Unit Testing

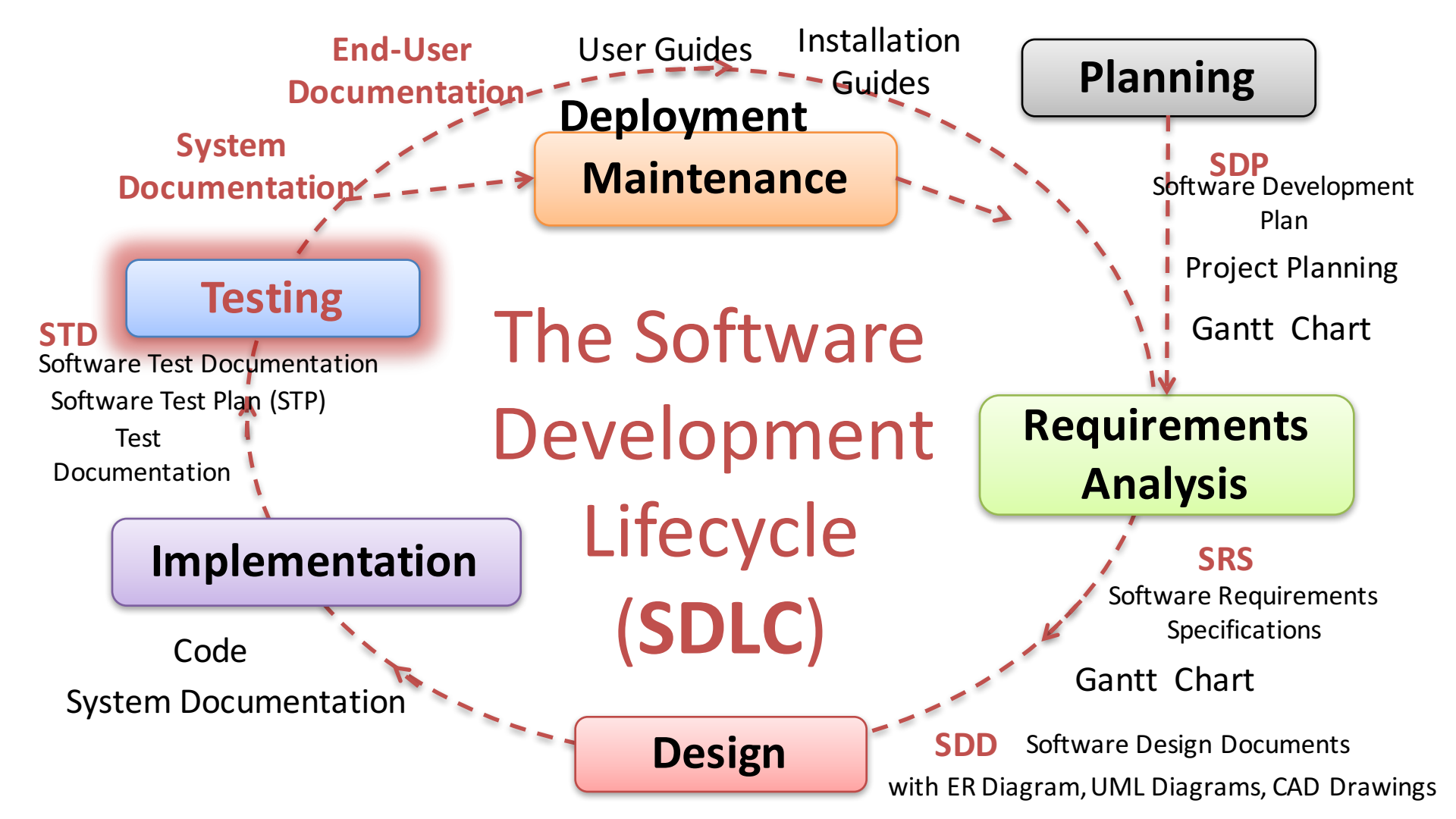
Hans-Petter Halvorsen, M.Sc.

Contents

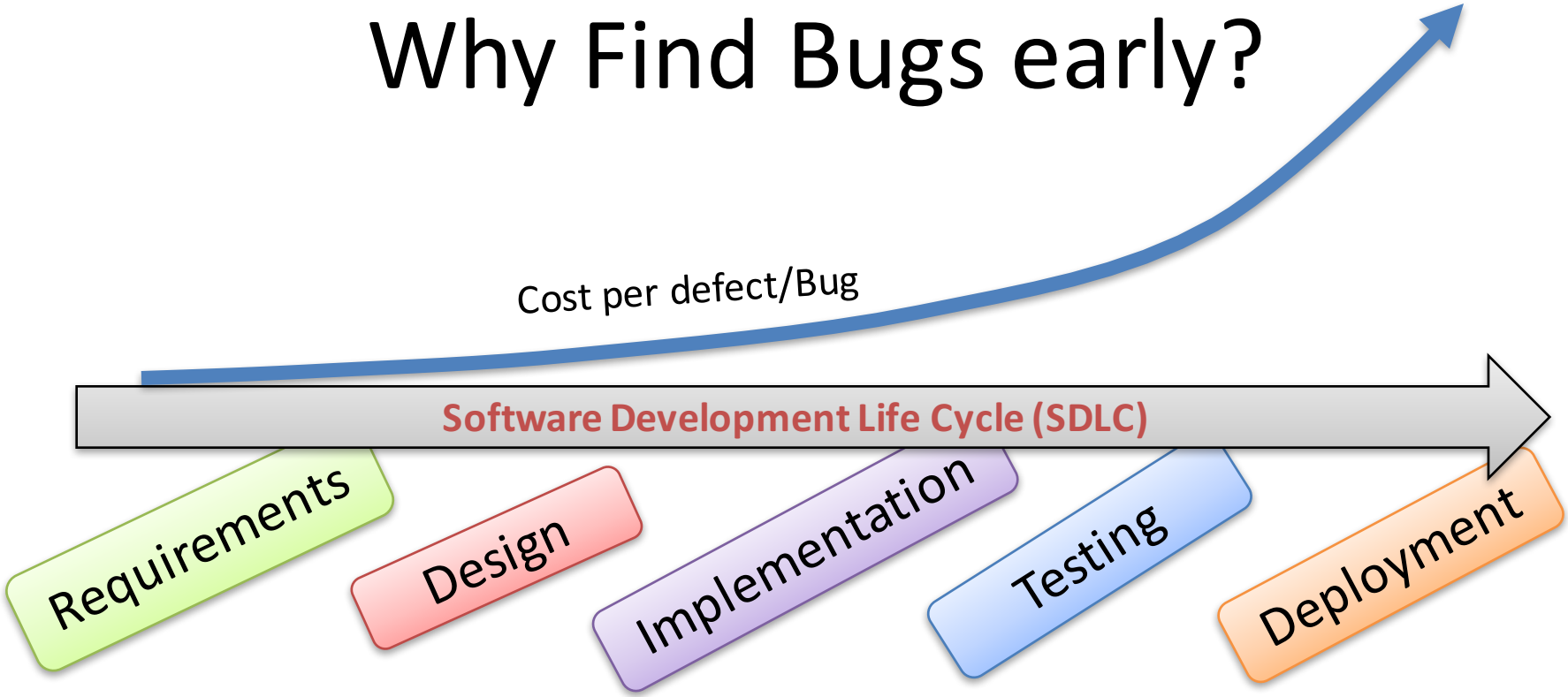
1. What is Testing?
   - Short Introduction to Testing
2. What is Unit Testing?
3. Unit Testing in Visual Studio

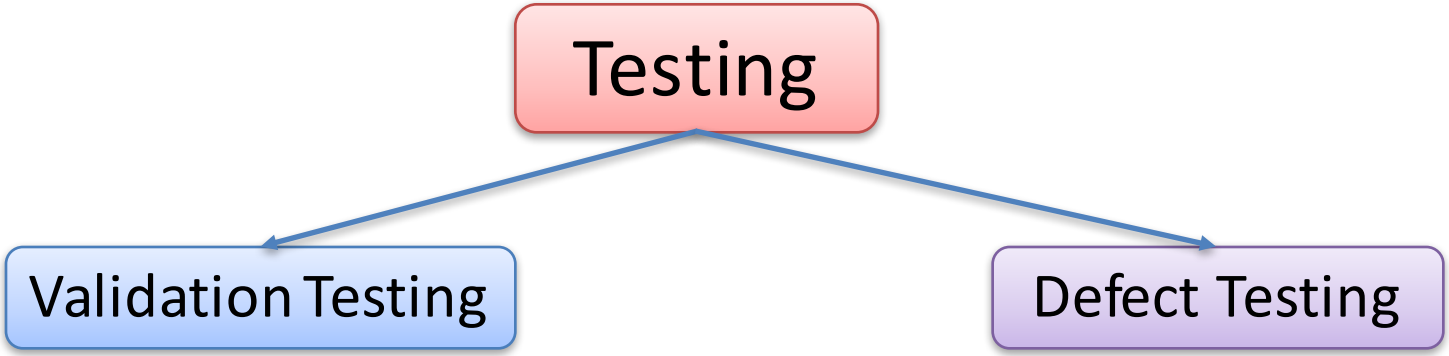
Introduction to Testing

Hans-Petter Halvorsen, M.Sc.


End-User Documentation
User Guides
Installation Guides
Planning
Deployment
System Documentation
Maintenance
SDP
Software Development Plan
Project Planning
Gantt Chart
Testing
STD
Software Test Documentation
Software Test Plan (STP)
Test Documentation
The Software Development Lifecycle (SDLC)
Requirements Analysis
SRS
Software Requirements Specifications
Gantt Chart
Implementation
Code
System Documentation
Design
SDD
Software Design Documents
with ER Diagram, UML Diagrams, CAD Drawings

Why Find Bugs early?





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.

