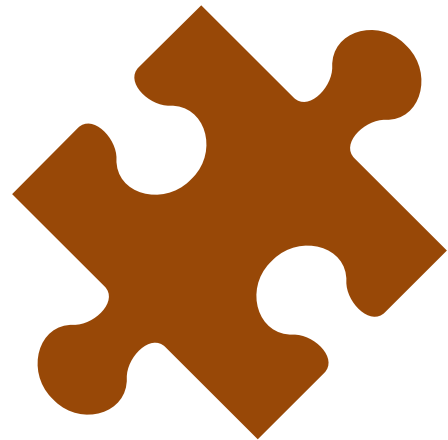


ASP.NET Core

Unit Testing



Hans-Petter Halvorsen

Contents

1. What is Testing?

- Short Introduction to Testing

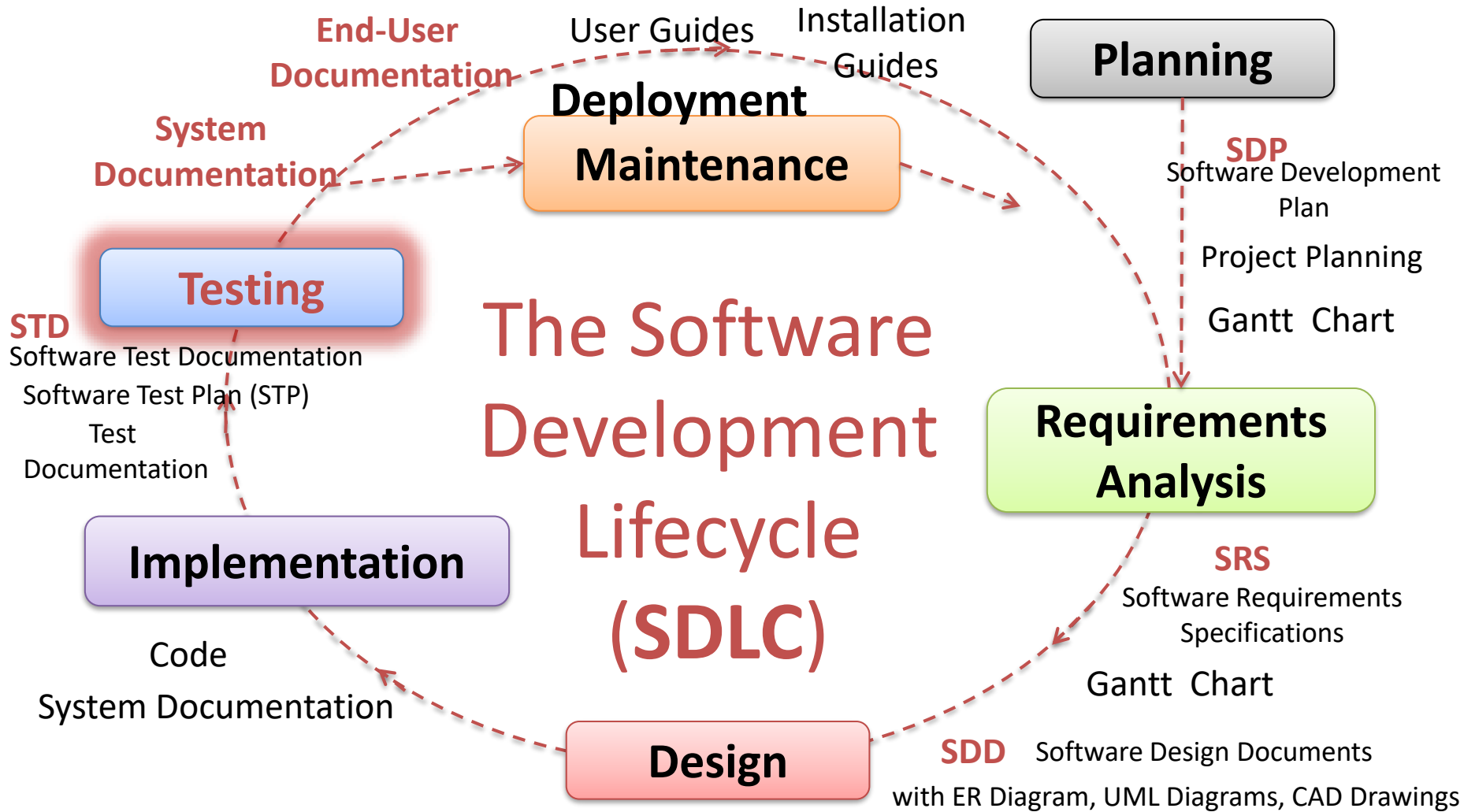
2. What is Unit Testing?

