Testing C# Code

Gill Cleeren
CTO Xpirit Belgium

@gillcleeren | xpirit.com/gill

Agenda

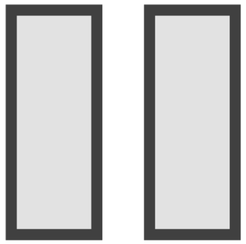
Testing your application using the debugger

Writing a unit test

Testing Your Application Using the Debugger

Using the Debugger

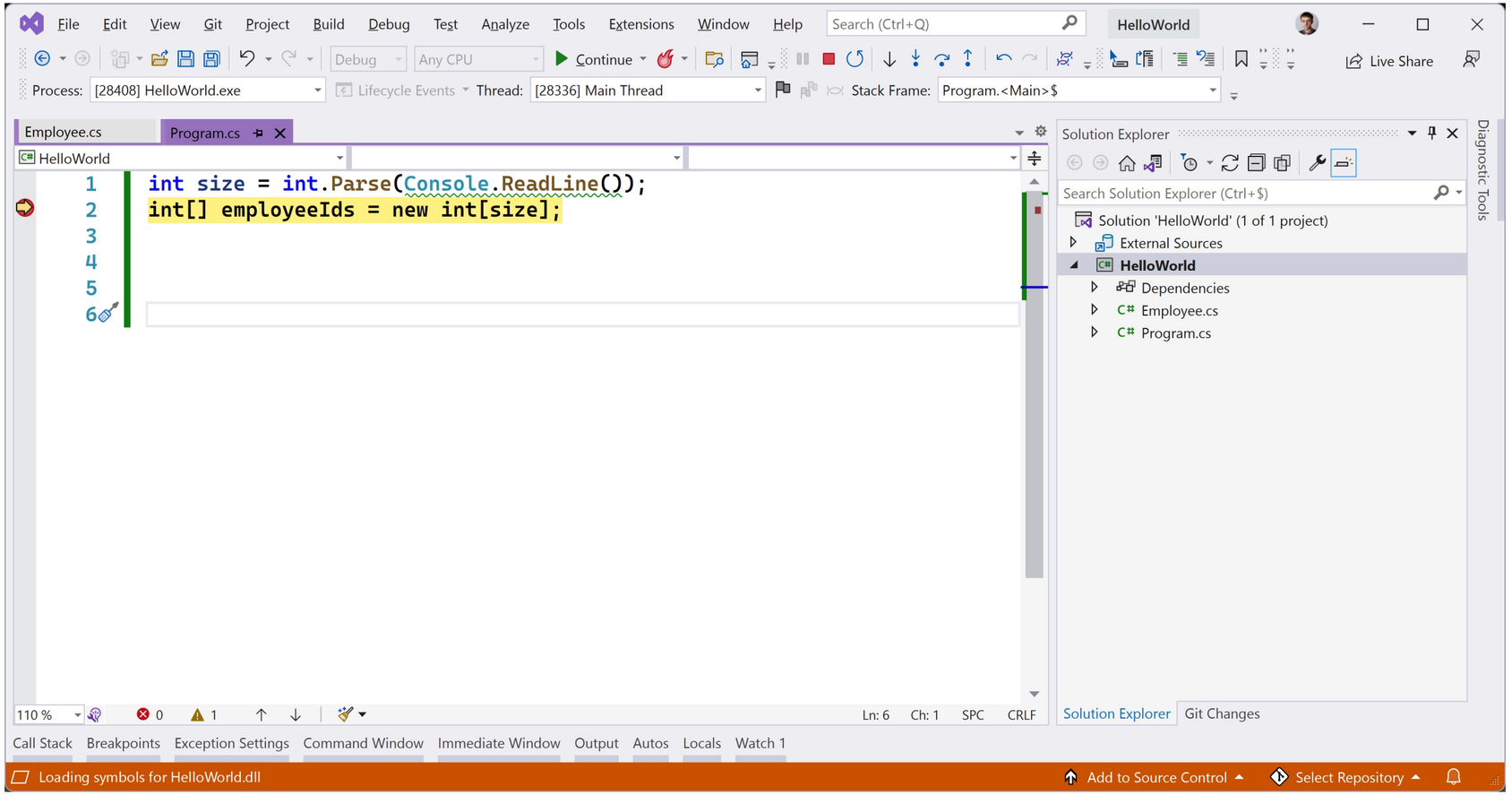
Run in Debug mode



Breakpoints

Stepping through the code

Using Break Mode and Breakpoints



Debugger Commands

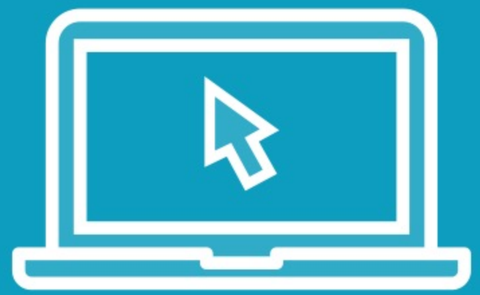
F5: start debugging

F11: Step Into

F10: Step over

Shift + F11: Step out

Demo



Using the debugger

Understanding the debugger windows

Writing a Unit Test

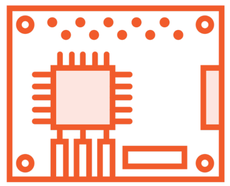
We aren’t perfect...

Introducing new bugs is easy!