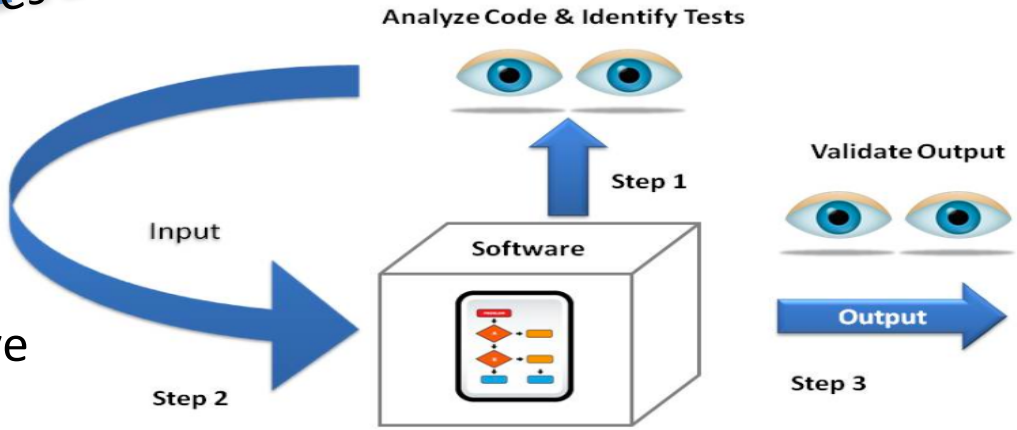
Find inputs or input sequences where the behavior of the software is incorrect, undesirable, or does not conform to its specifications.
These are caused by defects (bugs) in the software.

I. Sommerville, *Software Engineering*, 10 ed.: Pearson, 2015.

Test Categories
Black-box vs. White-box Testing

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

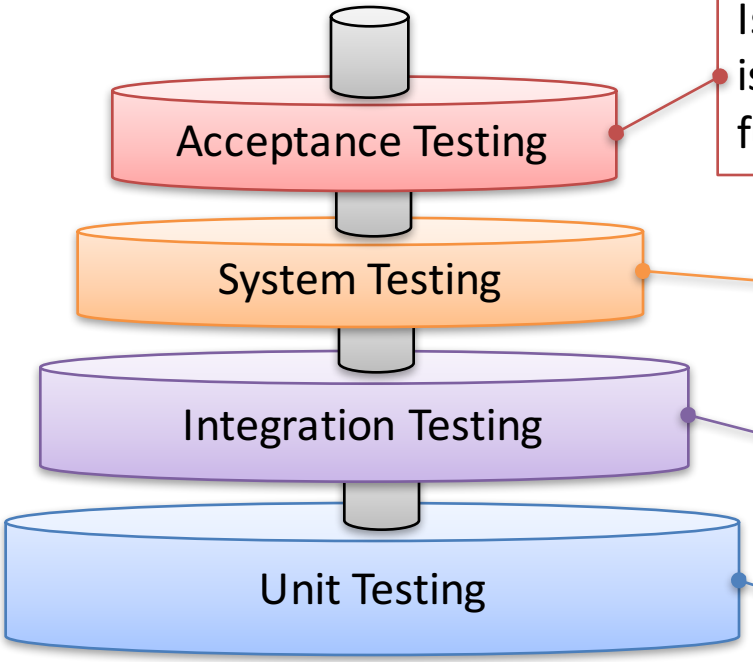
Grey-box Testing



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)

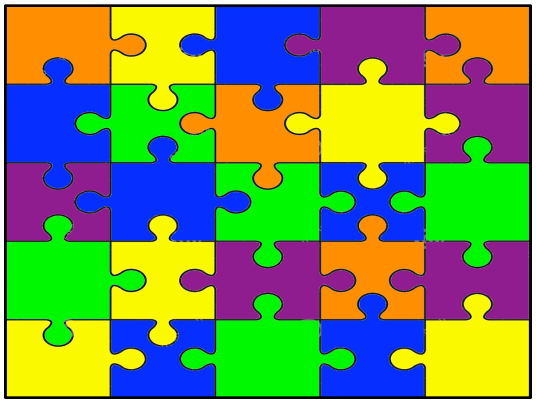
Any module, program, object separately testable

Levels of Testing

Unit Testing: Test each parts independently and isolated

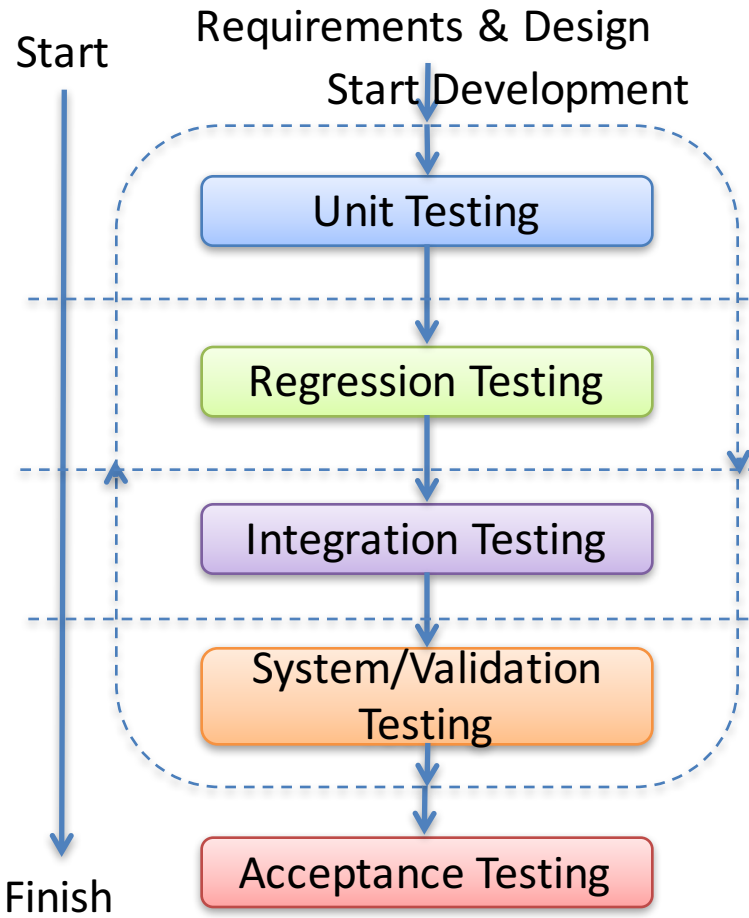
Integration Testing: Make sure that different pieces work together. Test the Interfaces between the different pieces. Interaction with other systems (Hardware, OS, etc.)

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 Tests are written by the Developers as part of the Programming. Each part is developed and Unit tested separately (Every Class and Method in the code)

Regression testing is testing the system to check that changes have not "broken" previously working code. Both Manually & Automatically (Re-run Unit Tests)

Integration testing means the system is put together and tested to make sure everything works together.

System testing is typically Black-box Tests that validate the entire system against its requirements, i.e Checking that a software system meets the specifications

The Customer needs to test and approve the software before he can take it into use. FAT/SAT.

Unit Testing

Hans-Petter Halvorsen, M.Sc.

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 eXreme Programming (XP) (an Agile method)
- **Unit Tests are important part of Software Development today – either you are using TDD or not**

Unit Tests Frameworks

Unit Tests Framework are usually integrated with the IDE

- **Visual Studio Unit Test Framework**. Unit Tests are built into Visual Studio (no additional installation needed)

Others:

- **JUnit** (Java)
  - JUnit is a unit testing framework for the Java programming language.
- **NUnit** (.NET)
  - NUnit is an open source unit testing framework for Microsoft .NET. It serves the same purpose as JUnit does in the Java world
- **PHPUnit** (PHP)
- LabVIEW Unit Test Framework Toolkit
- etc.

All of them work in the same manner – but we will use the Visual Studio Unit Test Framework

http://en.wikipedia.org/wiki/Visual_Studio_Unit_Testing_Framework

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”)

Example:

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

```
...

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 depend 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 are allowed to Check-in your Code in the Source Code Control System

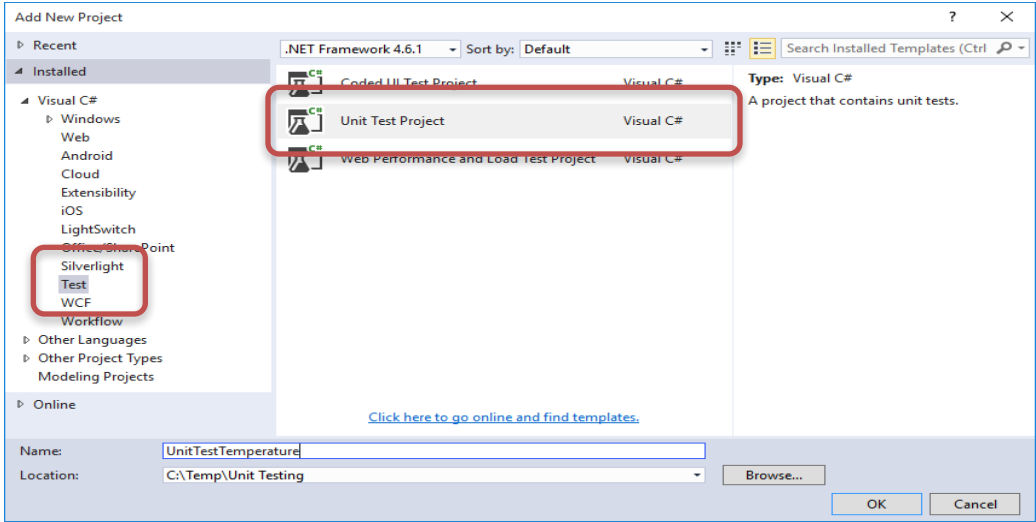
http://www.uio.no/studier/emner/matnat/ifi/INF5530

Unit Testing in Visual Studio

Hans-Petter Halvorsen, M.Sc.

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

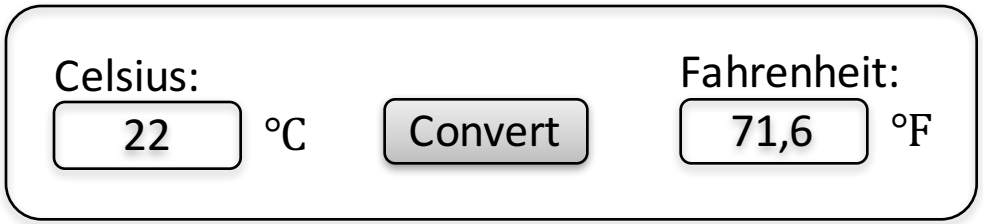
Unit Testing in Visual Studio

Hans-Petter Halvorsen, M.Sc.

Convert to Fahrenheit

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

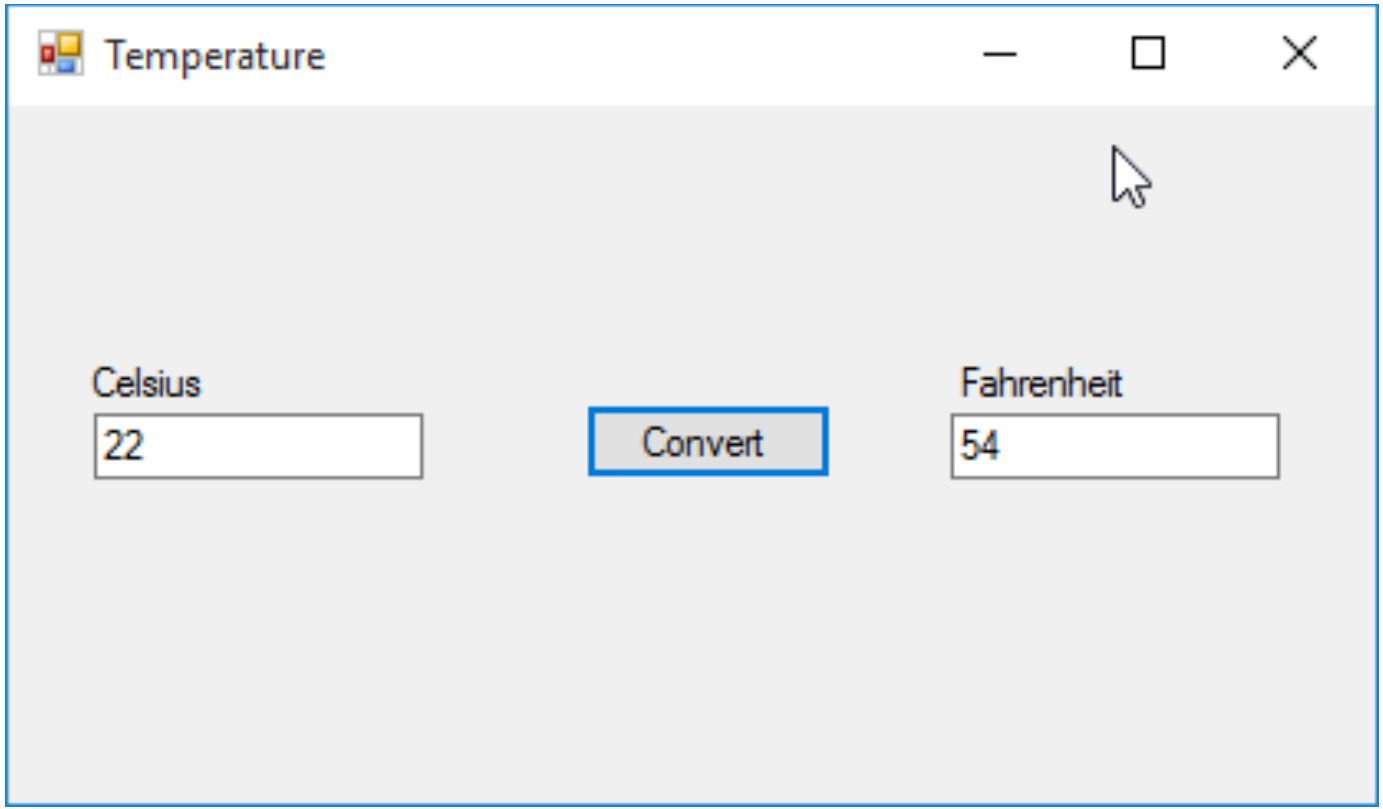
A simple sketch of the User Interface:



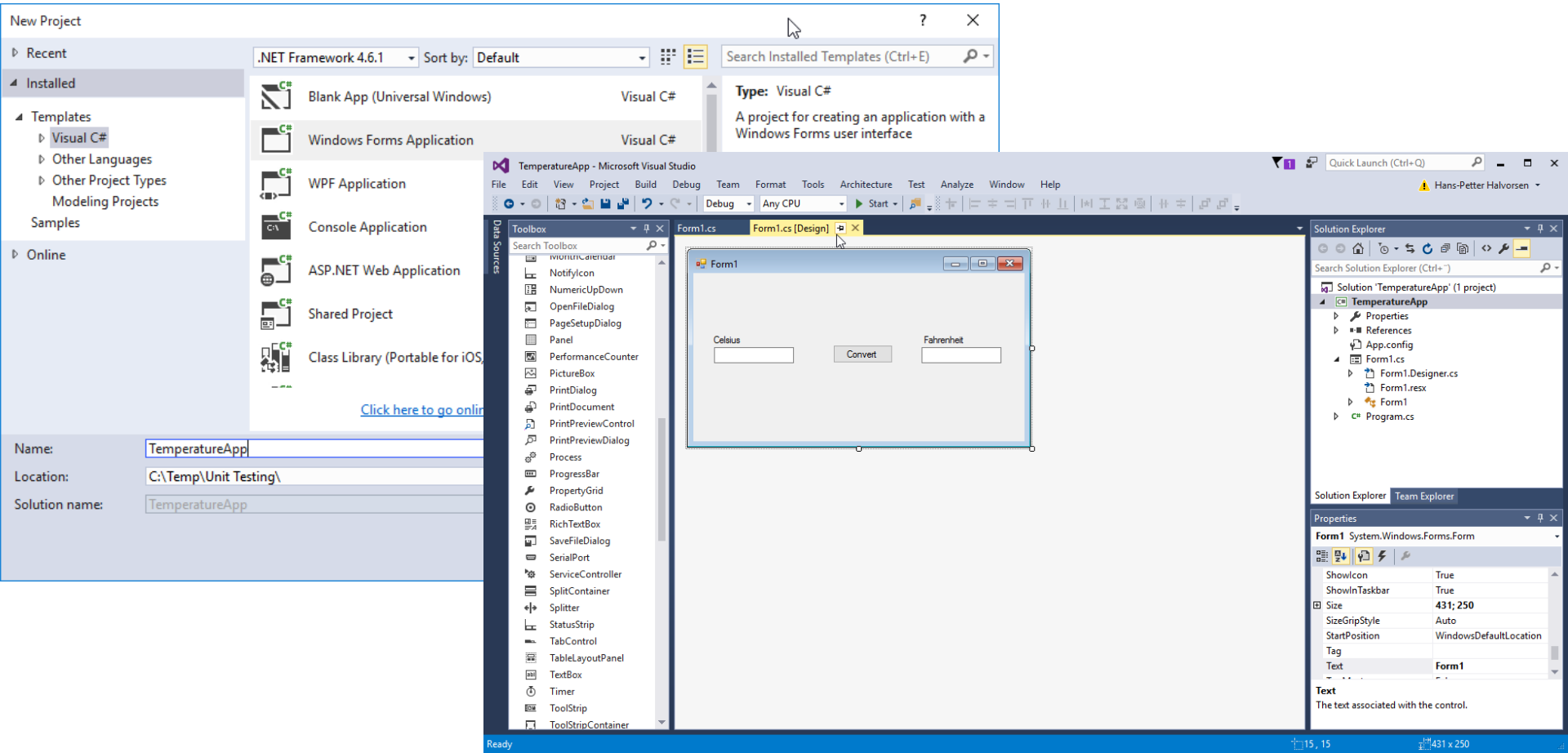
Conversion Formula:

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

User Interface

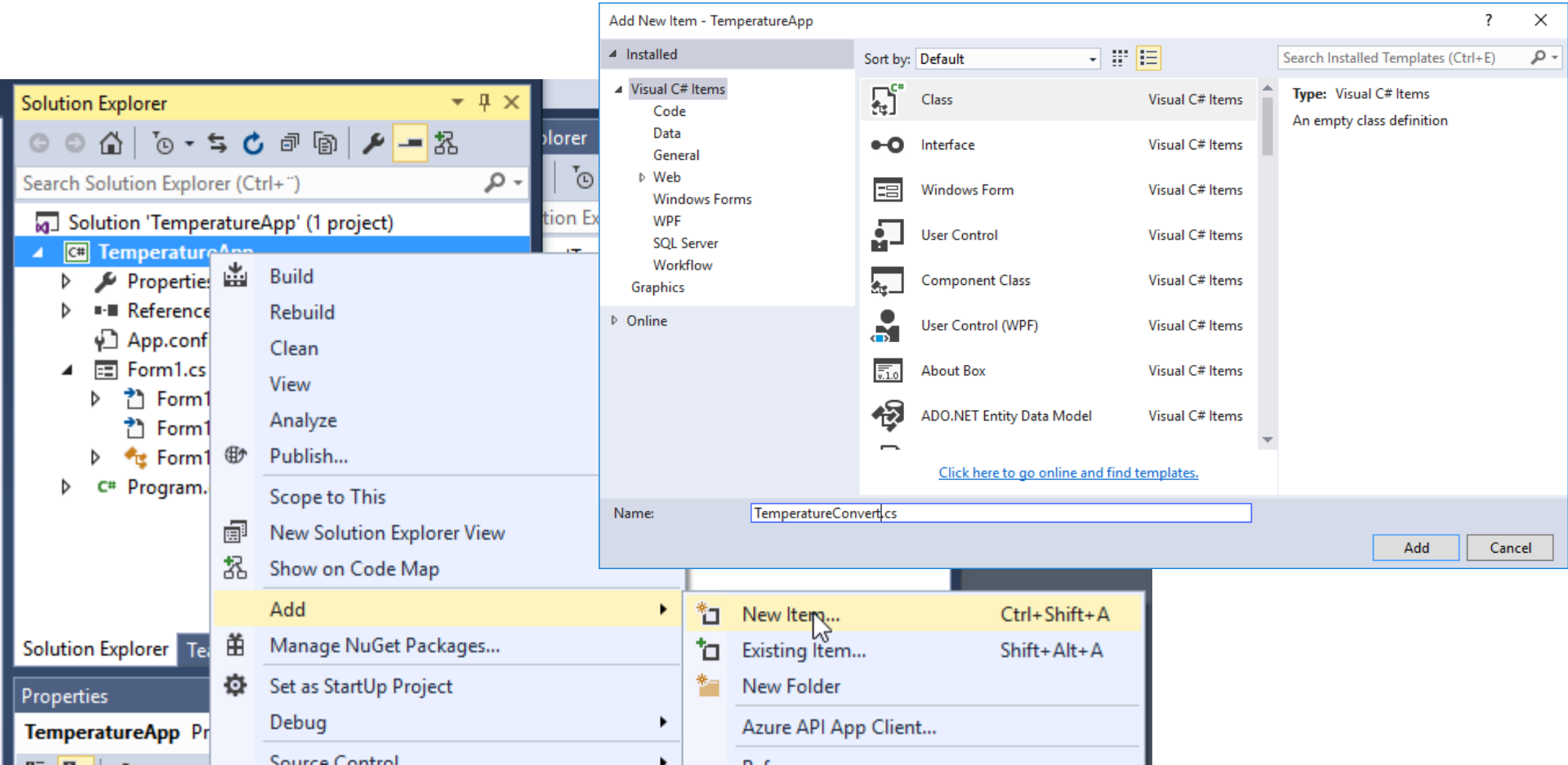


Create your GUI

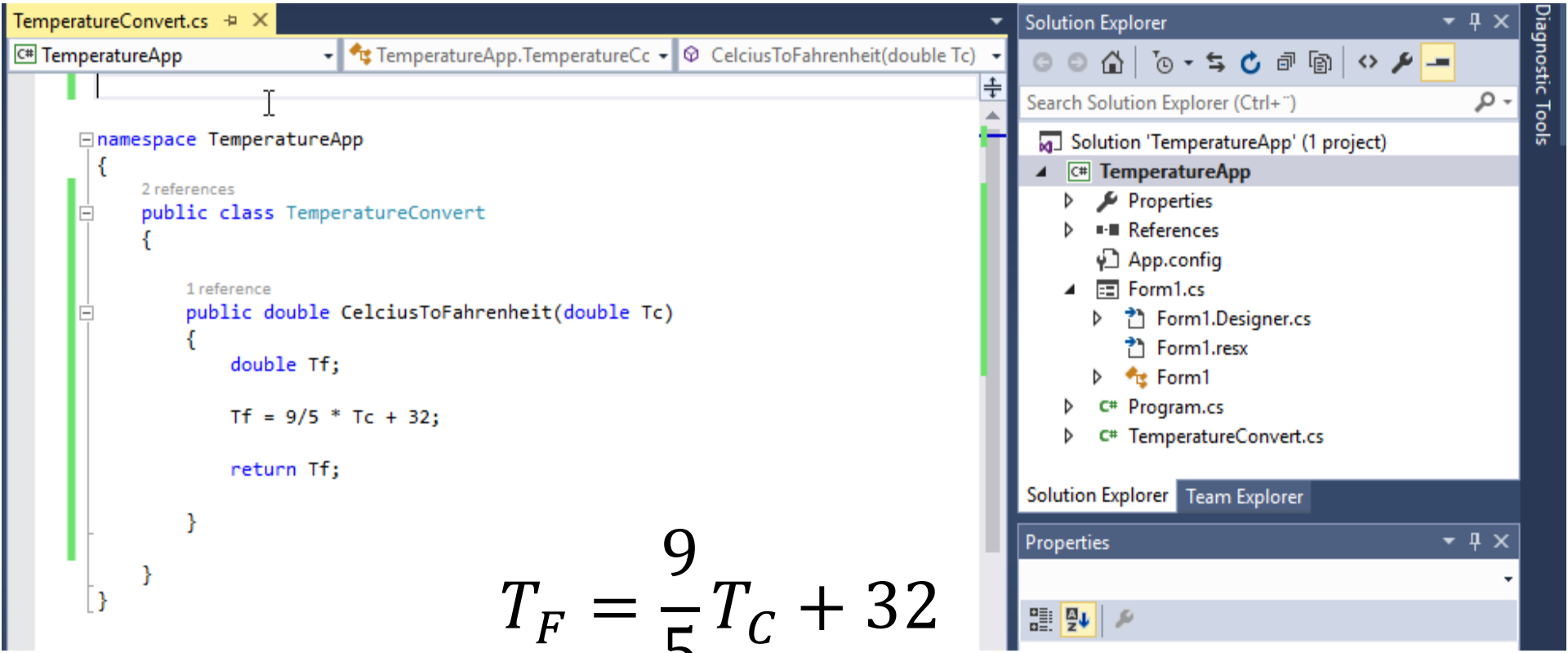



DEMO

Add Class


Add New Item - TemperatureApp
Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
SQL Server
Workflow
Graphics
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class
Visual C# Items
Interface
Windows Form
User Control
Component Class
User Control (WPF)
About Box
ADO.NET Entity Data Model
Type: Visual C# Items
An empty class definition
Click here to go online and find templates.
Name: TemperatureConvert.cs
Add
Cancel
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'TemperatureApp' (1 project)
Properties
References
App.config
Form1.cs
Program.cs
Build
Rebuild
Clean
View
Analyze
Publish...
Scope to This
New Solution Explorer View
Show on Code Map
Add
Manage NuGet Packages...
Set as StartUp Project
Debug
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Folder
Azure API App Client...
Properties
TemperatureApp

Class

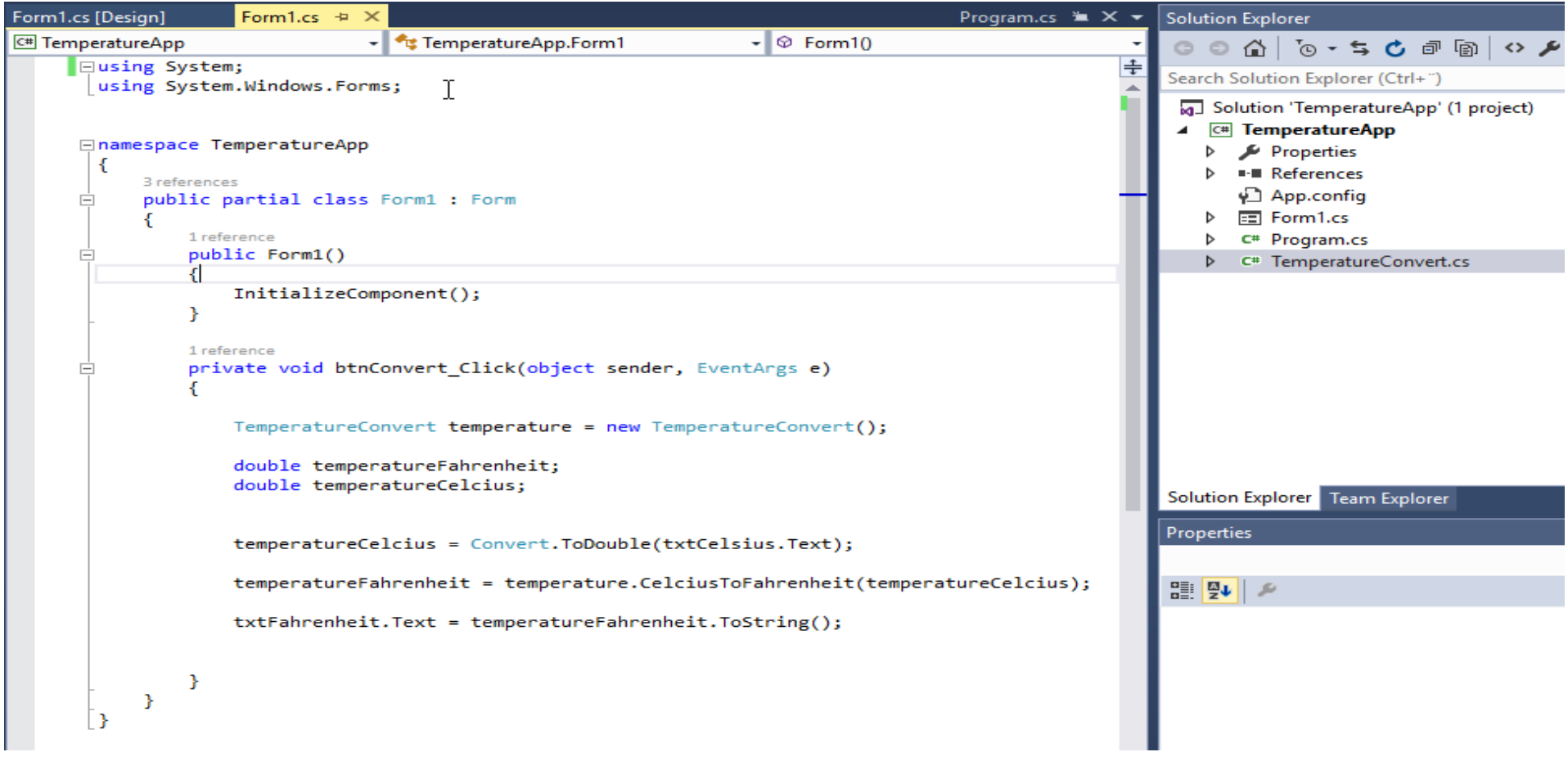


$$T_F = \frac{9}{5} T_C + 32$$