3. Unit Testing in Visual Studio

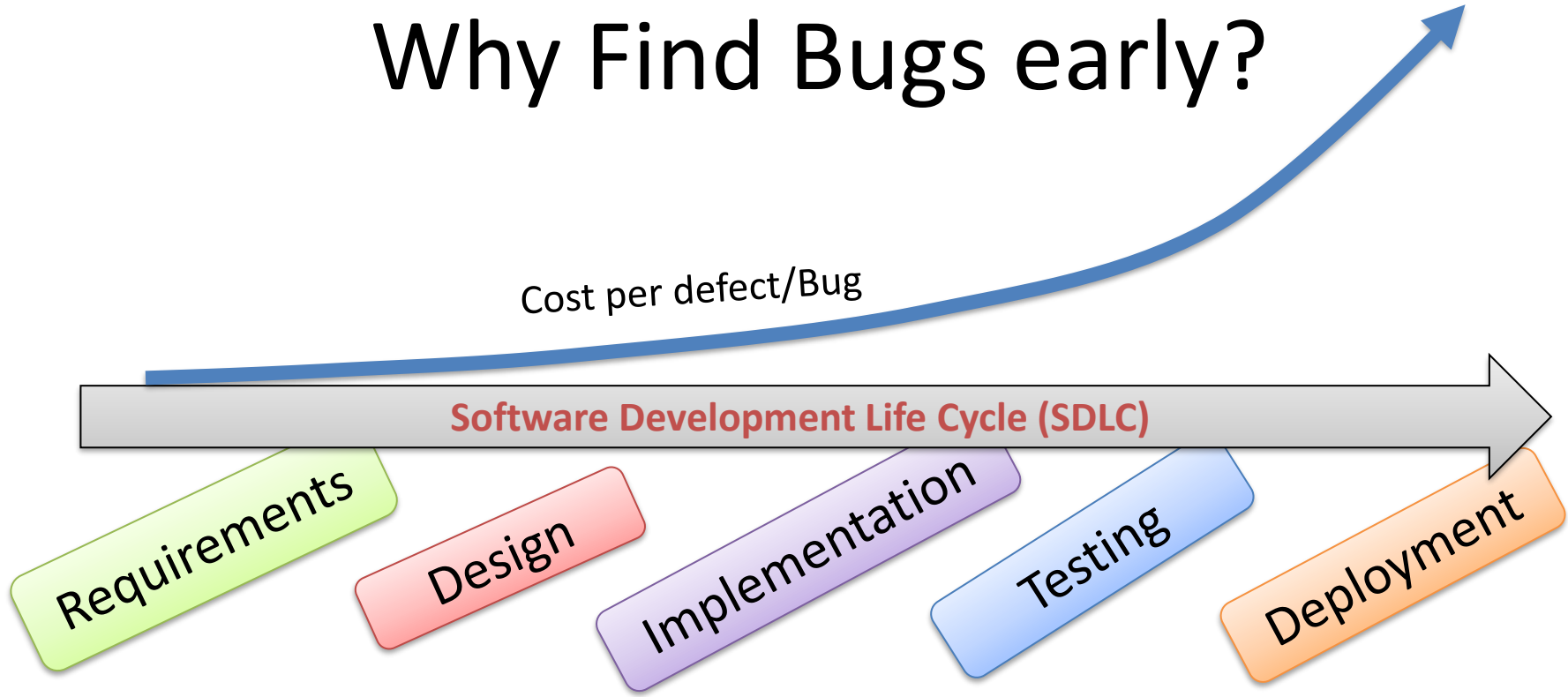


Introduction to Testing

Hans-Petter Halvorsen



Why Find Bugs early?



