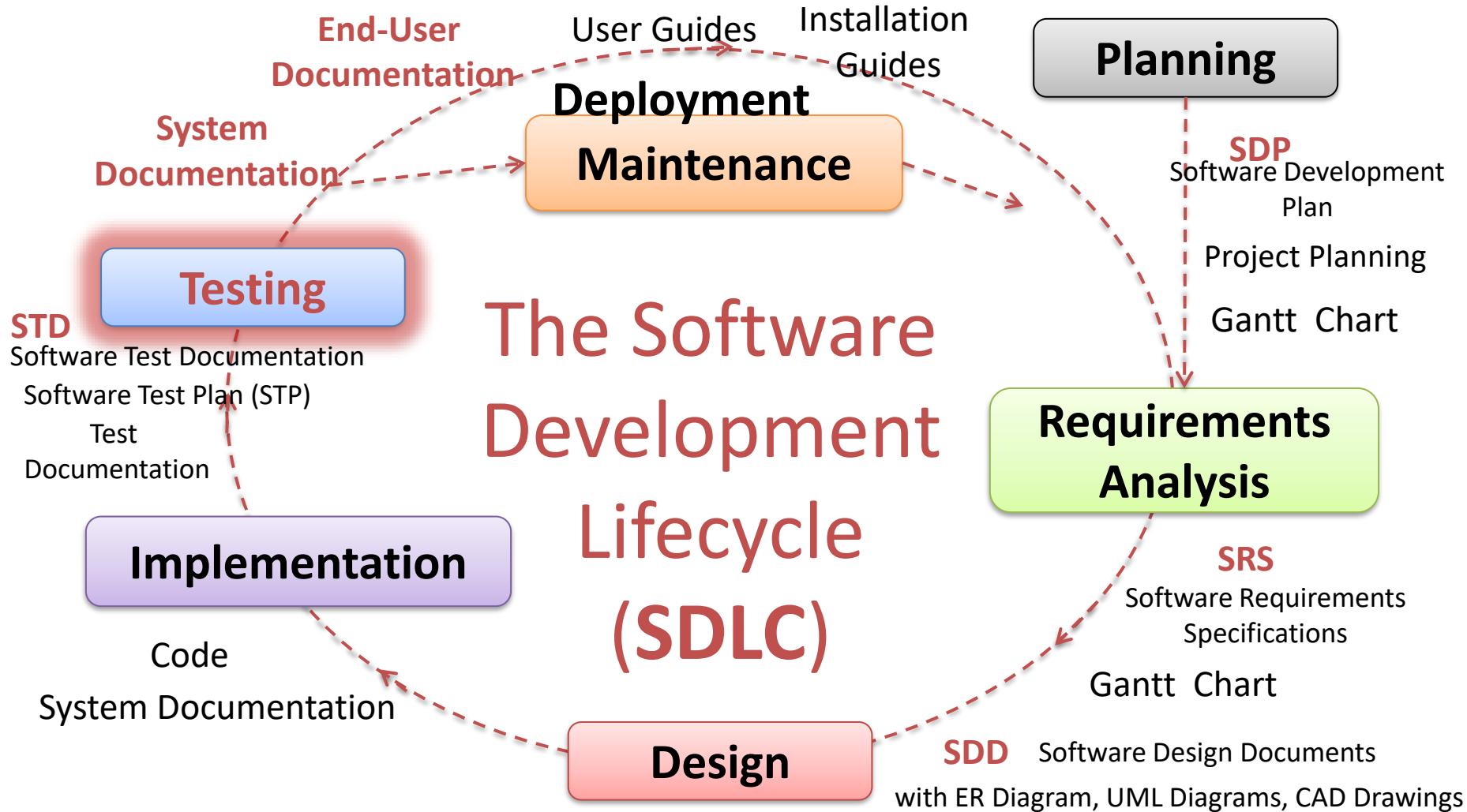
1. What is Testing?
 - Short Introduction to Testing
2. What is Unit Testing?
3. Unit Testing in Visual Studio



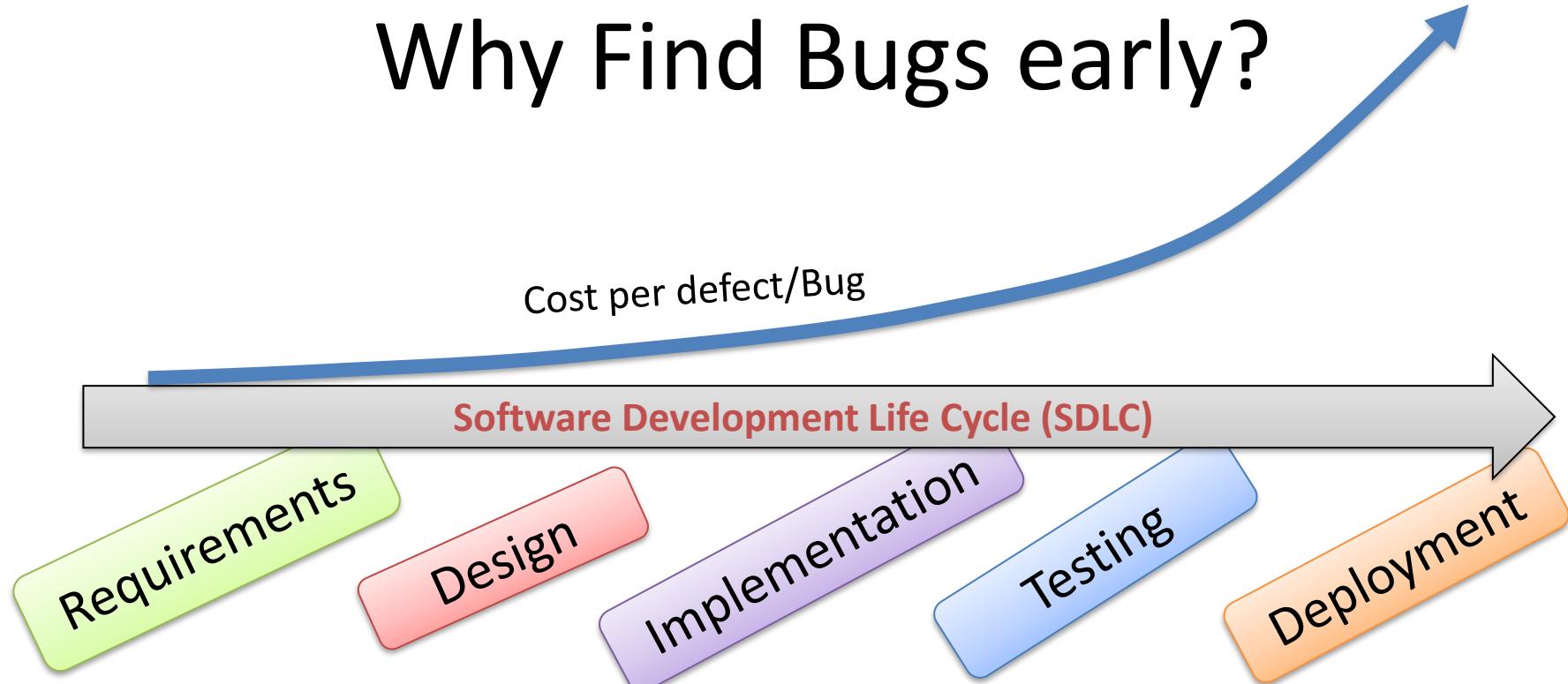
Introduction to Testing

Hans-Petter Halvorsen

The Software Development Lifecycle (SDLC)



Why Find Bugs early?