We can try harnessing our code using a unit test

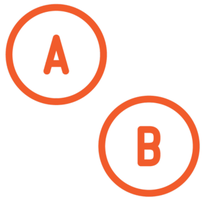
Introducing Unit Tests

Code to test other code



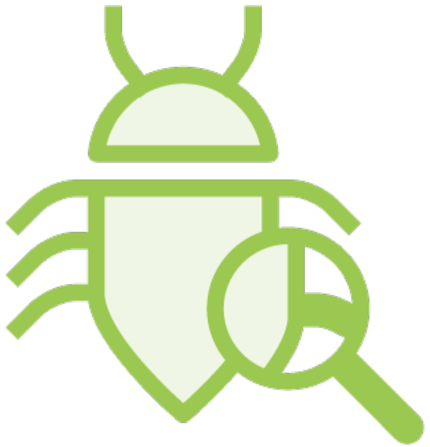
Small components of the application

Validate value

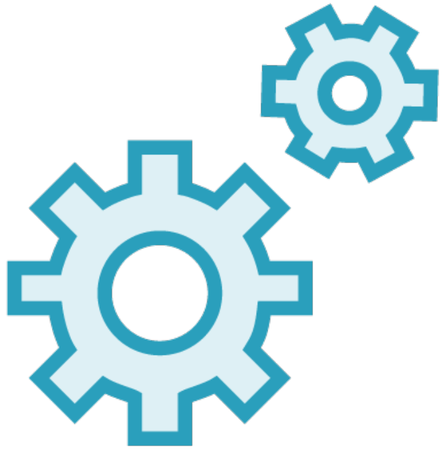


Isolate part of the code

Advantages of Unit Tests



Find bugs

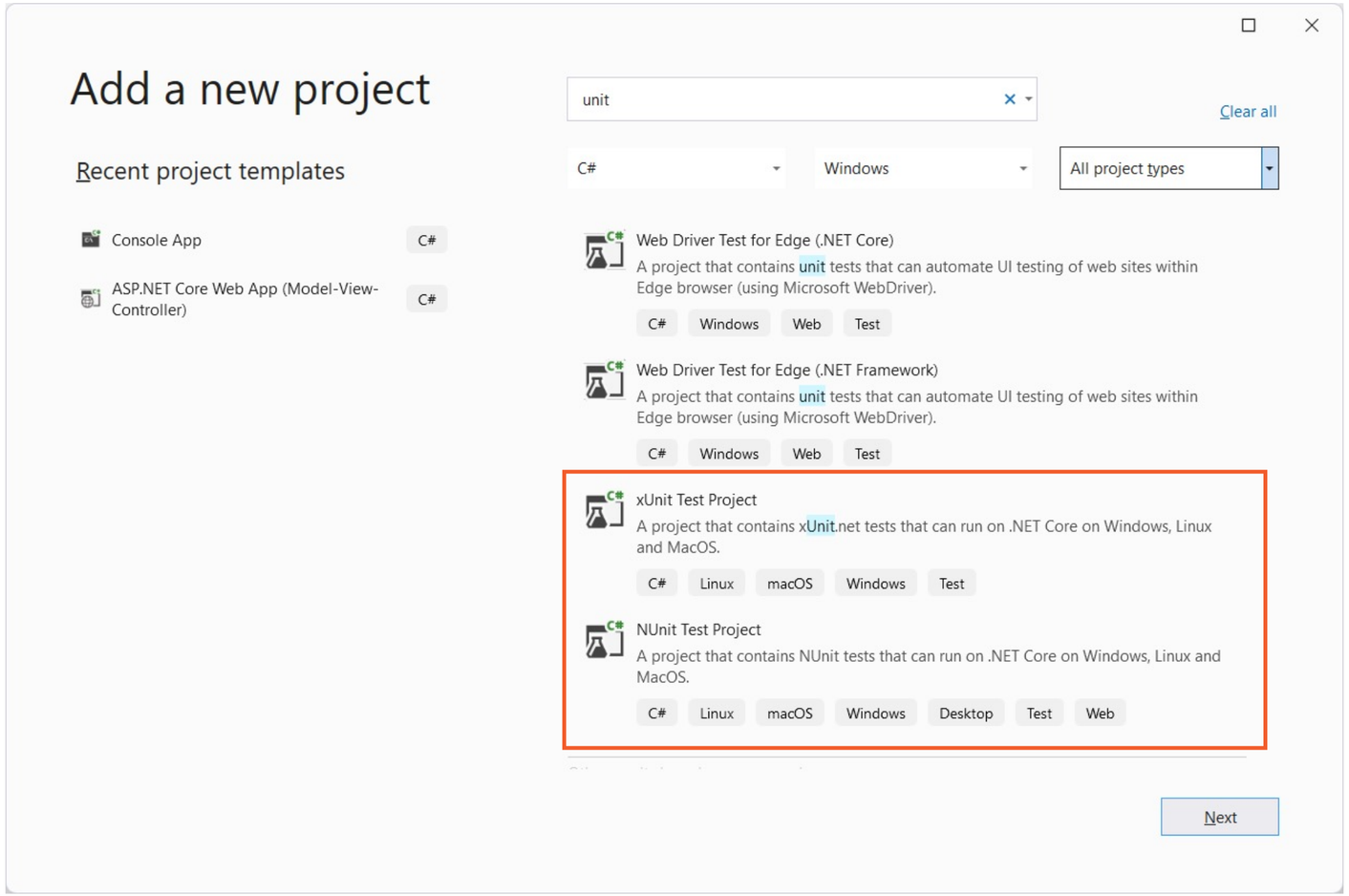


Change without fear of breaking something