Testing

```
graph TD; Testing[Testing] --> Validation[Validation Testing]; Testing --> Defect[Defect 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

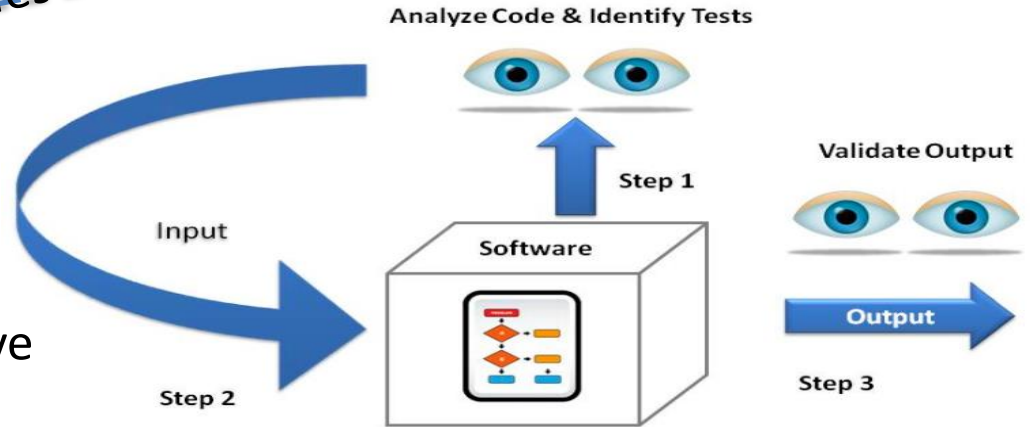
Black-box vs. White-box Testing

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



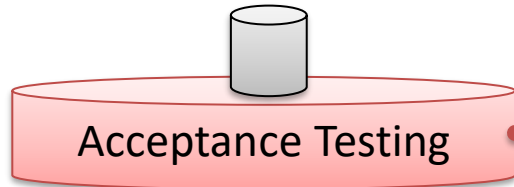
Grey-box Testing

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

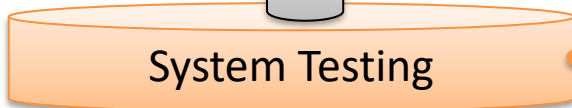


Typically done by Developers, etc

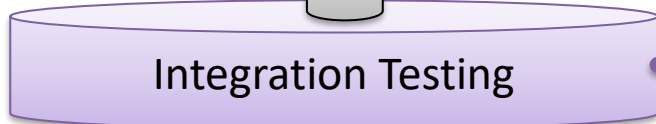
Levels of Testing



Is the responsibility of the customer – in general. The goal is to gain confidence in the system; especially in its non-functional characteristics



The behavior of the whole product (system) as defined by the scope of the project



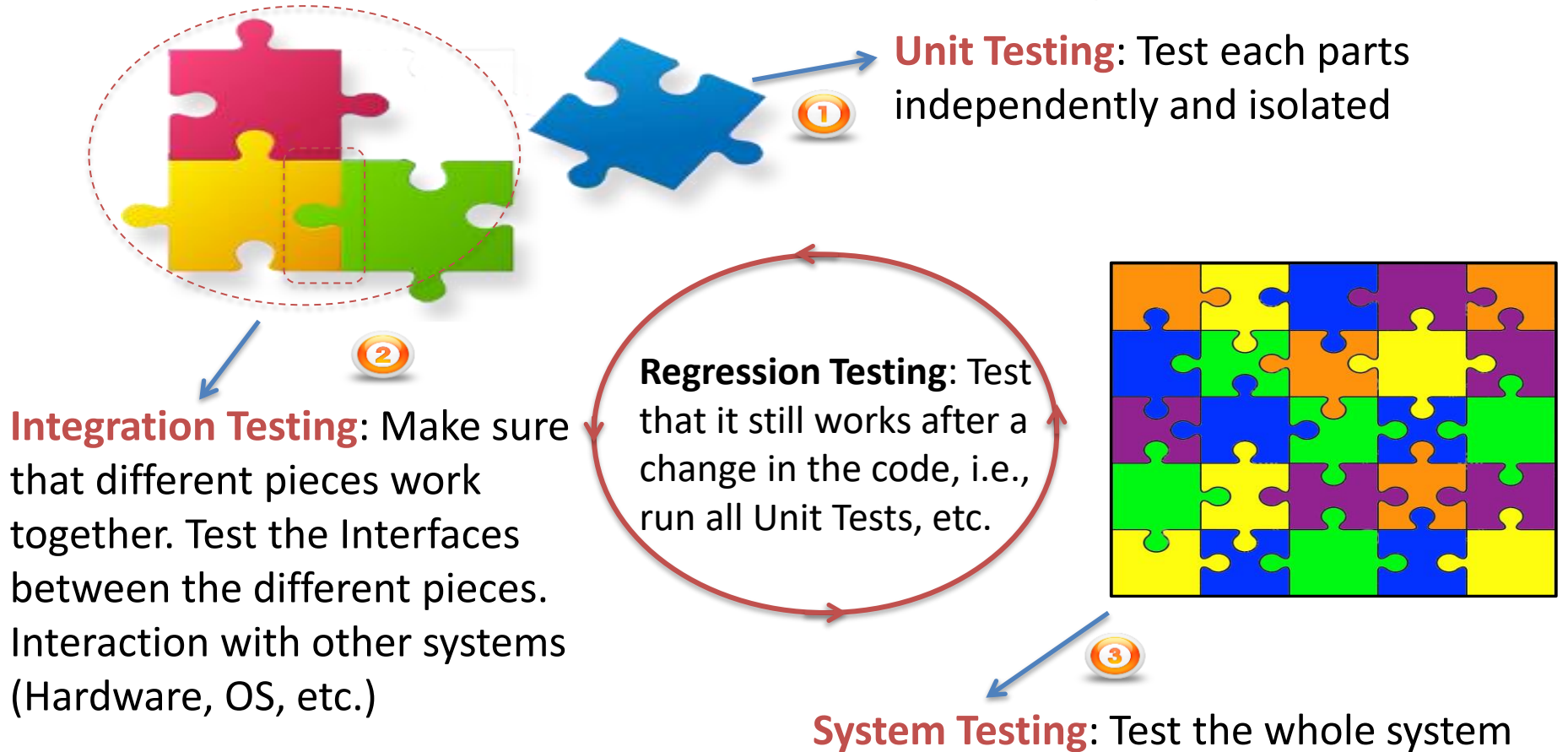
Interface between components; interactions with other systems (OS, HW, etc)

Unit Testing

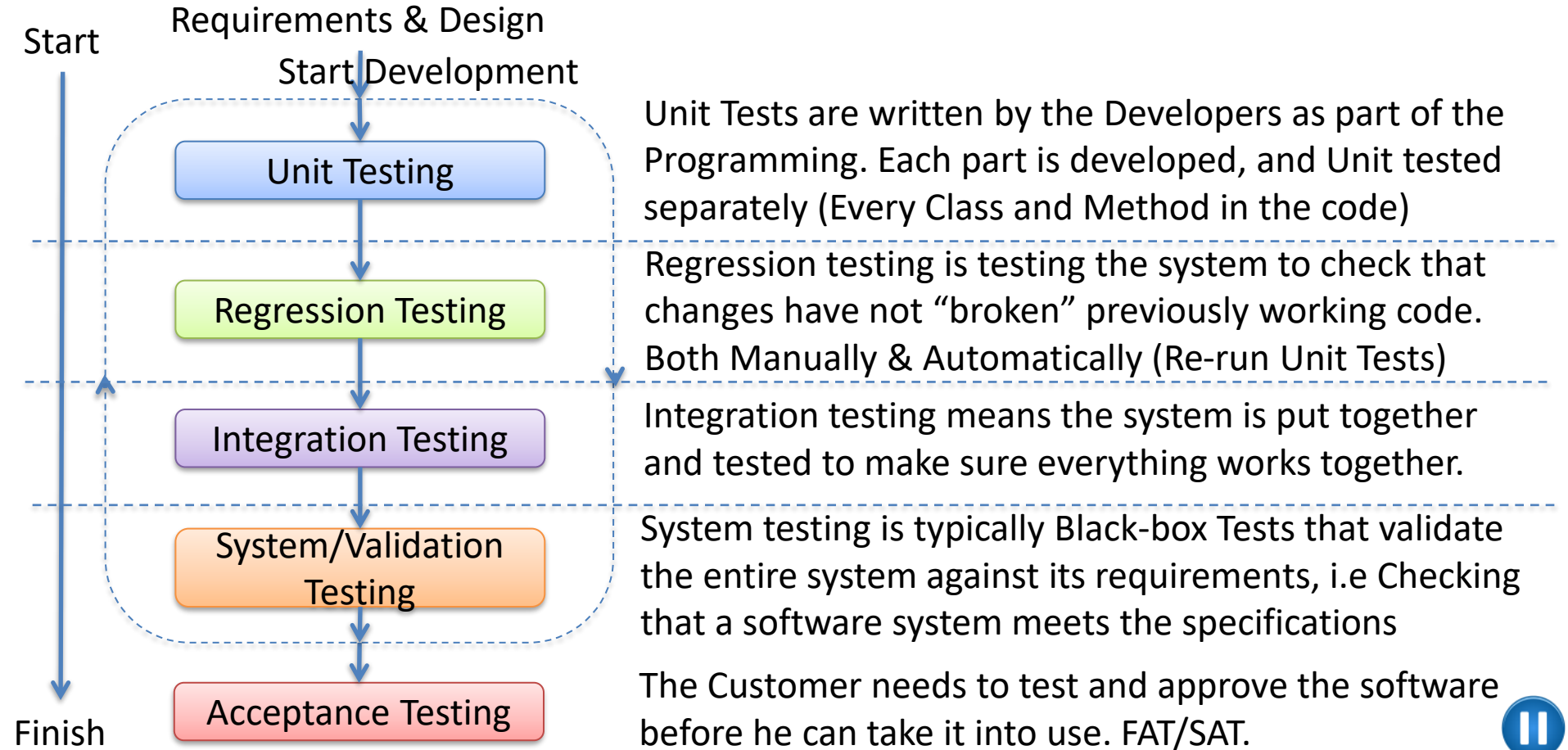
The diagram shows a stack of four cylindrical levels representing testing stages. The bottom level is a blue cylinder labeled 'Unit Testing'. It is connected to a descriptive text box on the right by a blue line.

Any module, program, object separately testable