Testing

Validation Testing

Demonstrate to the Developer and the Customer that the Software meets its Requirements.

Custom Software:

There should be at least one test for every requirement in the SRS document.

Generic Software:

There should be tests for all of the system features that will be included in the product release.

Defect Testing

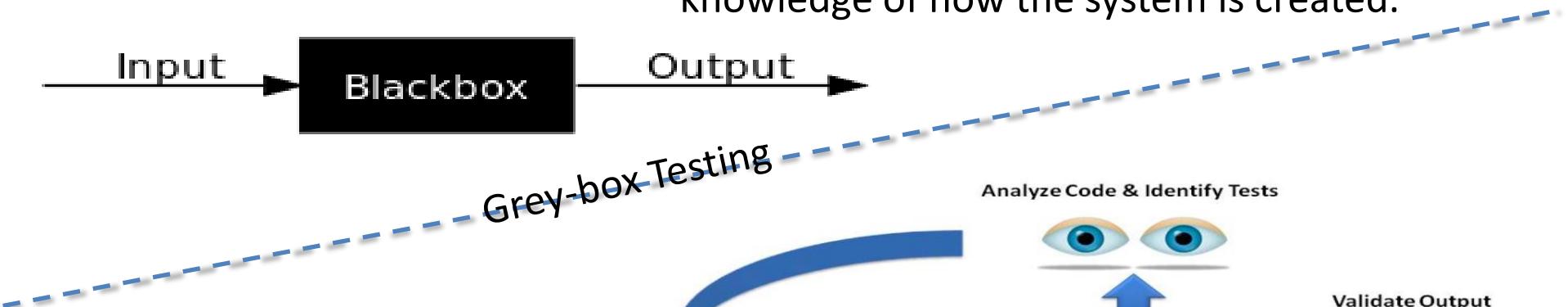
Find inputs or input sequences where the behavior of the software is incorrect, undesirable, or does not conform to its specifications.

These are caused by defects (bugs) in the software.

Test Categories

Black-box vs. White-box Testing

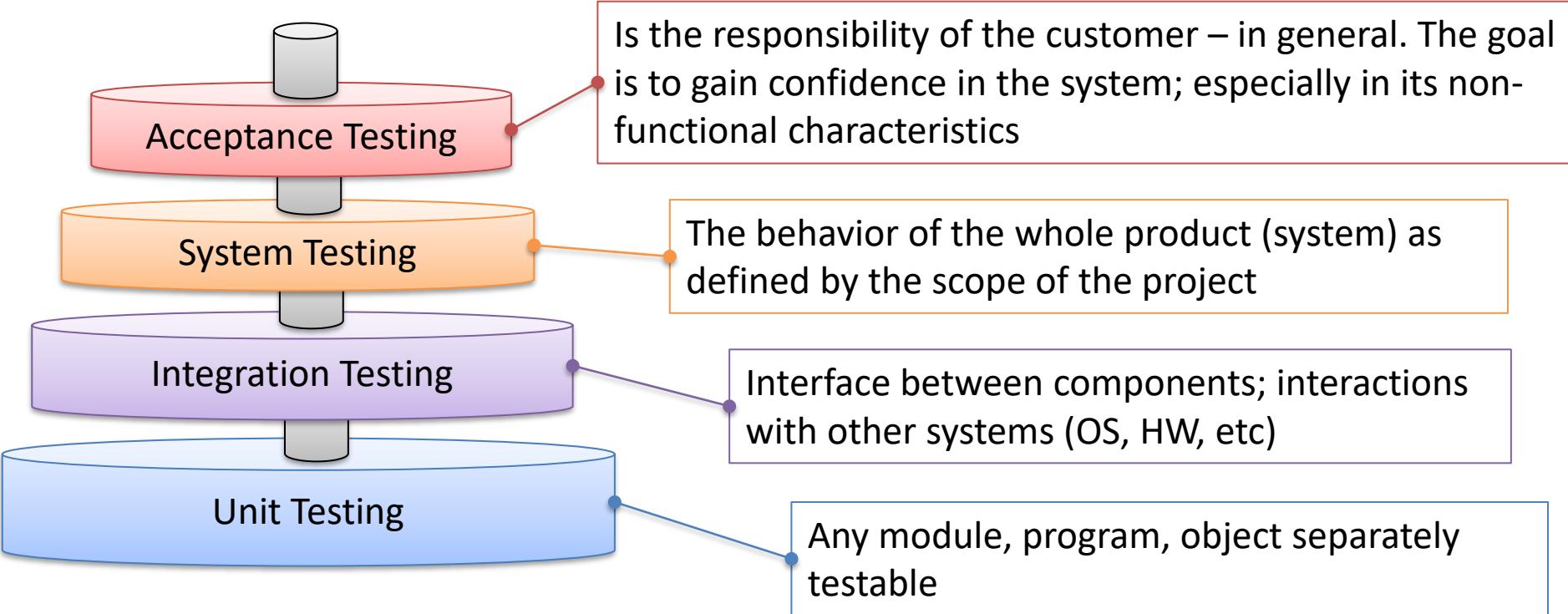
Black-box Testing: You need no knowledge of how the system is created.



White-box Testing: You need to have knowledge of how (Design and Implementation) the system is built

Typically done by Developers, etc

Levels of Testing



Levels of Testing

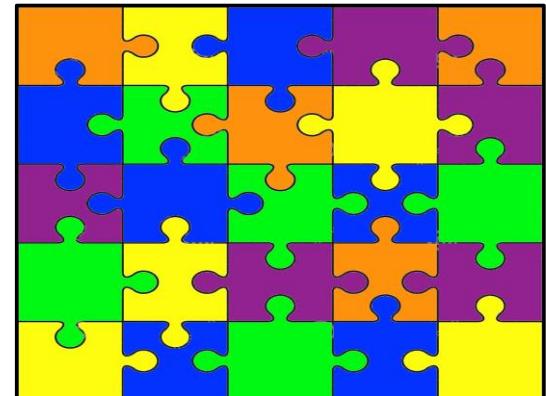


Unit Testing: Test each parts independently and isolated

Integration Testing: Make sure that different pieces work together. Test the Interfaces between the different pieces. Interaction with other systems (Hardware, OS, etc.)