```
public class TemperatureConvert
{

        public double CelciusToFahrenheit(double Tc)
        {
            double Tf;

            Tf = 9/5 * Tc + 32;

            return Tf;
        }

}
```


DEMO

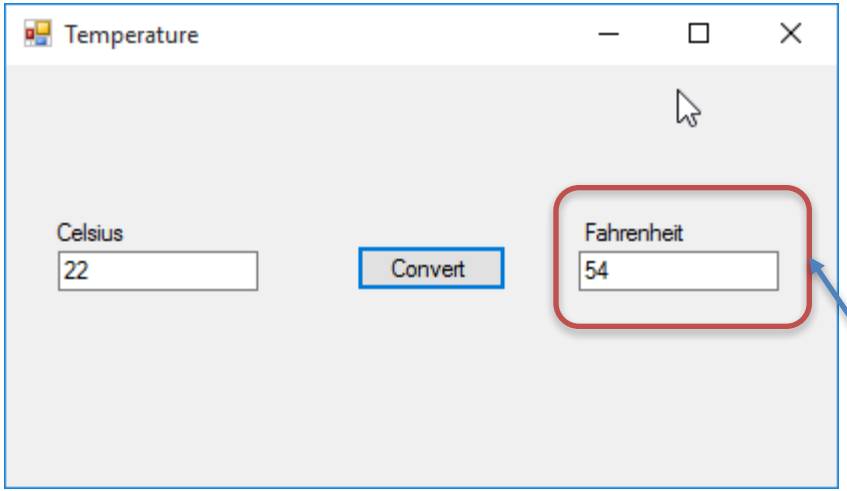
Create your Code



```
using System;
using System.Windows.Forms;

namespace TemperatureApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnConvert_Click(object sender, EventArgs e)
        {

            TemperatureConvert temperature = new TemperatureConvert();

            double temperatureFahrenheit;
            double temperatureCelcius;

            temperatureCelcius = Convert.ToDouble(txtCelsius.Text);

            temperatureFahrenheit = temperature.CelciusToFahrenheit(temperatureCelcius);

            txtFahrenheit.Text = temperatureFahrenheit.ToString();