Levels of Testing



Levels of Testing



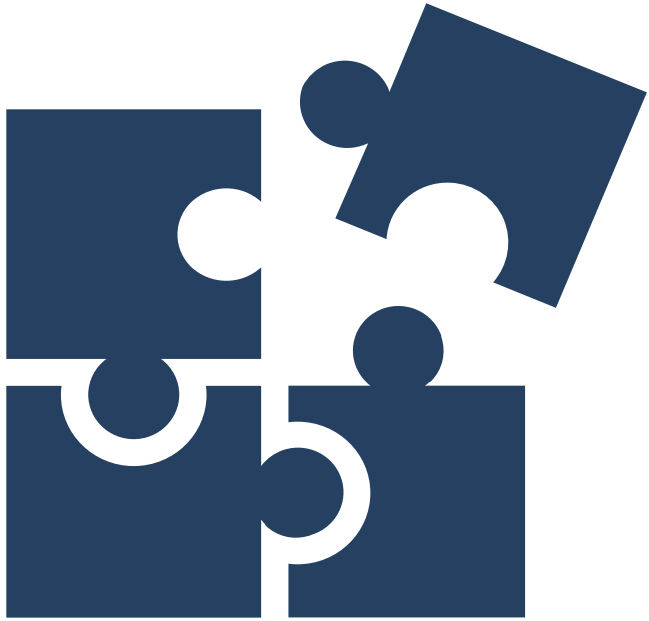


Unit Testing

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

Add a new project

Recent project templates

ASP.NET Core Web Application C#

Test C# ASP.NET Core 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)

Next

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

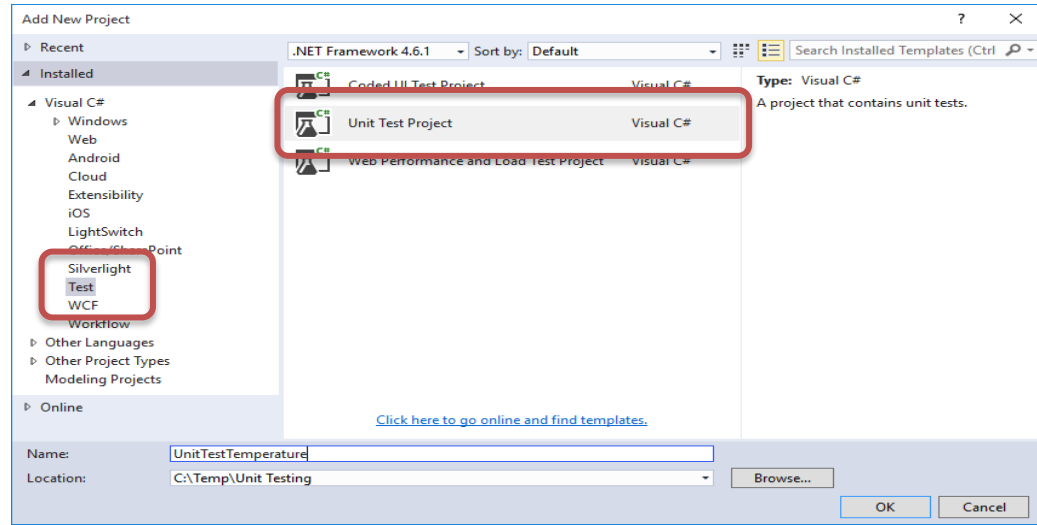


Unit Testing in Visual Studio

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:

Celsius: °C Fahrenheit: °F

Conversion Formula:

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

User Interface

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

Temperature Conversion

Temperature [Celsius]:

Temperature [Fahrenheit]:

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 displays the Visual Studio IDE with the following components:

- Menu Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q) with the text "FahrenheitApp".
- Toolbox:** SQL Server, Object Explorer.
- Code Editor:** Shows the content of `Temperature.cshtml`. The code includes a page directive, a model reference, and a form with two input fields and a submit button.