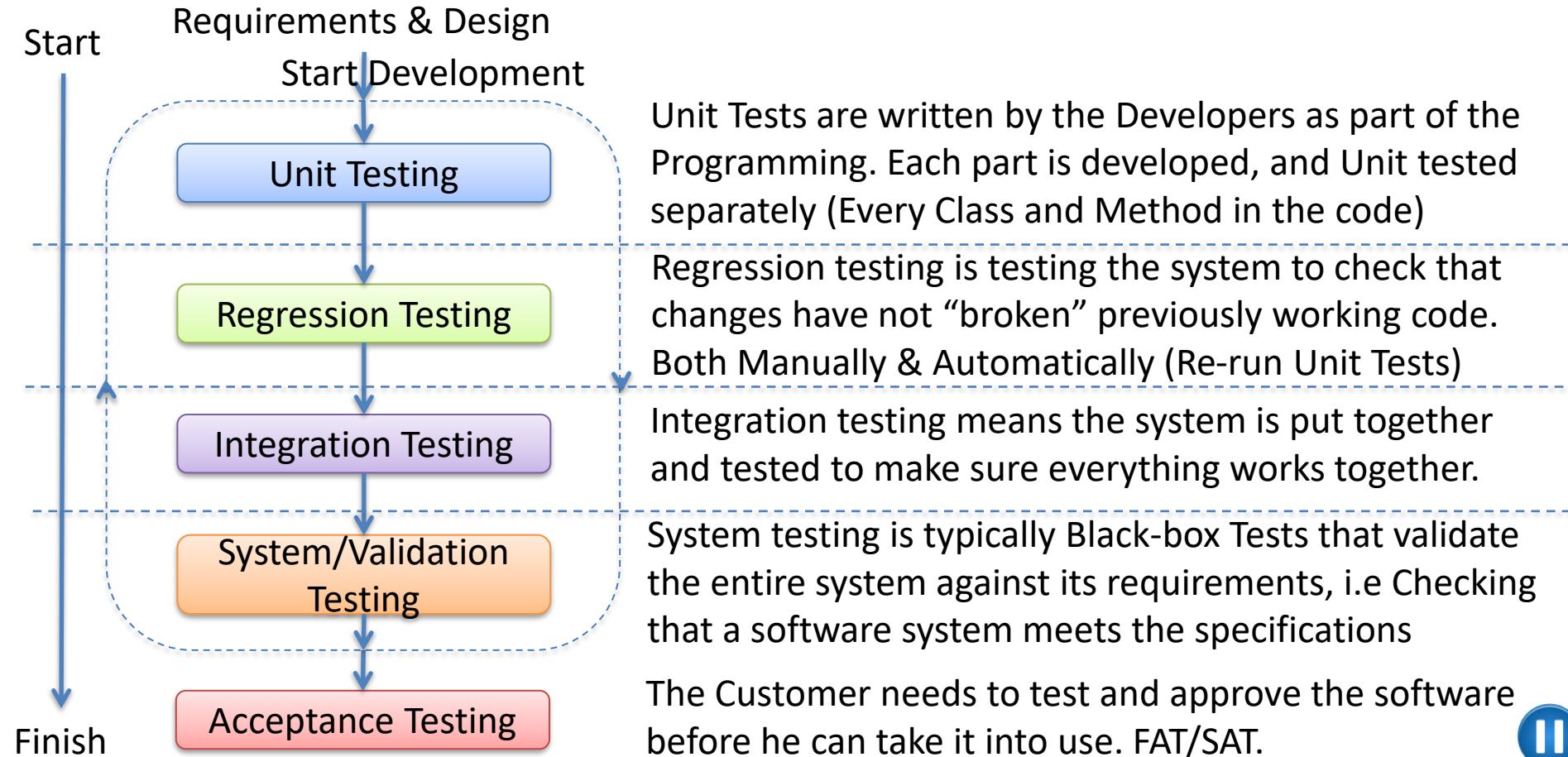


Regression Testing: Test that it still works after a change in the code, i.e., run all Unit Tests, etc.



System Testing: Test the whole system

Levels of Testing



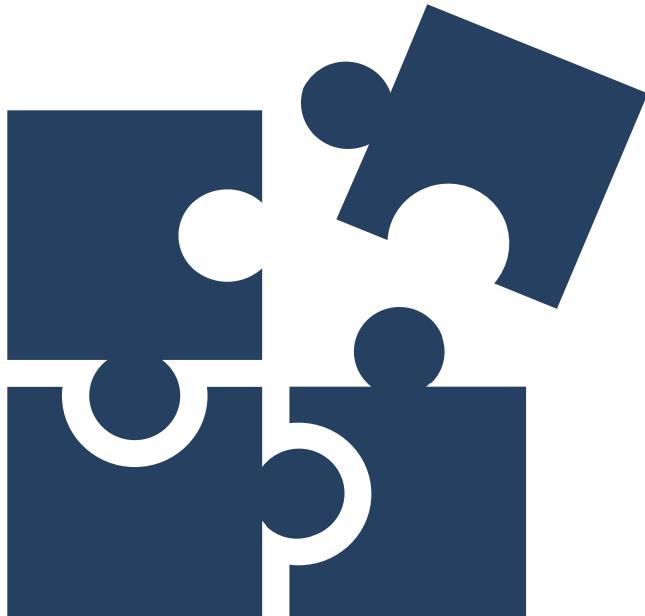


Unit Testing

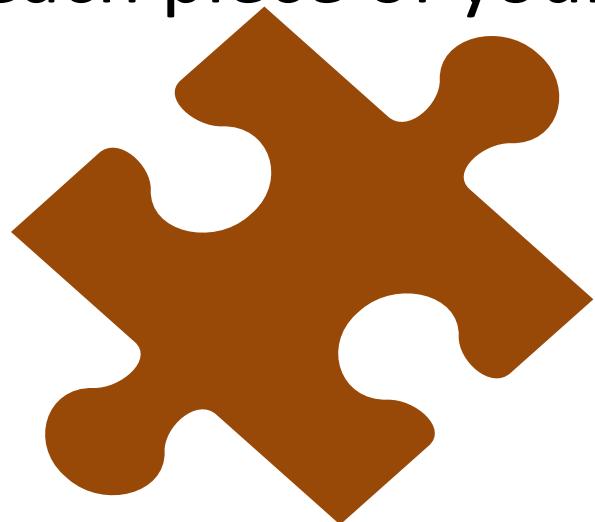
Hans-Petter Halvorsen

Unit Testing

System to be Tested



Then we take out each Unit and Test it by making a Unit Test for each piece of your system



What are Unit Tests

- Unit Testing (or *component testing*) refers to tests that verify the functionality of a specific section of code, usually at the function level.
- **In an object-oriented environment, this is usually at the class and methods level.**
- **Unit Tests are typically written by the developers as part of the programming**
- **Automatically executed** (e.g., Visual Studio and Team Foundation Server have built-in functionality for Unit Testing)



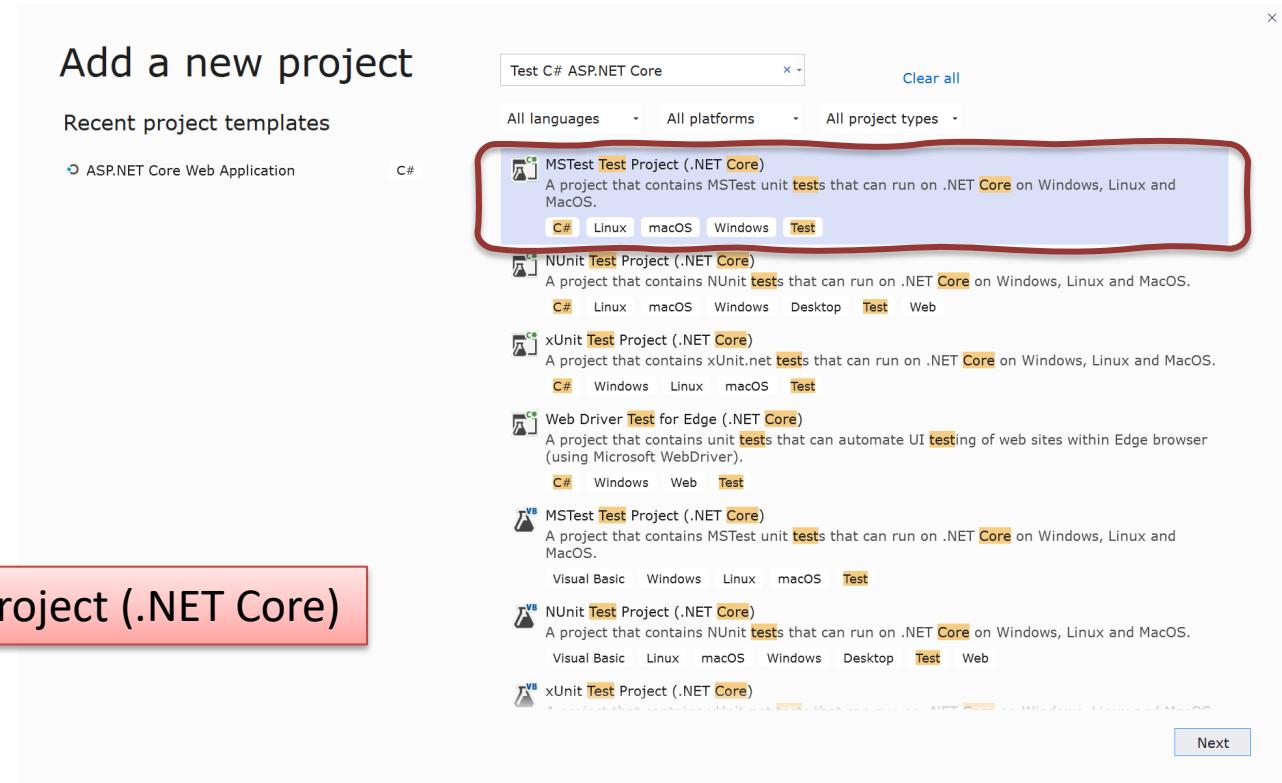
Test Driven Development (TDD)