        }
    }
}
```


DEMO

Test Application



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

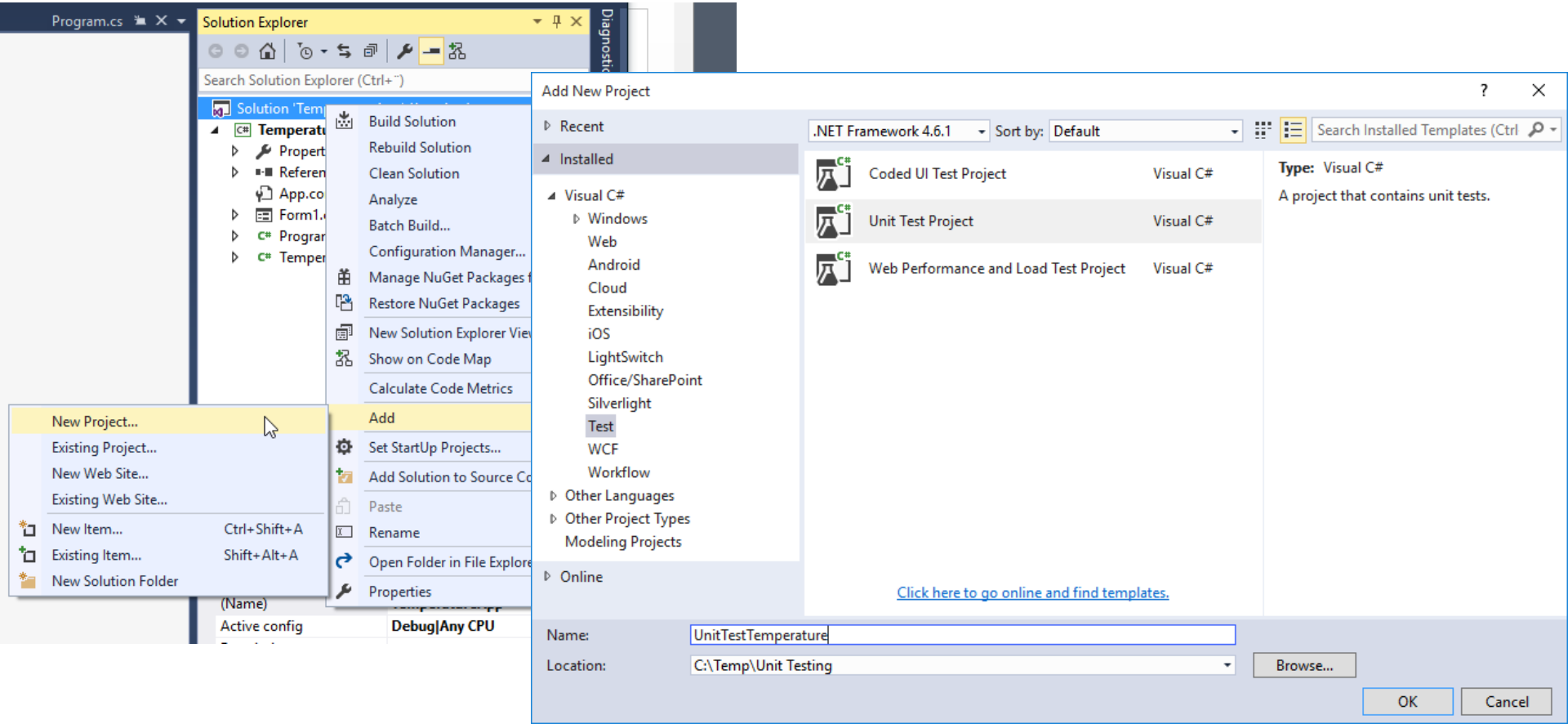
$$T_F = \frac{9}{5} \cdot 22 + 32$$

=71.6

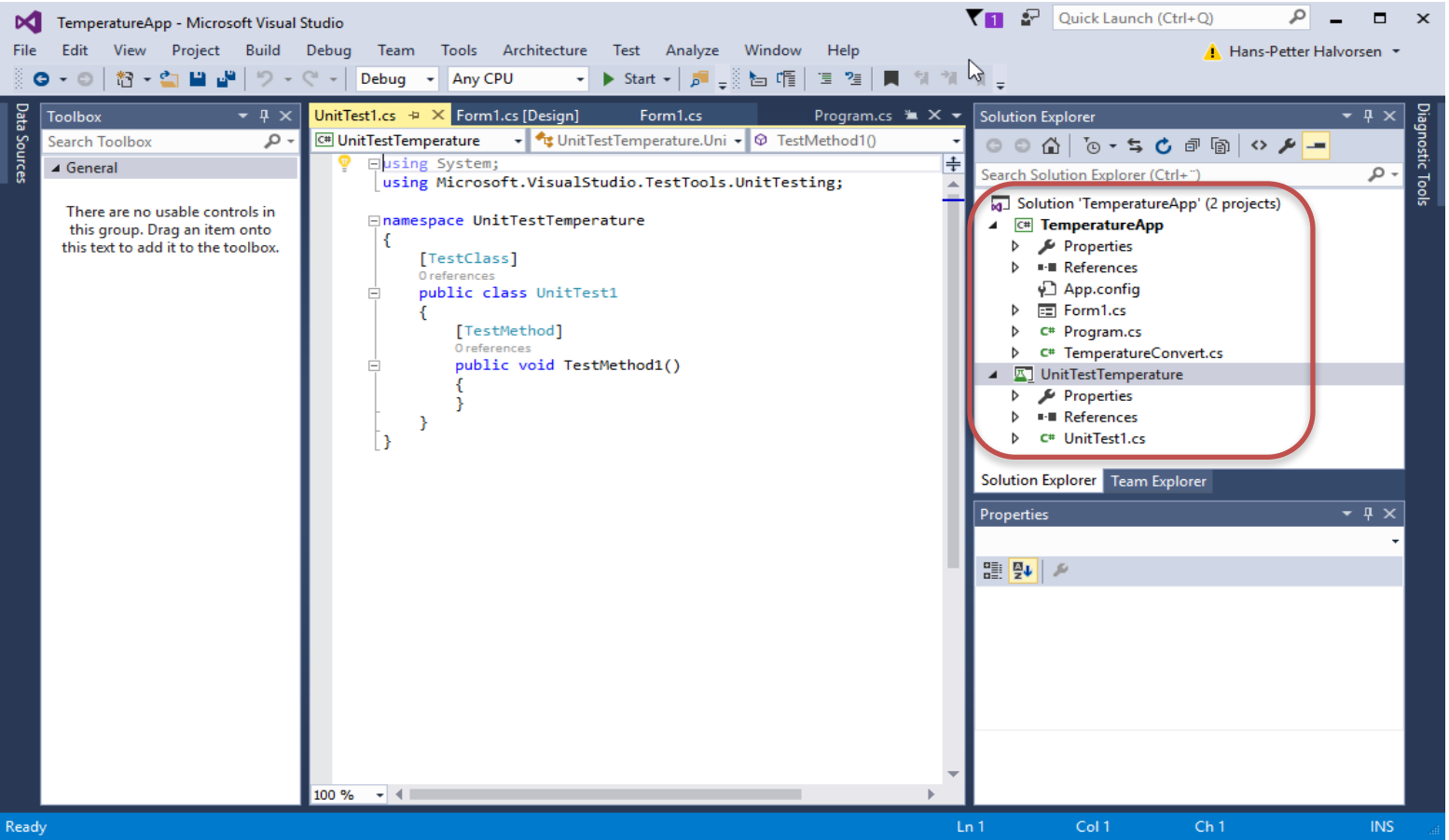
We get wrong Answer!


DEMO

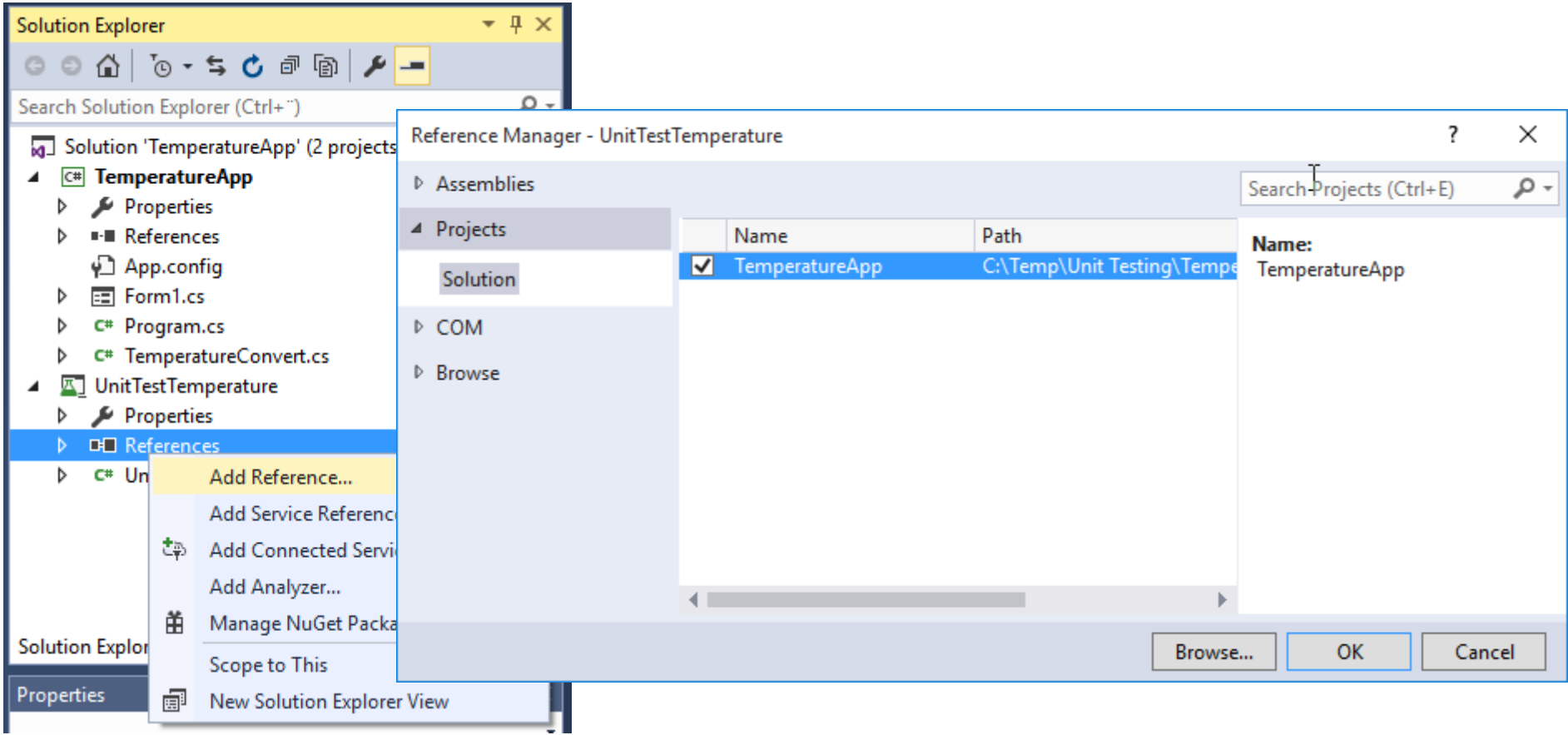
Create Unit Test Project



You have now 2 Projects in your Solution Explorer