```
1 @page
2 @model FahrenheitApp.Pages.TemperatureModel
3 @{
4 }
5
6 <div>
7
8     <h1>Temperature Conversion</h1>
9
10    <form name="bookForm" id="bookForm" method="post">
11
12        Temperature [Celsius]:
13        <br />
14        <input name="TemperatureCelsius" type="number" class="form-control input-lg" autofocus required value="@Model.temperatureCelsius"/>
15        <br />
16
17        Temperature [Fahrenheit]:
18        <br />
19        <input name="TemperatureFahrenheit" type="number" class="form-control input-lg" value="@Model.temperatureFahrenheit" disabled/>
20        <br />
21
22        <input id="ConvertButton" type="submit" value="Convert" class="btn btn-info" />
23
24    </form>
25
26 </div>
```
- Solution Explorer:** Shows the project structure for "FahrenheitApp", including folders for Models, Pages, and Shared, and files like `Temperature.cs`, `Temperature.cshtml`, and `_Pages_Temperature`.
- Properties Window:** Currently empty.
- Status Bar:** Shows "No issues found" and "Ln: 1 Ch: 1 SPC CRLF".

Testing

FahrenheitApp Home Temperature Privacy

Temperature Conversion

Temperature [Celsius]:

Temperature [Fahrenheit]:

Convert

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

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

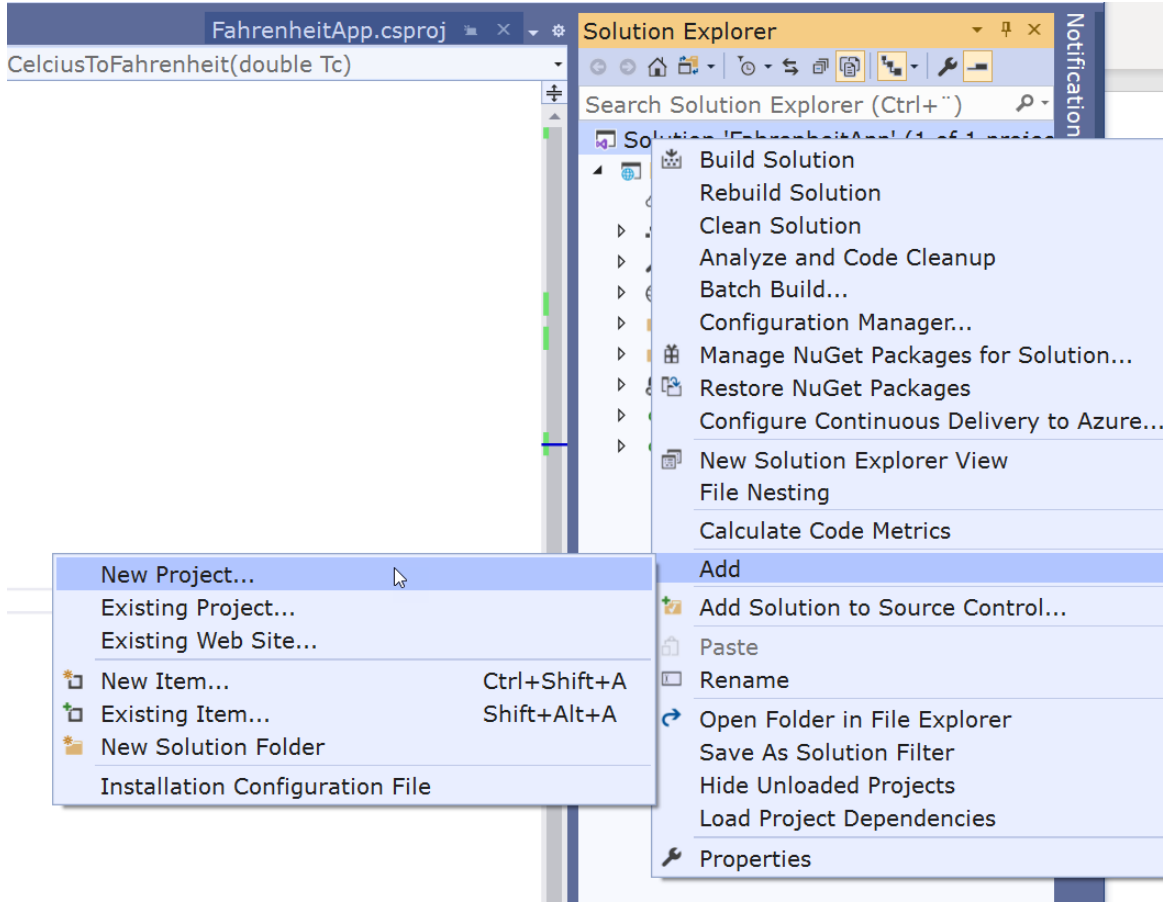
=71.6

© 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#

Test C# ASP.NET Core

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.