- Coding and Testing are done in parallel
- The Tests are normally written before the Code
- Introduced as part of eXtreme Programming (XP) (an Agile method)
- **Unit Tests are important part of Software Development today – either you are using TDD or not**

Unit Tests Frameworks in Visual Studio

- MSTest
- NUnit
- xUnit

We will use **MSTest Test Project (.NET Core)**



Basic Concept in Unit Testing

The basic concept in Unit Testing is to Compare the results when running the Methods with some Input Data (“Actual”) with some Known Results (“Expected”)

The Assert Class contains different Methods that can
be used in Unit Testing

Example:

...

```
Assert.AreEqual(expected, actual, 0.001, "Test failed because...");
```

All Unit Tests

Framework have the
Assert Class

Compare

Error margin

Error message shown if
the Test fails

Unit Tests – Best Practice

- A Unit Test must only do one thing
- Unit Test must run independently
- Unit Tests must not be depending on the environment
- Test Functionality rather than implementation
- Test public behavior; private behavior relates to implementation details
- Avoid testing UI components
- Unit Tests must be easy to read and understand
- Create rules that make sure you need to run Unit Tests (and they need to pass) before you can Check-in your Code in the Source Code Control System

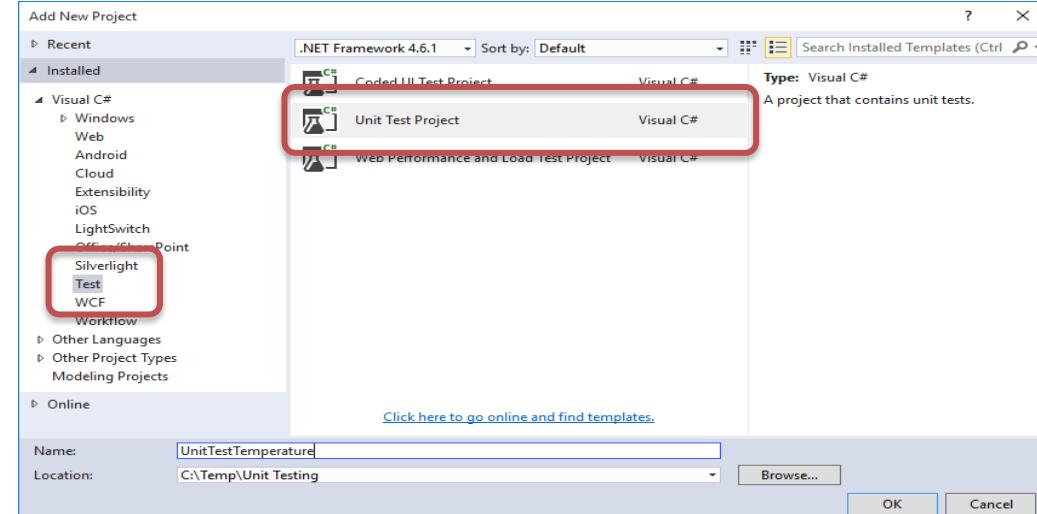


Unit Testing in Visual Studio

Hans-Petter Halvorsen

Unit Testing in Visual Studio

- Visual Studio have built-in features for Unit Testing
- We need to include a “Test Project” in our Solution



Test Method Requirements

A test method must meet the following requirements:

- The method must be decorated with the **[TestMethod]** attribute.
- The method must return void.
- The method cannot have parameters.



Example

Unit Testing in Visual Studio

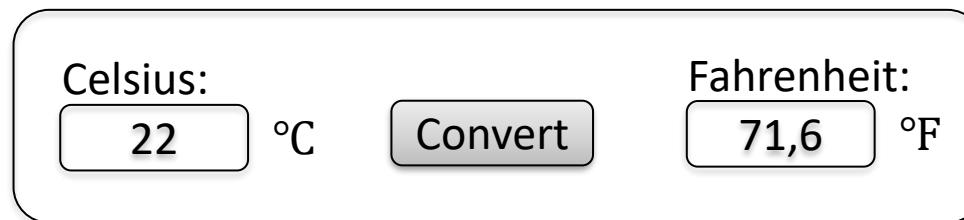
Hans-Petter Halvorsen

ASP.NET Core Application

Convert to Fahrenheit

Create the following Application (e.g., WinForm App or ASP.NET App)

A simple sketch of the User Interface:



Conversion Formula:

$$T_F = \frac{9}{5} T_C + 32$$

User Interface

FahrenheitApp [Home](#) [Temperature](#) [Privacy](#)

Temperature Conversion

Temperature [Celsius]:

22

Temperature [Fahrenheit]:

54

[Convert](#)

Add Class i your Models Folder

```
namespace FahrenheitApp.Models
{
    public static class Temperature
    {
        public static double CelciusToFahrenheit(double Tc)
        {
            double Tf;

Tf = 9 / 5 * Tc + 32;

            return Tf;
        }
    }
}
```

Create your GUI

The screenshot shows a Microsoft Visual Studio interface with the following details:

- File Menu:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Project Title:** FahrenheitApp.
- Solution Explorer:** Shows the project structure:
 - Solution 'FahrenheitApp' (1 of 1 project)
 - FahrenheitAPP
 - Connected Services
 - Dependencies
 - Properties
 - wwwroot
 - Models
 - Temperature.cs
 - Pages
 - Shared
 - _ViewImports.cshtml
 - _ViewStart.cshtml
 - Error.cshtml
 - Index.cshtml
 - Privacy.cshtml
 - Temperature.cshtml
 - Temperature.cshtml.cs
 - _Pages_Temperature
 - appsettings.json
 - Program.cs
 - Startup.cs
- Properties Window:** Located at the bottom right, showing basic file properties.
- Code Editor:** Displays the 'Temperature.cshtml' file content, which includes C# Razor code for a temperature conversion form.
- Toolbox:** On the left side, showing various development tools.
- Status Bar:** At the bottom, showing '100 %' and 'No issues found'.
- Bottom Bar:** Contains icons for Add to Source Control, Undo, Redo, and other navigation.