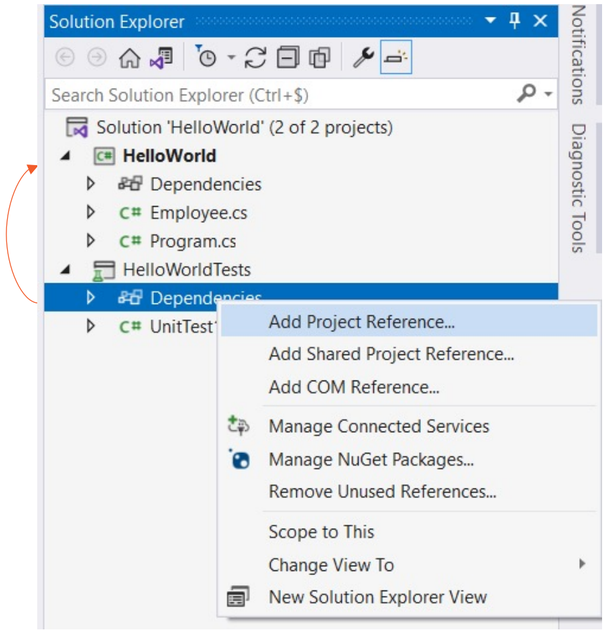
Improve quality

Documentation of the code

Creating a Unit Test Project



Sidestep: Adding a Reference

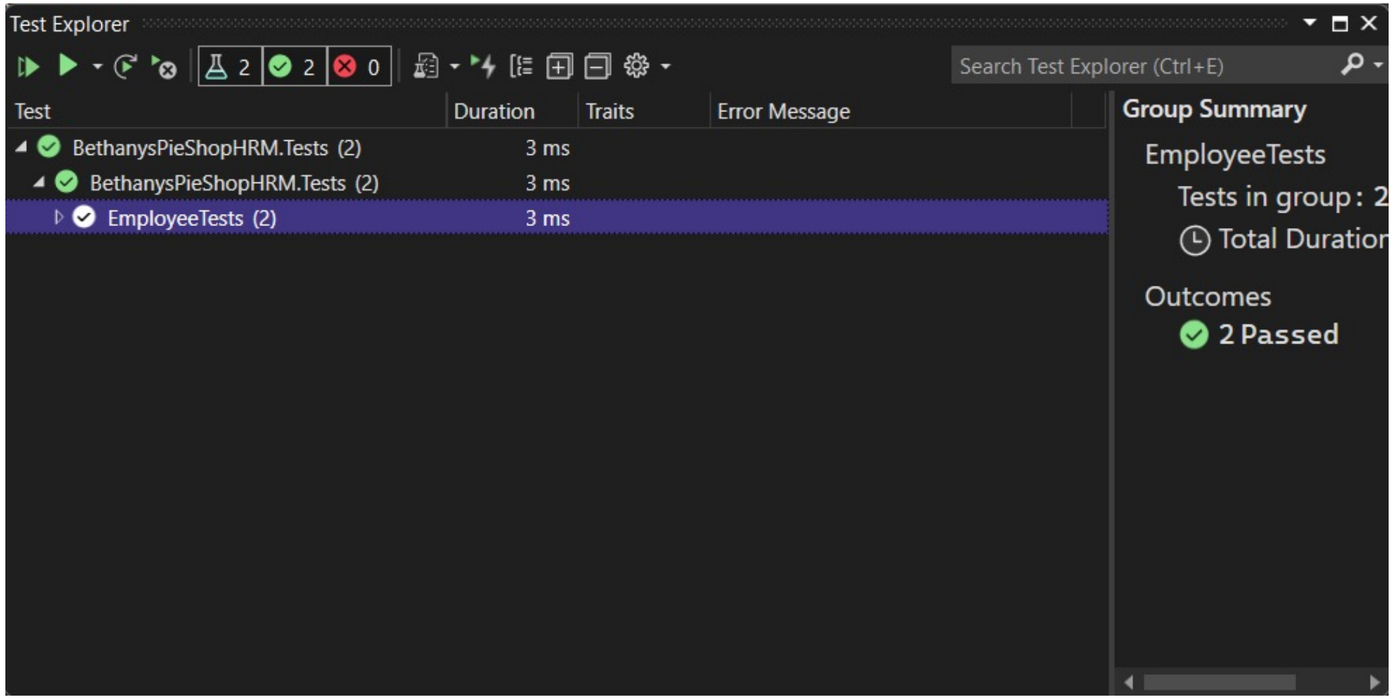


Structure of a Unit Test

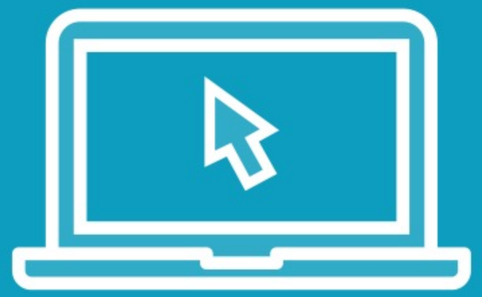
Writing a Unit Test

```
public class EmployeeTests
{
    [Fact]
    public void PerformWork_Adds_DefaultNumberOfHours_IfNoValueSpecified()
    {
        //Arrange
        Employee employee = new Employee(...);

        //Act
        employee.PerformWork();

        //Assert
        Assert.Equal(1, employee.NumberOfHoursWorked);
    }
}
```

Running Tests with Test Explorer



Demo



Creating a unit test project

Adding a unit test

Running the test using Test Explorer

Summary

Using the debugger, we can test our code and inspect values

Unit tests can help with making code more resistant to errors being introduced

Up next:

Working with files