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

The screenshot displays the Visual Studio IDE with a solution named 'FahrenheitApp' containing two projects: 'FahrenheitApp' and 'UnitTestProject'. The 'Solution Explorer' on the right is circled in red, highlighting the project structure. The 'UnitTestProject' is selected, and its properties are shown in the 'Properties' window at the bottom right.

Solution Explorer Structure:

- Solution 'FahrenheitApp' (2 of 2 projects)
 - FahrenheitApp
 - Connected Services
 - Dependencies
 - Properties
 - wwwroot
 - Models
 - Pages
 - appsettings.json
 - Program.cs
 - Startup.cs
 - UnitTestProject
 - Dependencies
 - UnitTest1.cs

Code in UnitTest1.cs:

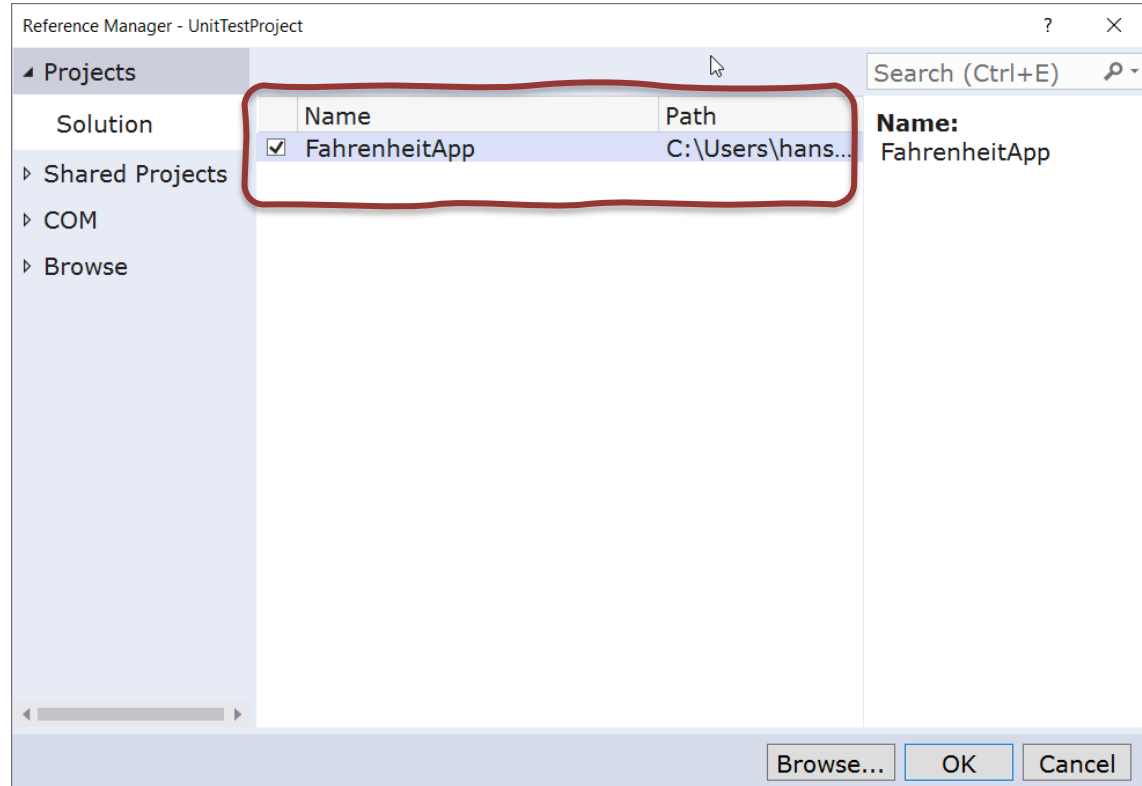
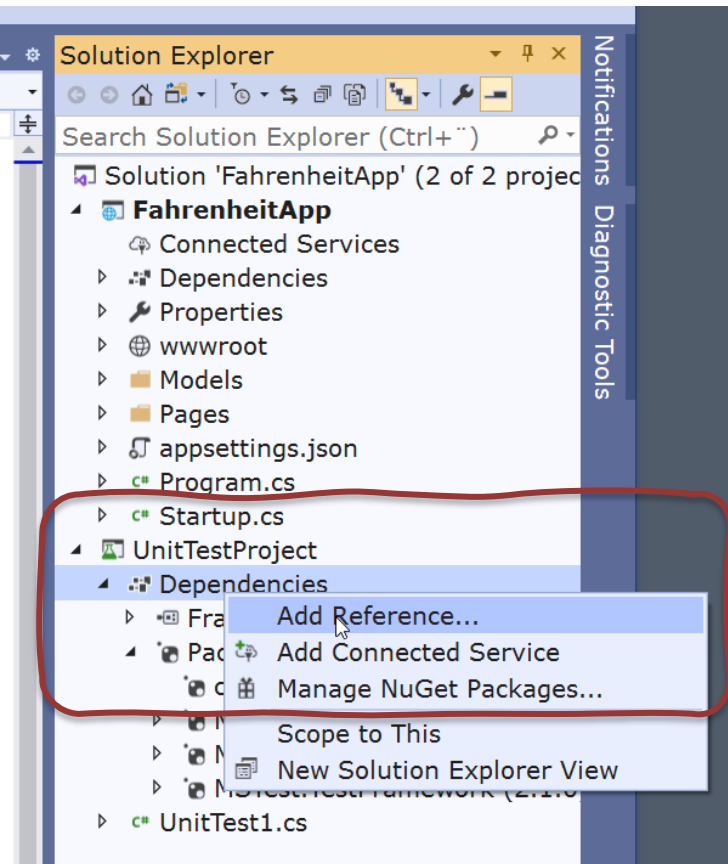
```
1 using Microsoft.VisualStudio.TestTools.UnitTesting;  
2  
3 namespace UnitTestProject  
4 {  
5     [TestClass]  
6     public class UnitTest1  
7     {  
8         [TestMethod]  
9         public void TestMethod1()  
10        {  
11        }  
12    }  
13 }  
14
```

Properties Window (UnitTestProject Project Properties):

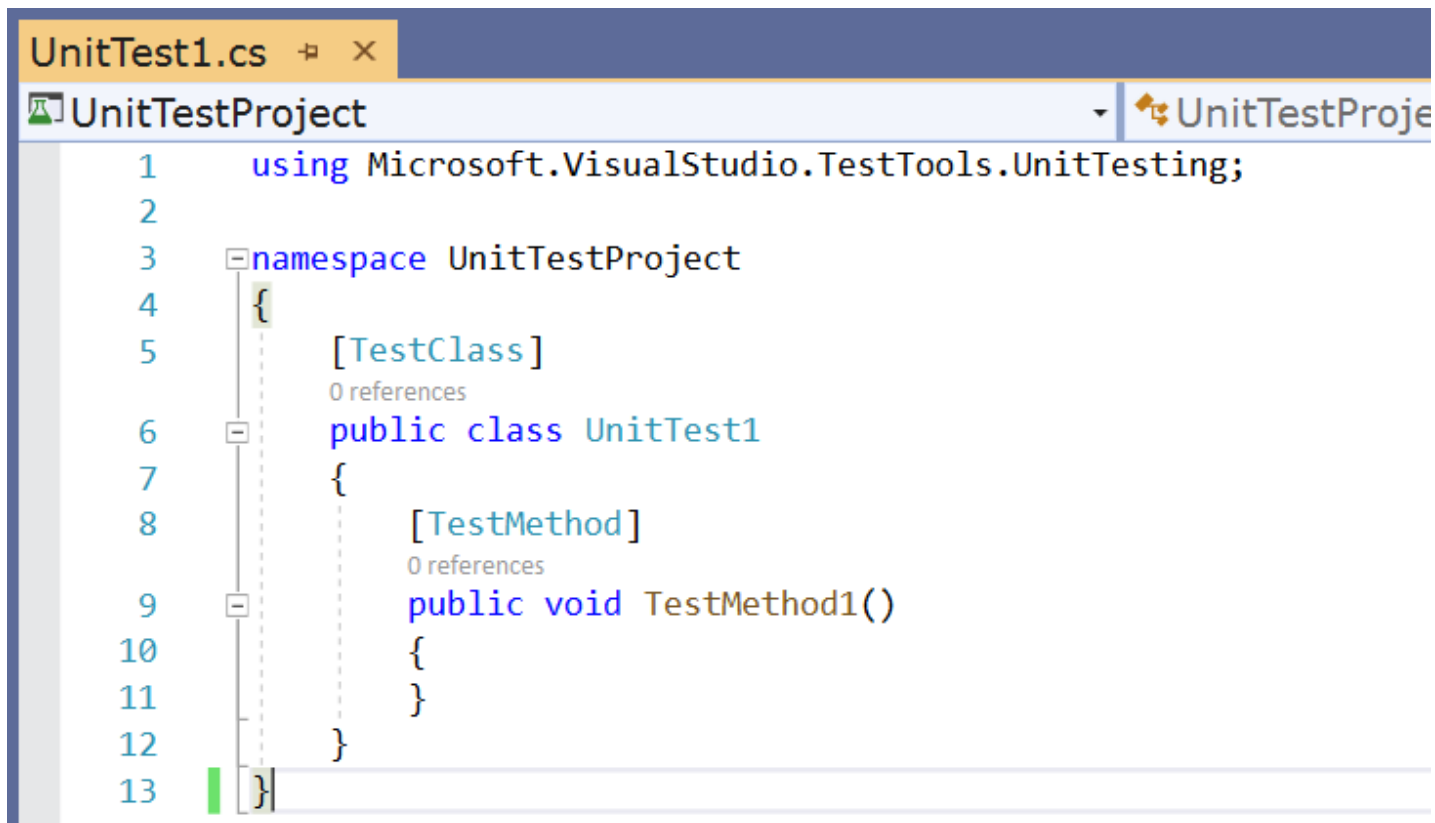
File Name	UnitTestProject.csproj
Full Path	C:\Users\hansp\OneD
Project Folder	C:\Users\hansp\OneD

File Name: Name of the project file.

Add Reference to the Code under Test



Create the 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 }
```