Testing

FahrenheitApp Home Temperature Privacy

Temperature Conversion

Temperature [Celsius]:

22

Temperature [Fahrenheit]:

54

Convert

$$T_F = \frac{9}{5} T_C + 32$$

$$T_F = \frac{9}{5} \cdot 22 + 32$$

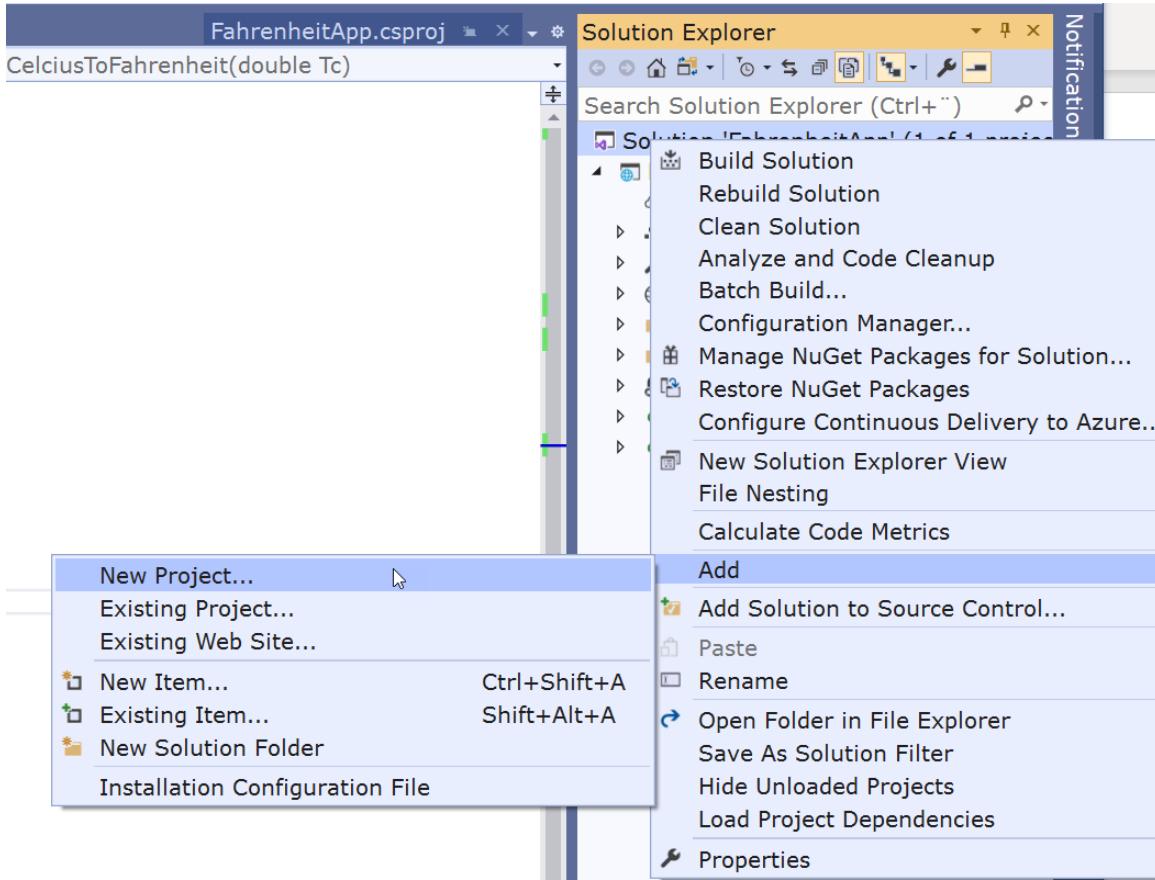
$$= 71.6$$

© Developed by Hans-Petter Halvorsen (<https://www.halvorsen.blog>)

We get wrong Answer!

Unit Test Project

Create Unit Test Project



Add a new project

Recent project templates

ASP.NET Core Web Application

C#

Clear all

All languages

All platforms

All project types

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 Windows 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

Web Driver Test for Edge (.NET Core)

A project that contains unit tests that can automate UI testing of web sites within Edge browser (using Microsoft WebDriver).

C# Windows Web Test

MSTest Test Project (.NET Core)

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

Visual Basic 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.

Visual Basic 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.

Visual Basic Windows Linux macOS Test

Next



Configure your new project

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

Project name

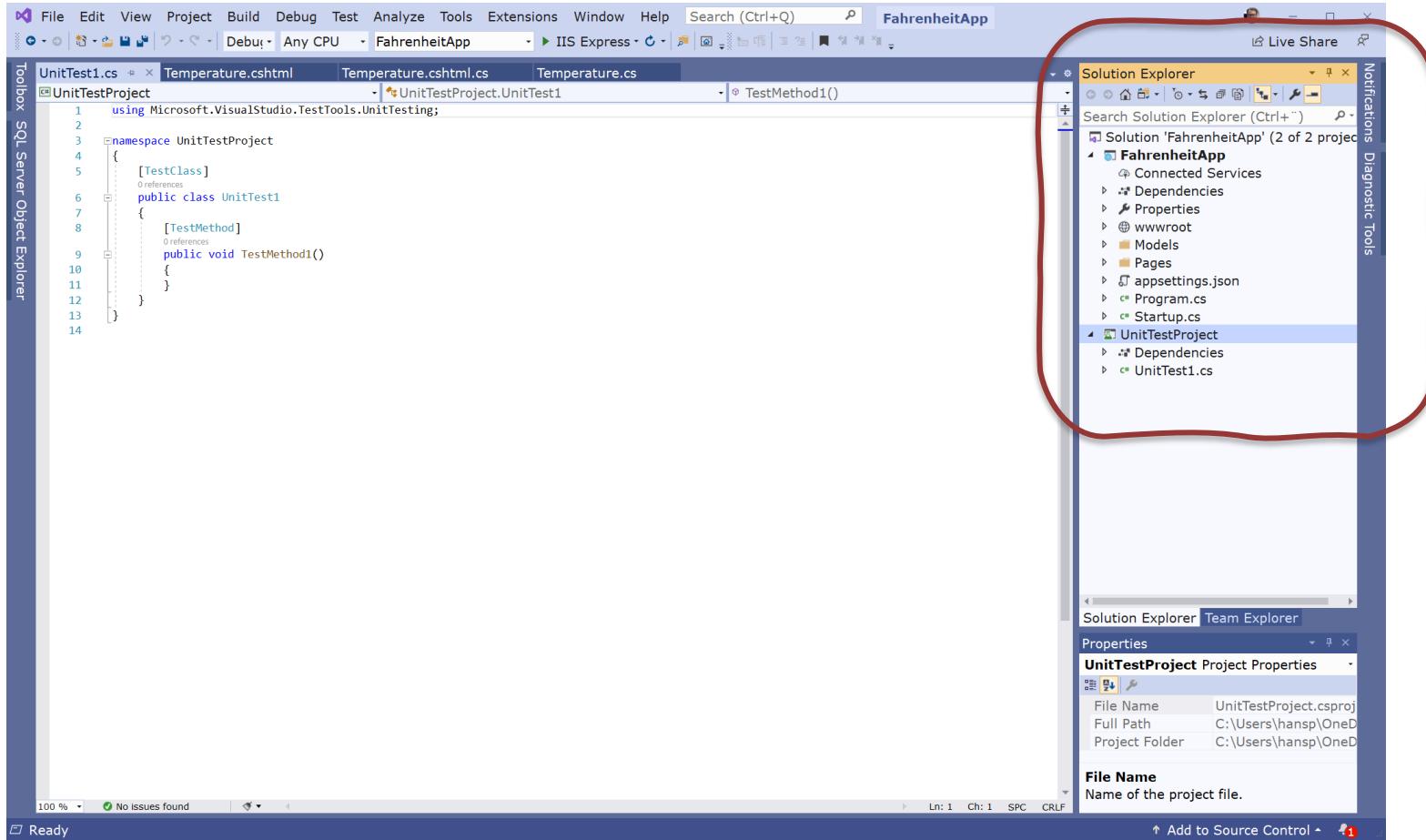
Location

...