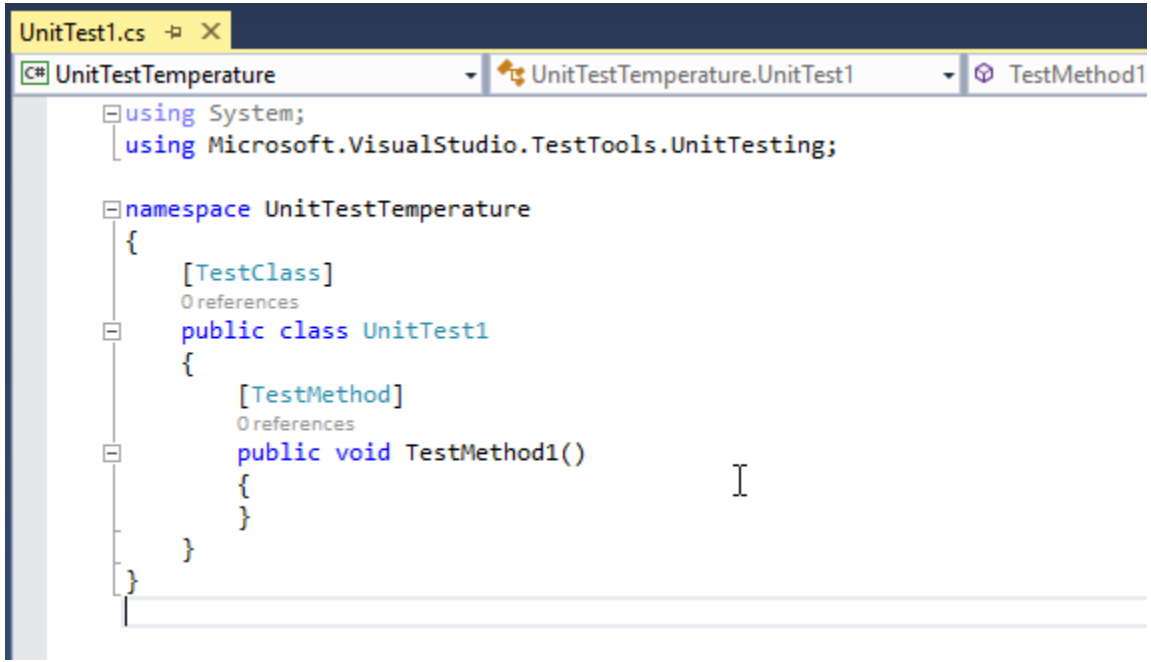


Add Reference to the Code under Test

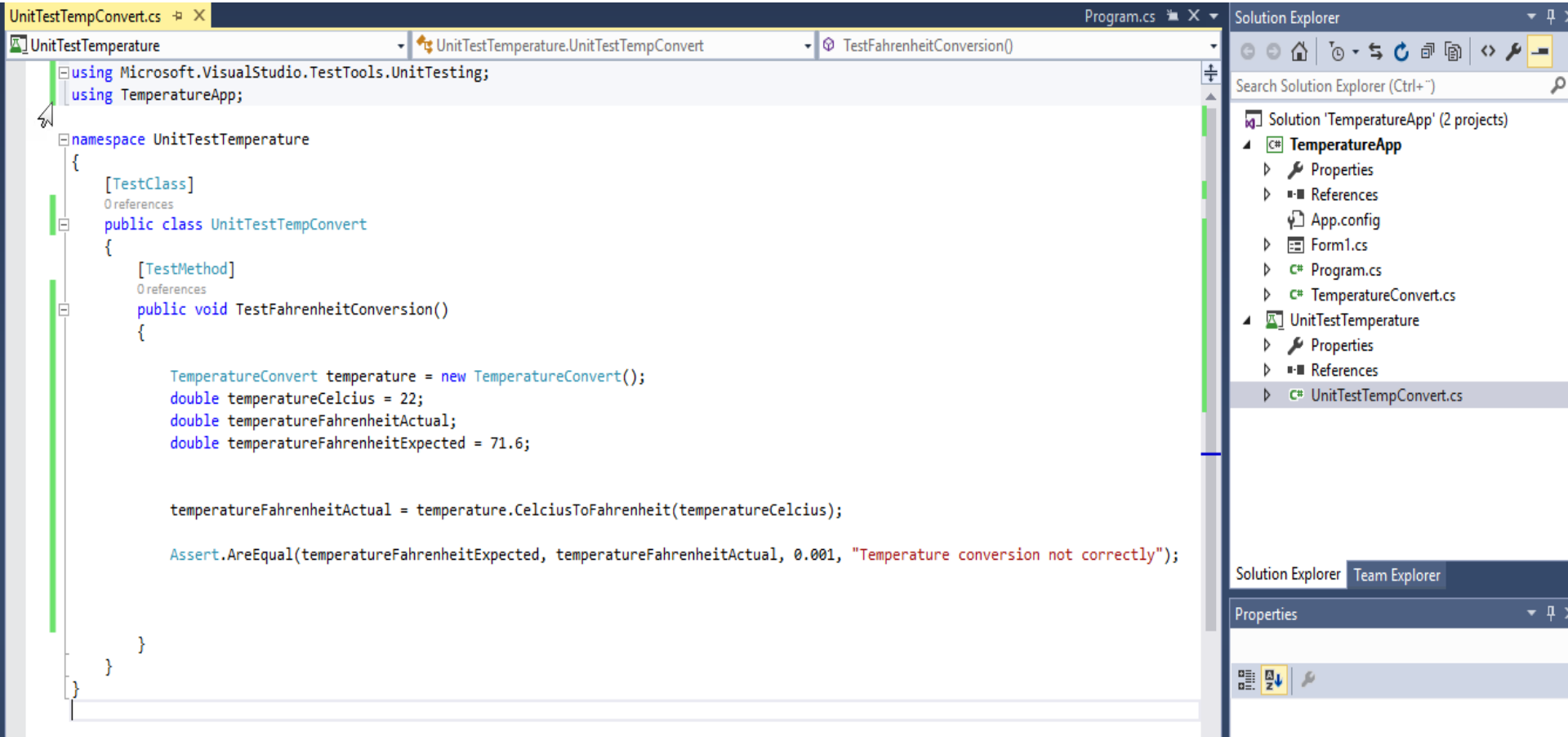



DEMO

Create the Unit Test Code



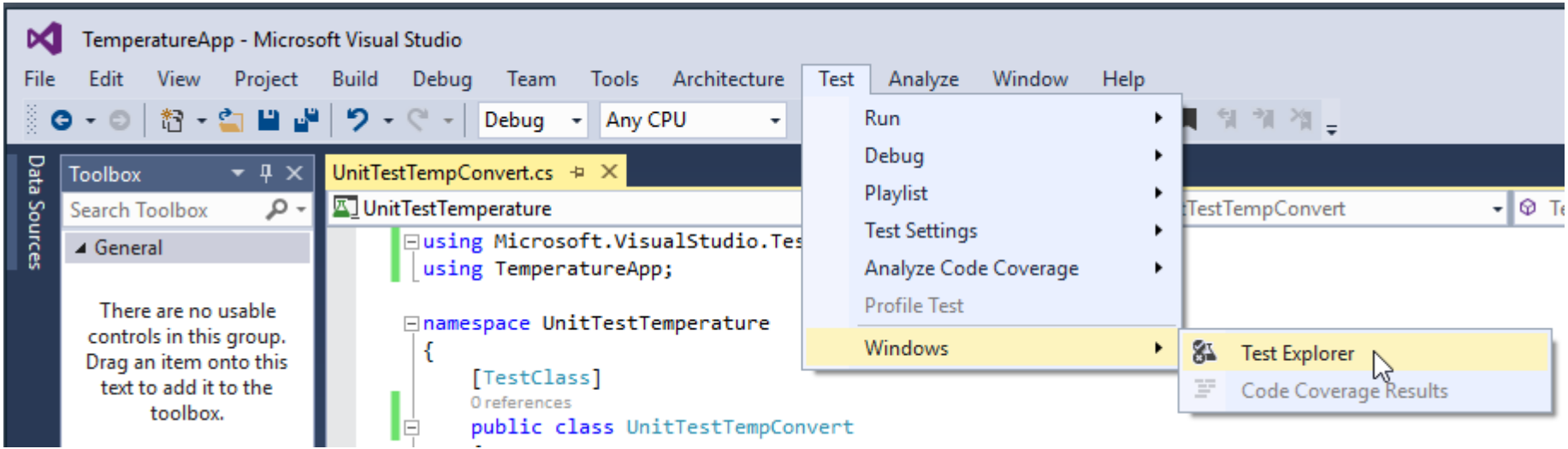
Create the Unit Test Code



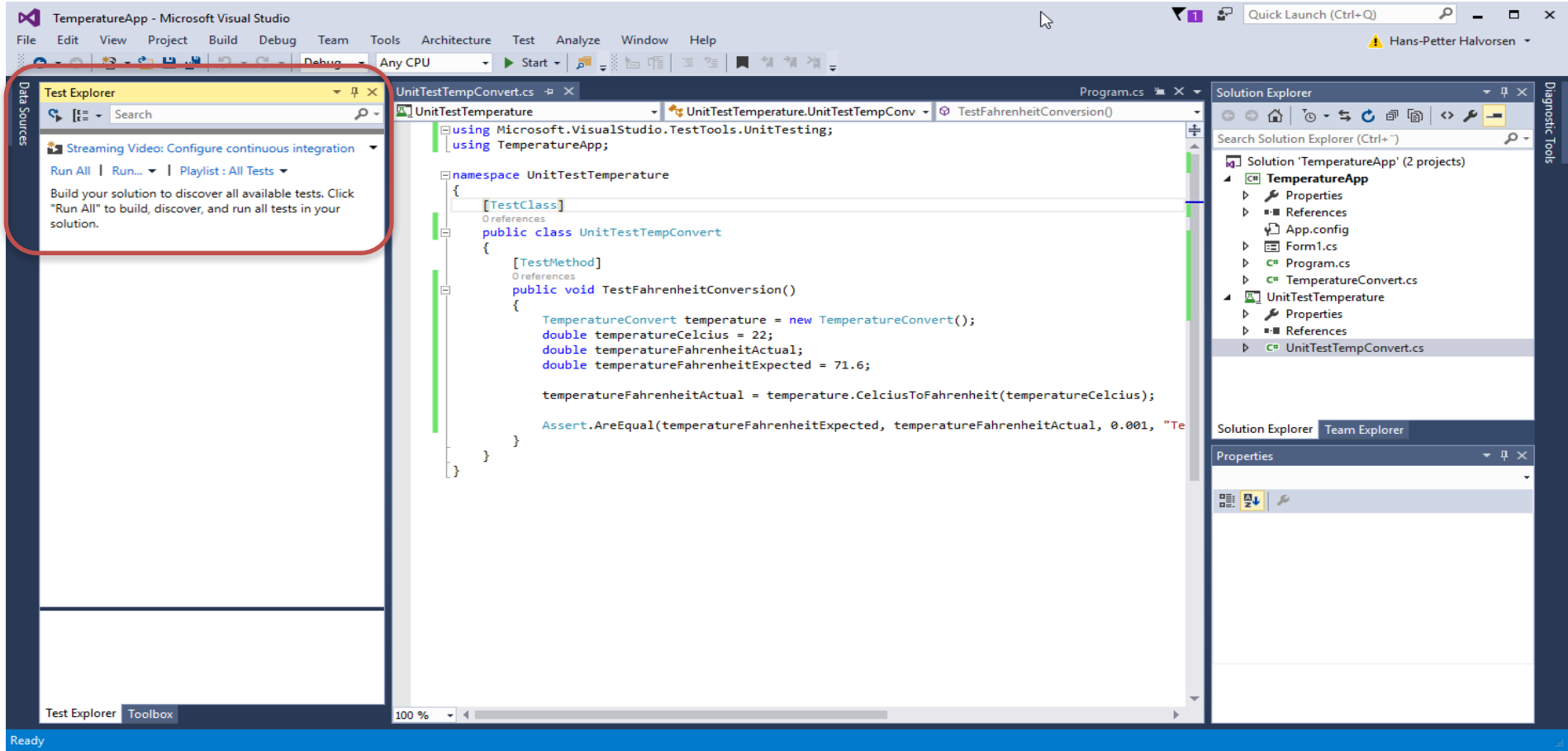
```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TemperatureApp;