Create the Unit Test Code

The screenshot displays the Visual Studio IDE with the following components:

- Menu Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Toolbar:** Includes icons for file operations, search, and execution. The current configuration is 'Any CPU' and 'FahrenheitApp'.
- Code Editor:** Shows the file `UnitTestTemperature.cs` with the following code:

```
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 }
```
- Solution Explorer:** Shows the project structure for 'FahrenheitApp' and 'UnitTestProject'. The 'UnitTestProject' contains a 'Packages' folder with dependencies like 'coverlet.collector (1.2.0)', 'Microsoft.NET.Test.Sdk (16.5.0)', 'MSTest.TestAdapter (2.1.0)', and 'MSTest.TestFramework (2.1.0)'. The file `UnitTestTemperature.cs` is selected.
- Properties Window:** Currently empty.
- Status Bar:** Shows '100 %', 'No issues found', and 'Ln: 12 Ch: 10 SPC CRLF'.
- Bottom Bar:** Displays 'Build succeeded' and 'Add to Source Control'.

```
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;
```

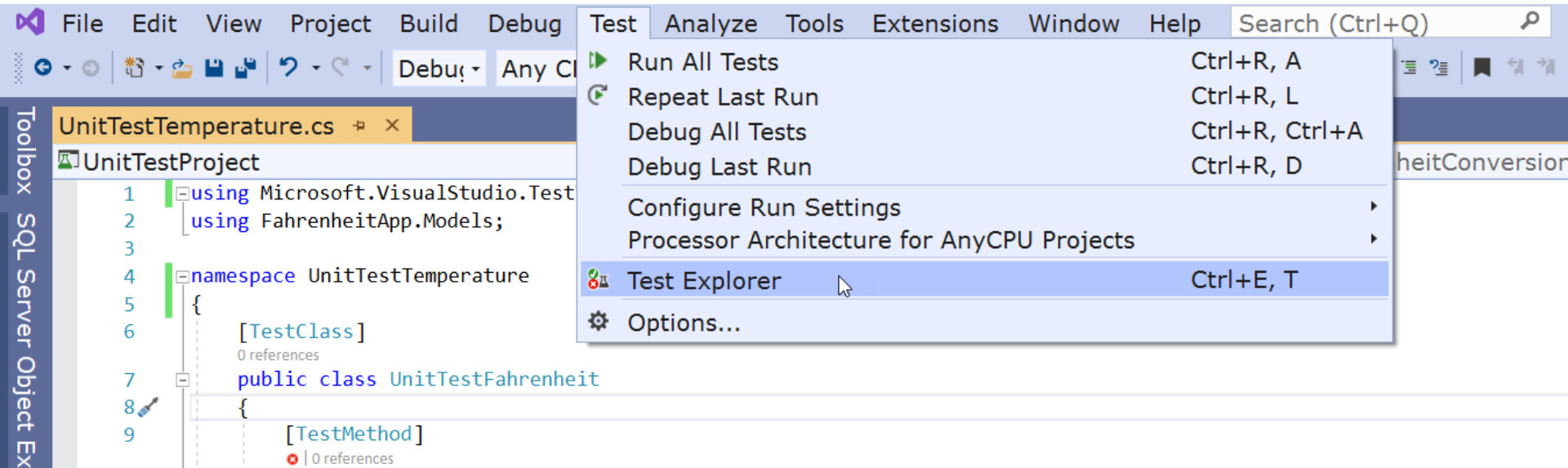
```
        temperatureFahrenheitActual = Temperature.CelciusToFahrenheit(temperatureCelcius);
```

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

```
}
```

```
}
```

Test Explorer



Start Running the Unit Test

The screenshot shows the Visual Studio IDE with the Test Explorer open. The test `TestFahrenheitConversion` in the `UnitTestFahrenheit` project has failed. The error message is: `Assert.AreEqual failed. Expected a difference no greater than <0.001> between expected value <71.6> and actual value <71.6>`. The stack trace points to `UnitTestFahrenheit.TestFahrenheitConversion()` at line 18.