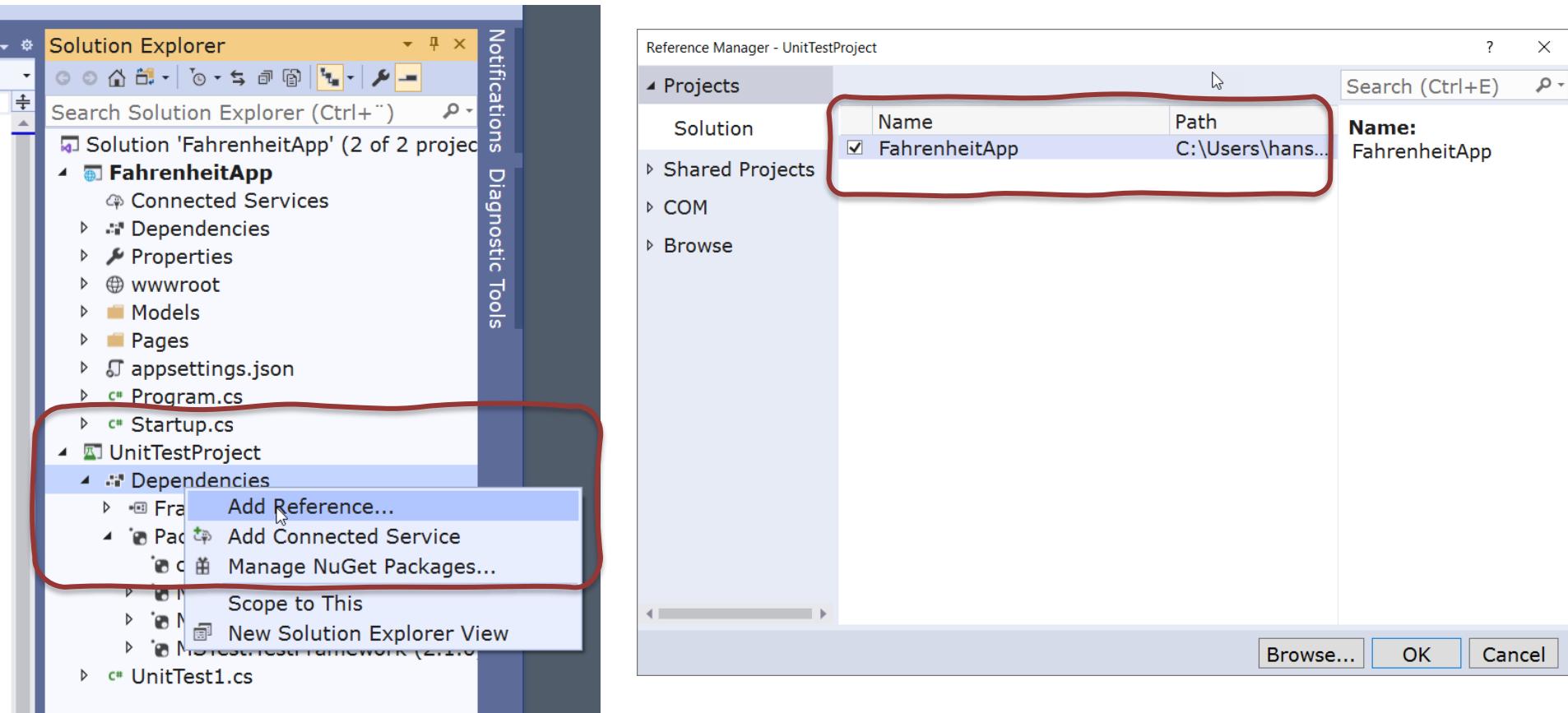
Back

Create

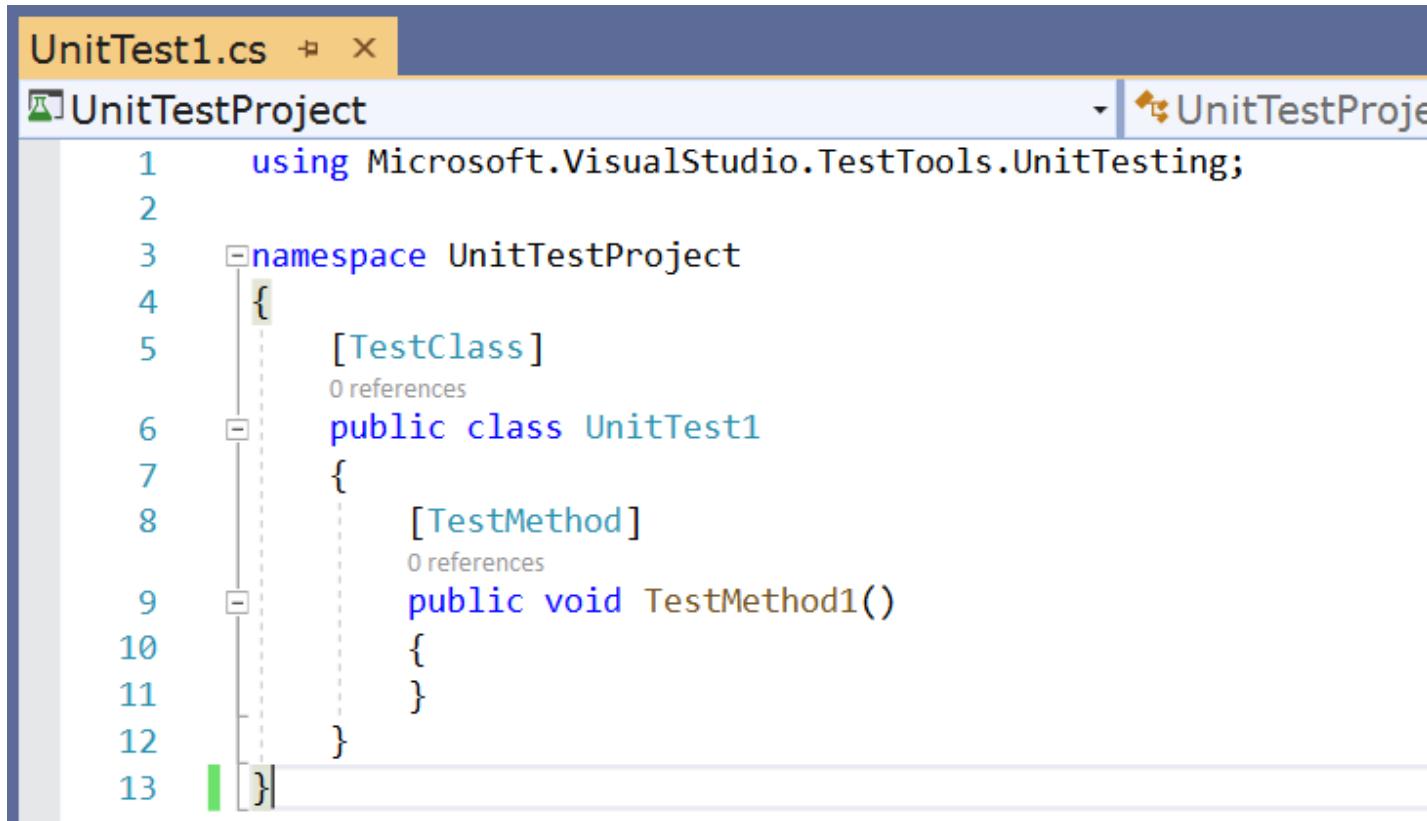
You have now 2 Projects in your Solution Explorer



Add Reference to the Code under Test



Create the Unit Test Code



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar at the top displays "UnitTest1.cs" and "UnitTests" under the "UnitTests" tab. The main area is a code editor showing C# unit test code:

```
1  using Microsoft.VisualStudio.TestTools.UnitTesting;
2
3  namespace UnitTestProject
4  {
5      [TestClass]
6      public class UnitTest1
7      {
8          [TestMethod]
9          public void TestMethod1()
10         {
11     }
12 }
13 }
```

The code uses color-coded syntax highlighting for keywords like `using`, `namespace`, `public`, and `class`. Braces and brackets are also highlighted. The code editor has a vertical scroll bar on the left and a status bar at the bottom.