namespace UnitTestTemperature
{
   [TestClass]
   public class UnitTestTempConvert
   {
      [TestMethod]
      public void TestFahrenheitConversion()
      {
         TemperatureConvert temperature = new TemperatureConvert();
         double temperatureCelcius = 22;
         double temperatureFahrenheitActual;
         double temperatureFahrenheitExpected = 71.6;

         temperatureFahrenheitActual = temperature.CelciusToFahrenheit(temperatureCelcius);

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


DEMO

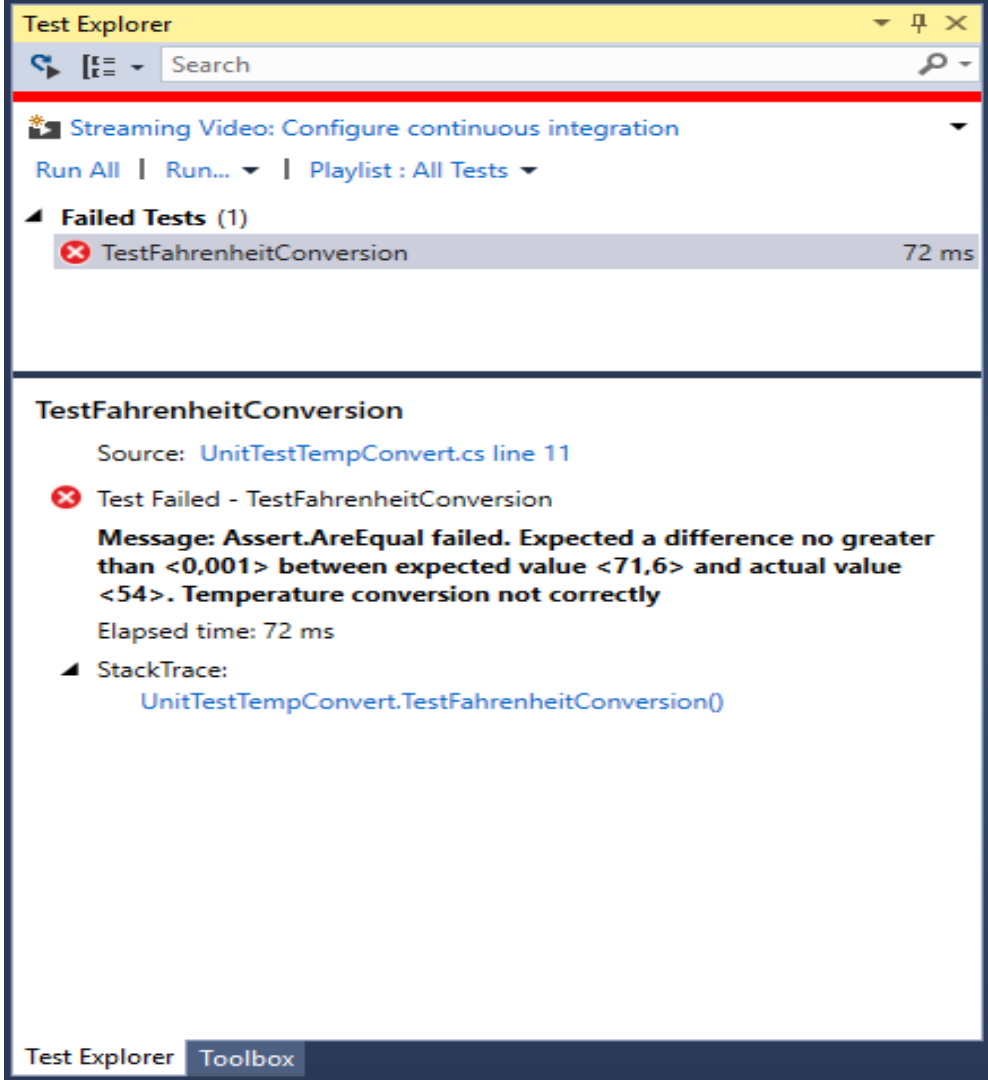
Test Explorer



Start Running the Unit Test



Test Results




DEMO

Debugging

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

```
public class TemperatureConvert
{

        public double CelciusToFahrenheit(double Tc)
        {
            double Tf;

            Tf = 9/5 * Tc + 32;

            return Tf;

        }

}
```