Test Explorer

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

Test Detail Summary

- TestFahrenheitConversion
 - Source: [UnitTestTemperature.cs](#) line 10
 - Duration: 97 ms
 - Message:
Assert.AreEqual failed. Expected a difference no greater than <0.001> between expected value <71.6> and actual value <71.6>
 - Stack Trace:
[UnitTestFahrenheit.TestFahrenheitConversion\(\)](#) line 18

Solution Explorer

- 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

Ready

Add to Source Control

Test Results

The screenshot displays the Visual Studio Test Explorer interface. The top bar shows the file name 'UnitTestTemperature.cs' and a search bar. Below the toolbar, the 'Test' pane on the left shows a tree view with the following structure:

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

The 'TestFahrenheitConversion' test is selected and highlighted in blue. The 'Test Detail Summary' pane on the right provides the following information:

- TestFahrenheitConversion** (failed)
 - Source: [UnitTestTemperature.cs](#) line 10
 - Duration: 25 ms
 - Message: Assert.AreEqual failed. Expected a difference no greater than <0.001> between expected value <71.6> and actual value <71.6>
 - Stack Trace:
 - [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

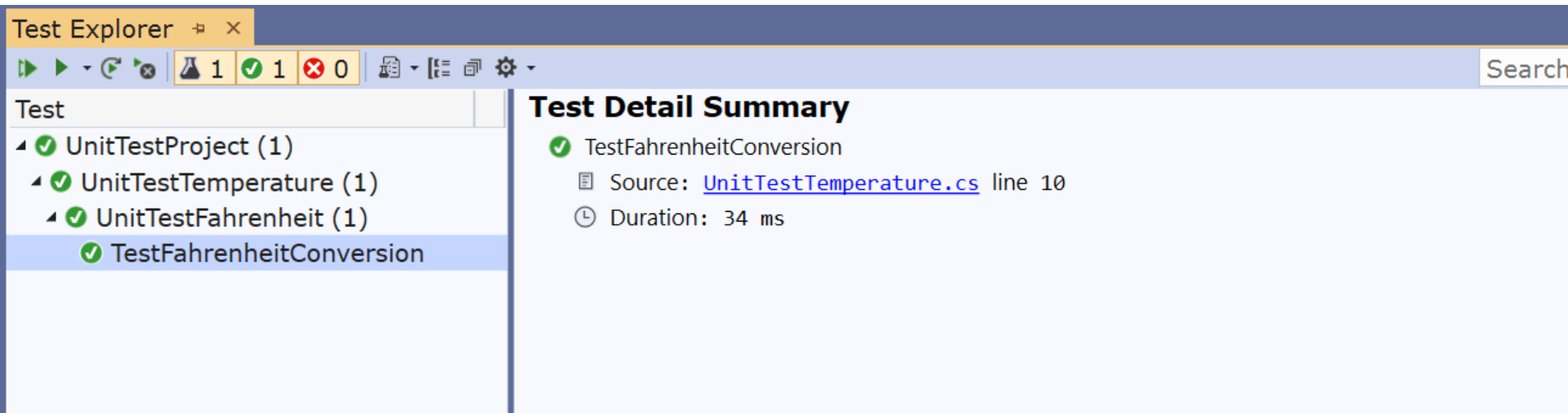
$$T_F = \frac{9}{5} T_C + 32$$

```
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 displays the Visual Studio Test Explorer interface. At the top, the title bar reads "Test Explorer". Below it, a toolbar contains icons for running tests, with a summary showing 1 passed test (green checkmark), 1 failed test (red X), and 0 skipped tests (red X). The main area is split into two panes. The left pane, titled "Test", shows a tree view of test items: "UnitTestProject (1)", "UnitTestTemperature (1)", "UnitTestFahrenheit (1)", and "TestFahrenheitConversion". The "TestFahrenheitConversion" item is selected and highlighted in blue. The right pane, titled "Test Detail Summary", provides information for the selected test: a green checkmark icon, the test name "TestFahrenheitConversion", the source file "[UnitTestTemperature.cs](#) line 10", and the execution duration "Duration: 34 ms".

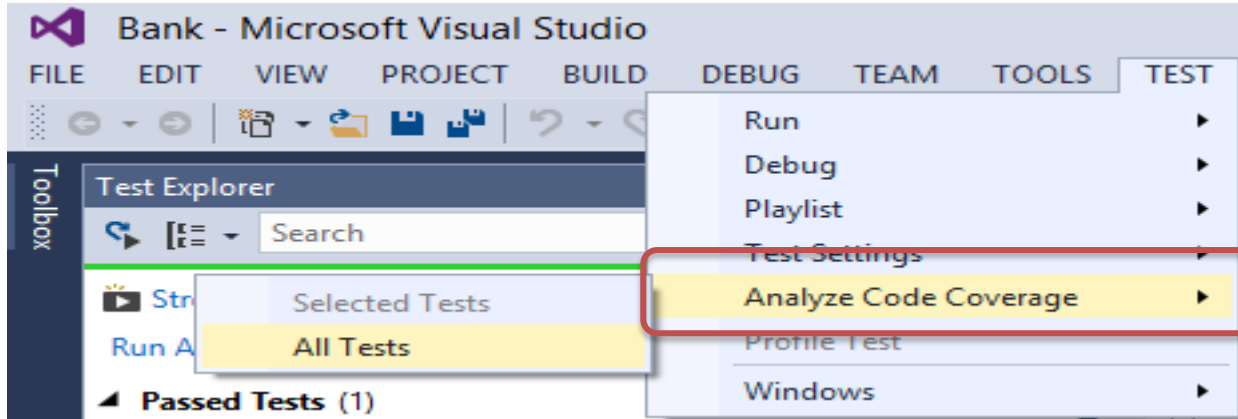
Everything Works! The Test Passed!

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 %
unitteststtemperature.dll	0	0,00 %	4	100,00 %

In this case the Unit Test covered 100% of the code. If we use If...Else... or similar, 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>