Create the Unit Test Code

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

- Menu Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Project Name:** FahrenheitApp.
- Toolbox:** Available on the left side.
- Solution Explorer:** Shows the solution structure:
 - Solution 'FahrenheitApp' (2 of 2 projects)
 - FahrenheitApp
 - Connected Services
 - Dependencies
 - Properties
 - wwwroot
 - Models
 - Pages
 - appsettings.json
 - Program.cs
 - Startup.cs
 - UnitTestProject
 - Dependencies
 - Frameworks
 - Packages
 - coverlet.collector (1.2.0)
 - Microsoft.NET.Test.Sdk (16.5.0)
 - MSTest.TestAdapter (2.1.0)
 - MSTest.TestFramework (2.1.0)
 - Projects
 - UnitTestTemperature.cs
- Properties Window:** Available at the bottom right.
- Status Bar:** 100 %, No issues found, Ln: 12 Ch: 10 SPC CRLF.
- Bottom Status Bar:** Build succeeded, Add to Source Control (with a red notification icon).

The code editor displays the `UnitTestTemperature.cs` file with the following content:

```
1  using Microsoft.VisualStudio.TestTools.UnitTesting;
2  using FahrenheitApp;
3  using FahrenheitApp.Models;
4
5  namespace UnitTestTemperature
6  {
7      [TestClass]
8      public class UnitTestFahrenheit
9      {
10         [TestMethod]
11         public void TestFahrenheitConversion()
12         {
13             double temperatureCelcius = 22;
14             double temperatureFahrenheitActual;
15             double temperatureFahrenheitExpected = 71.6;
16
17             temperatureFahrenheitActual = Temperature.CelciusToFahrenheit(temperatureCelcius);
18
19             Assert.AreEqual(temperatureFahrenheitExpected, temperatureFahrenheitActual, 0.001, "Temperature conversion not correctly");
20         }
21     }
22 }
```

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using FahrenheitApp.Models;
```

```
namespace UnitTestTemperature
```

```
{
```

```
    [TestClass]
```

```
    public class UnitTestFahrenheit
```

```
{
```

```
    [TestMethod]
```

```
    public void TestFahrenheitConversion()
```

```
{
```

```
    double temperatureCelcius = 22;
```

```
    double temperatureFahrenheitActual;
```

```
    double temperatureFahrenheitExpected = 71.6;
```

```
    temperatureFahrenheitActual = Temperature.CelciusToFahrenheit(temperatureCelcius);
```

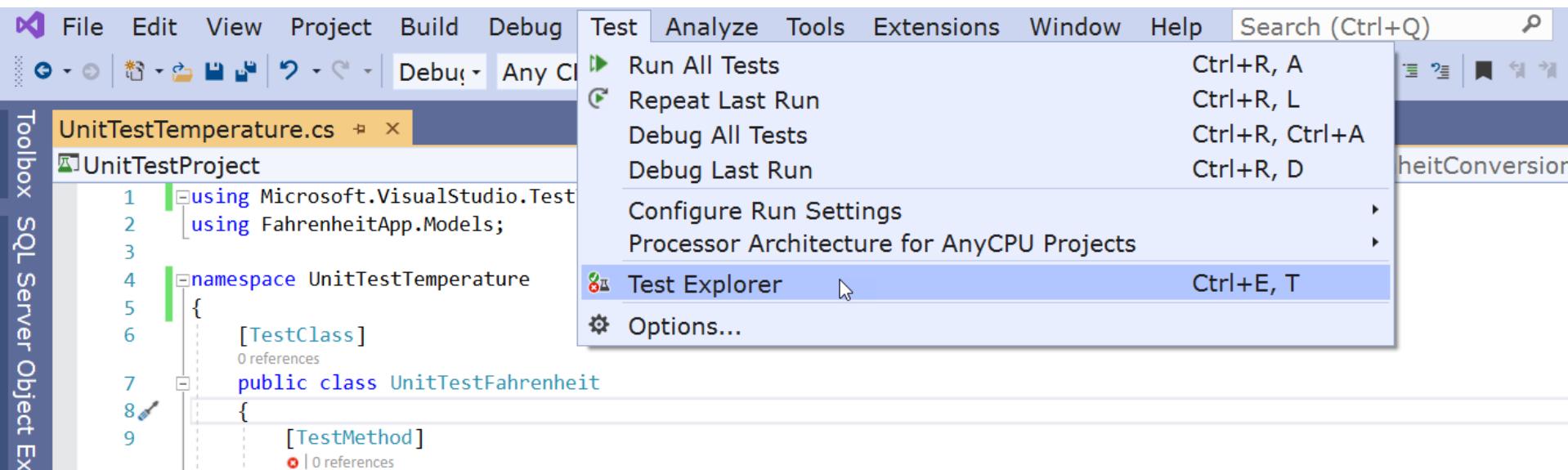
```
    Assert.AreEqual(temperatureFahrenheitExpected, temperatureFahrenheitActual, 0.001, "Temperature conversion  
not correctly");
```