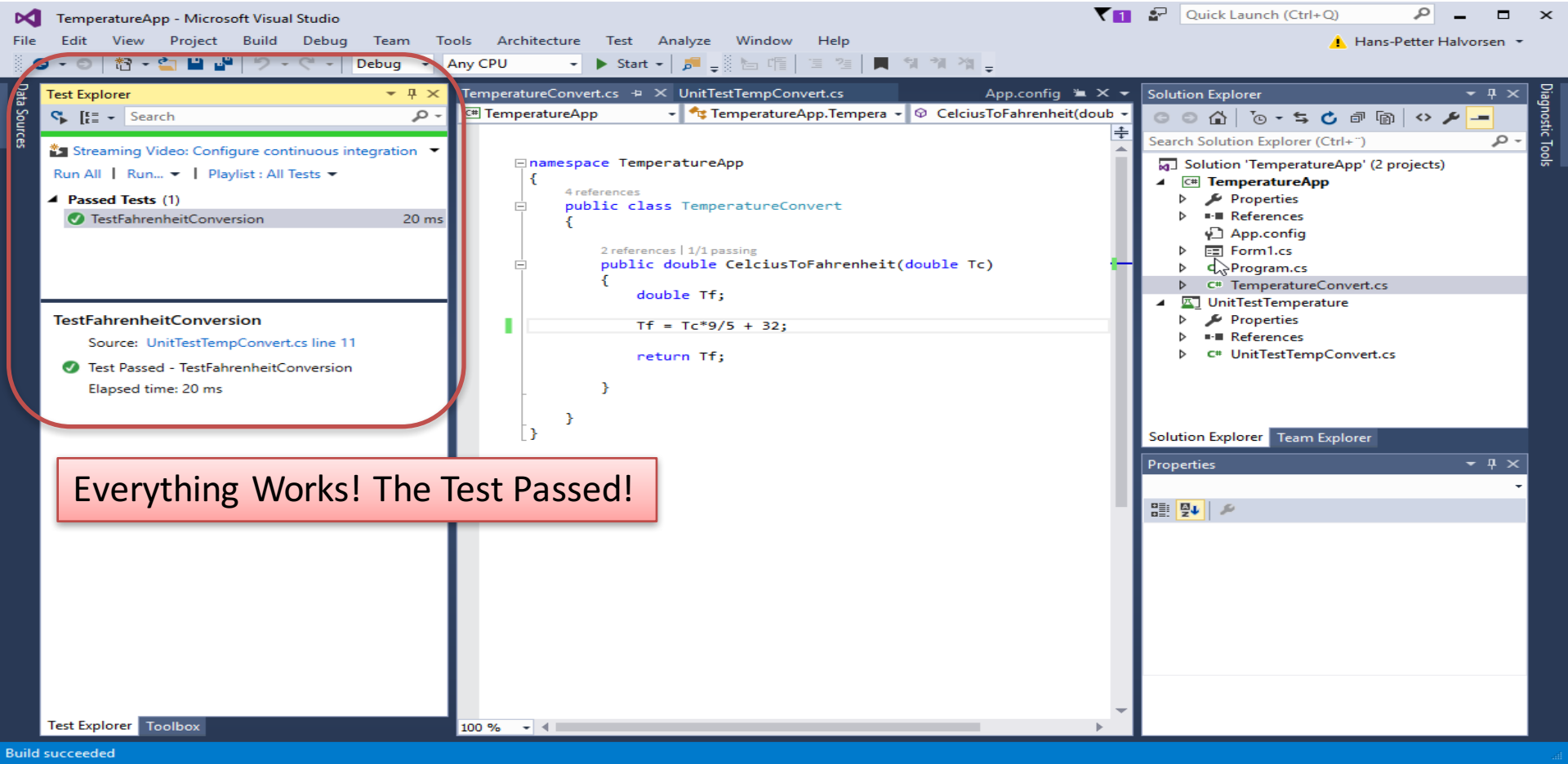
Probably Error in Formula?
What is wrong?

Fixing Bugs

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

```
public class TemperatureConvert
    {

        public double CelciusToFahrenheit(double Tc)
        {
            double Tf;

            Tf = Tc*9/5 + 32;

            return Tf;

        }

    }
```

Re-run Unit Test

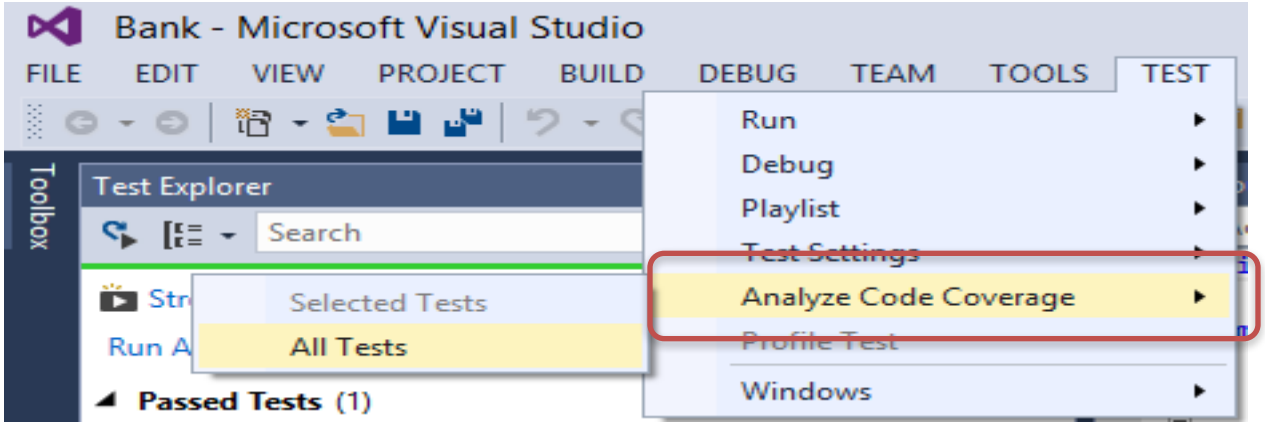



DEMO

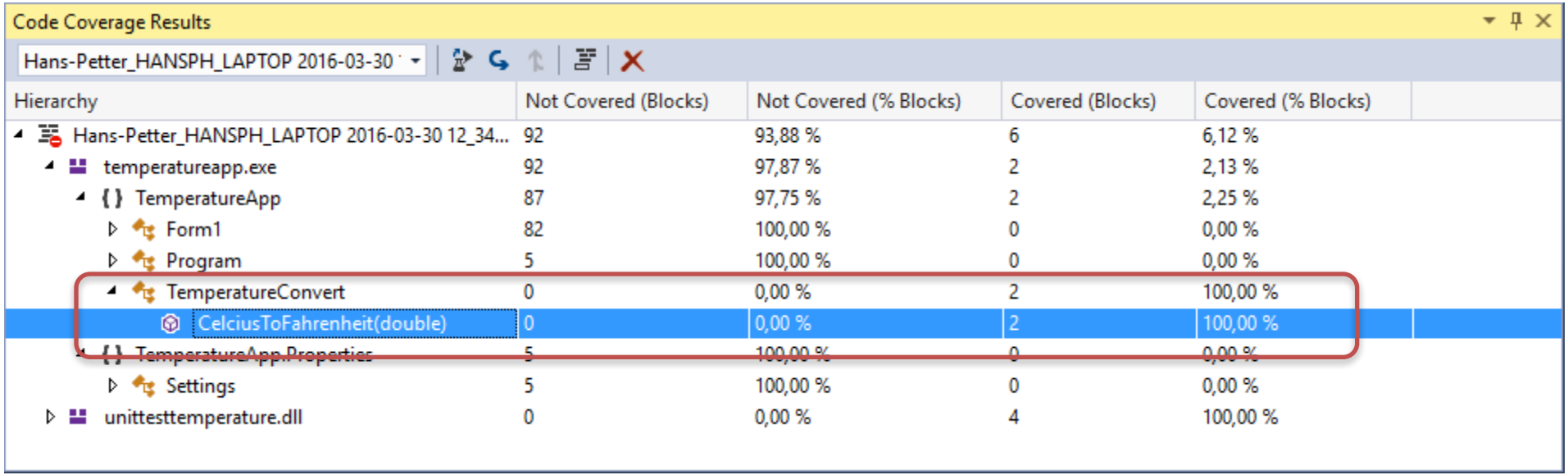
Checking Code Coverage

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.



Code Coverage Results



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


DEMO

Hans-Petter Halvorsen, M.Sc.

University College of Southeast Norway

www.usn.no

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

Blog: http://home.hit.no/~hansha/