```
}
```

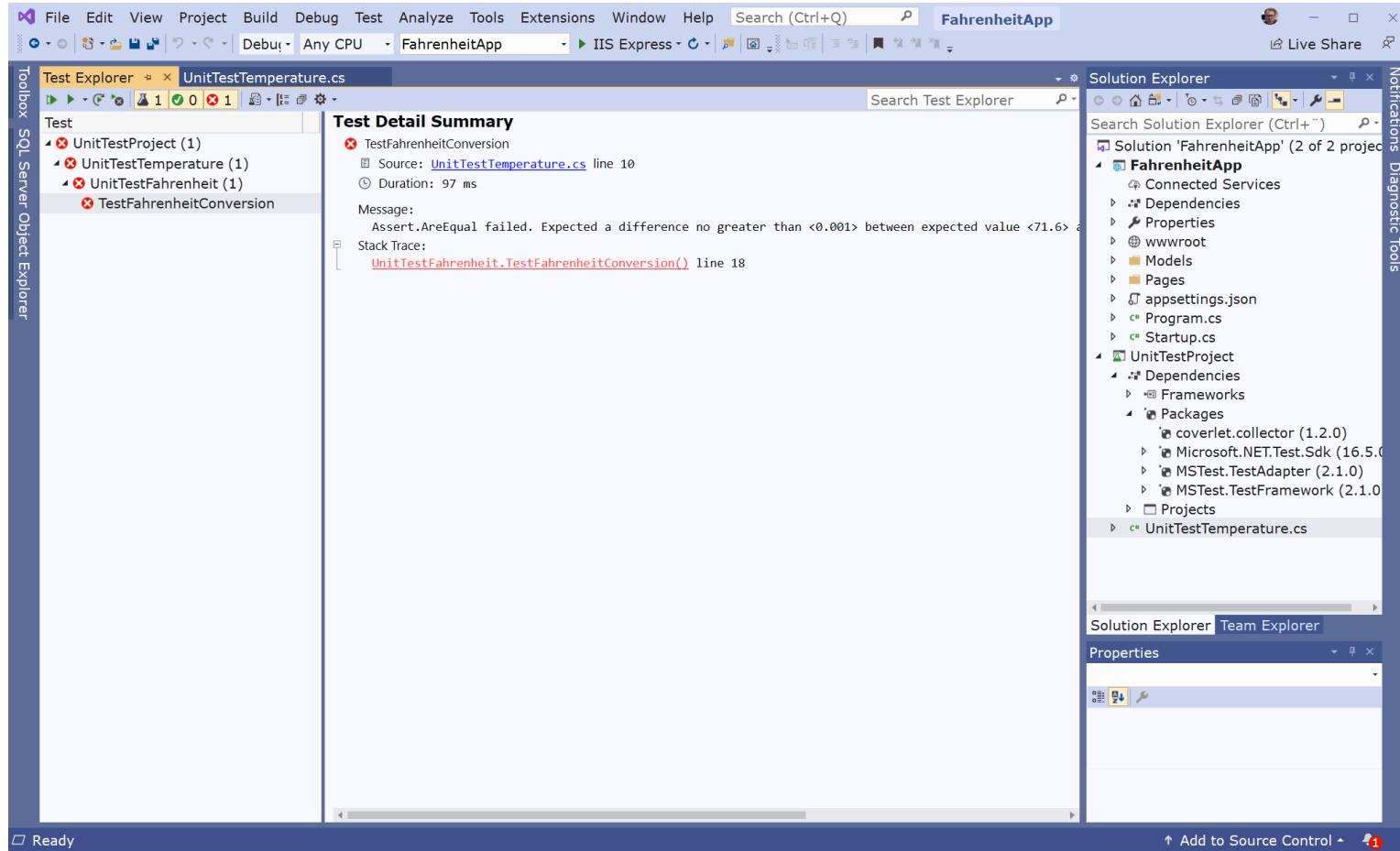
```
}
```

```
}
```

Test Explorer



Start Running the Unit Test



Test Results

The screenshot shows the Microsoft Test Explorer interface. The title bar displays "Test Explorer" and "UnitTestTemperature.cs". The toolbar includes icons for running tests, stopping, and filtering. The status bar shows 1 failed test (red) and 0 passed tests (green). The main area is titled "Test Detail Summary" and shows a failed test named "TestFahrenheitConversion". The test details include:

- Source: [UnitTestTemperature.cs](#) line 10
- Duration: 25 ms

The "Message" section indicates an `Assert.AreEqual` failure with the message: "Expected a difference no greater than <0.001> between expected value <71.6> and actual value <71.5>". The "Stack Trace" section shows the call stack: [UnitTestFahrenheit.TestFahrenheitConversion\(\)](#) line 18.

Debugging

$$T_F = \frac{9}{5} T_C + 32$$

```
namespace FahrenheitApp.Models
```

```
{
```

```
    public static class Temperature
```

```
{
```

```
    public static double CelciusToFahrenheit(double Tc)
```

```
{
```

```
    double Tf;
```

Tf = 9 / 5 * Tc + 32;

```
    return Tf;
```

```
}
```

```
}
```

```
}
```

Probably Error in Formula?
What is wrong?

Fixing Bugs

$$T_F = \frac{9}{5} T_C + 32$$

```
namespace FahrenheitApp.Models
```

```
{
```

```
    public static class Temperature
```

```
{
```

```
    public static double CelciusToFahrenheit(double Tc)
```

```
{
```

```
    double Tf;
```

```
Tf = Tc * 9/5 + 32;
```

```
    return Tf;
```

```
}
```

```
}
```

```
}
```

Re-run Unit Test

The screenshot shows the Visual Studio Test Explorer window. The status bar at the top indicates 1 test, 1 passed, and 0 failed. The left pane lists test projects and their results:

- UnitTestProject (1) - 1 passed
- UnitTestTemperature (1) - 1 passed
- UnitTestFahrenheit (1) - 1 passed
- TestFahrenheitConversion - 1 passed

The right pane displays the "Test Detail Summary" for the selected test, "TestFahrenheitConversion". It includes the following details:

- TestFahrenheitConversion
- Source: [UnitTestTemperature.cs](#) line 10
- Duration: 34 ms

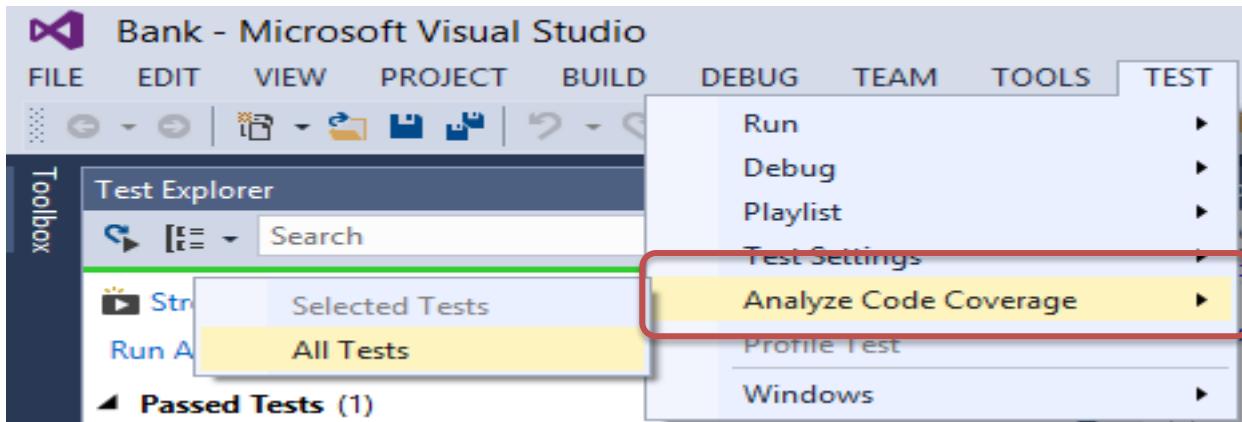
Everything Works! The Test Passed!

Checking Code Coverage

Note! The code coverage feature is available only in **Visual Studio Enterprise** edition.

Code Coverage

- Code coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested.
- Depending on the input arguments, different parts of the code will be executed. Unit Tests should be written to cover all parts of the code.



Note! The code coverage feature is available only in **Visual Studio Enterprise** edition.

Code Coverage Results

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)	
Hans-Petter_HANSPH_LAPTOP 2016-03-30 12_34...	92	93,88 %	6	6,12 %	
temperatureapp.exe	92	97,87 %	2	2,13 %	
TemperatureApp	87	97,75 %	2	2,25 %	
Form1	82	100,00 %	0	0,00 %	
Program	5	100,00 %	0	0,00 %	
TemperatureConvert	0	0,00 %	2	100,00 %	
CelciusToFahrenheit(double)	0	0,00 %	2	100,00 %	
TemperatureApp.Properties	5	100,00 %	0	0,00 %	
Settings	5	100,00 %	0	0,00 %	
unittesttemperature.dll	0	0,00 %	4	100,00 %	

In this case the Unit Test covered 100% of the code. If we use If...Else... or similiar, we typically need to write Unit Test for each If...Else... in order to cover all the Code

References

- <https://docs.microsoft.com/en-us/visualstudio/test/getting-started-with-unit-testing>

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: <https://www.halvorsen.blog>

YouTube: <https://www.youtube.com/IndustrialITandAutomation>

