Programming in C#. Fundamentals

SIGMA Software

Lesson 7
Unit tests

Unit tests

Introduction
3A – Arrange, Act and Assert

SIGMA Software

Why Write Automated Tests?

Happier development team

Fewer late nights/weekend work

More time to add new features

Happier users

Fewer defects reaching production causing annoyance

Reduced business cost

Defects found earlier in development lifecycle

Reliability

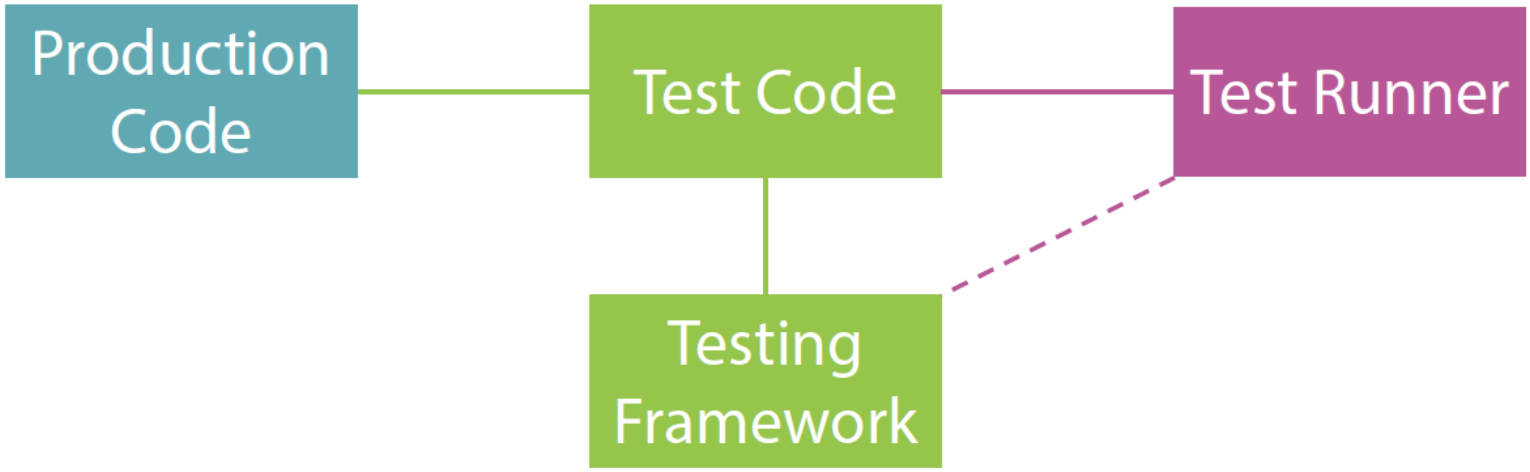
Exactly same test code runs each time

No variance between runs from Human error

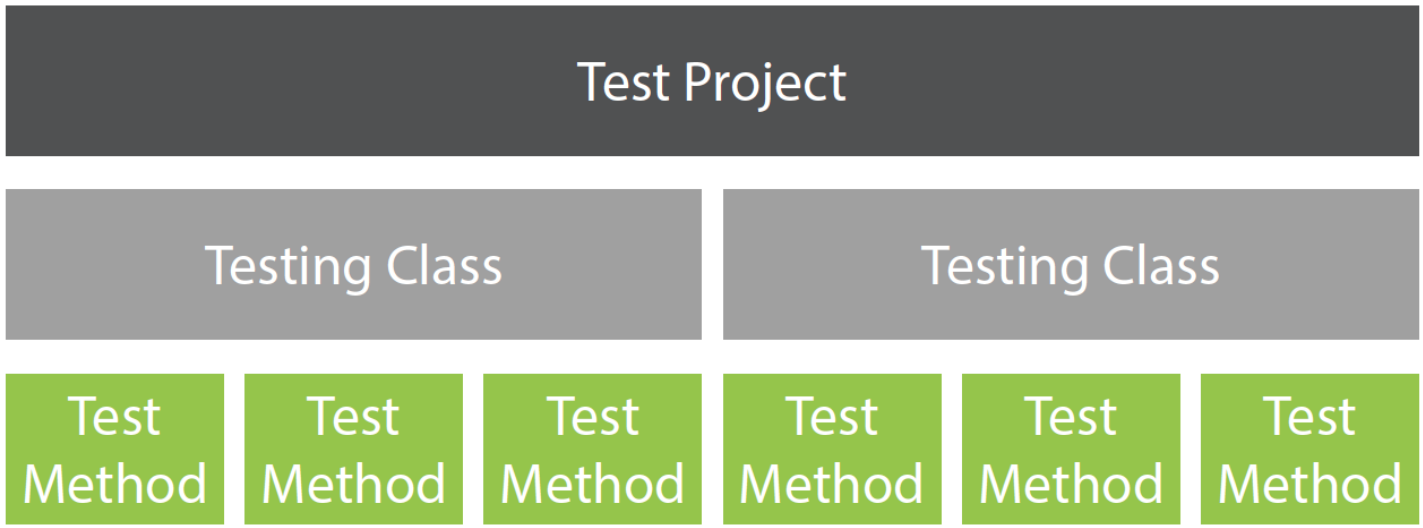
Faster execution

Quicker than a human performing tests manually

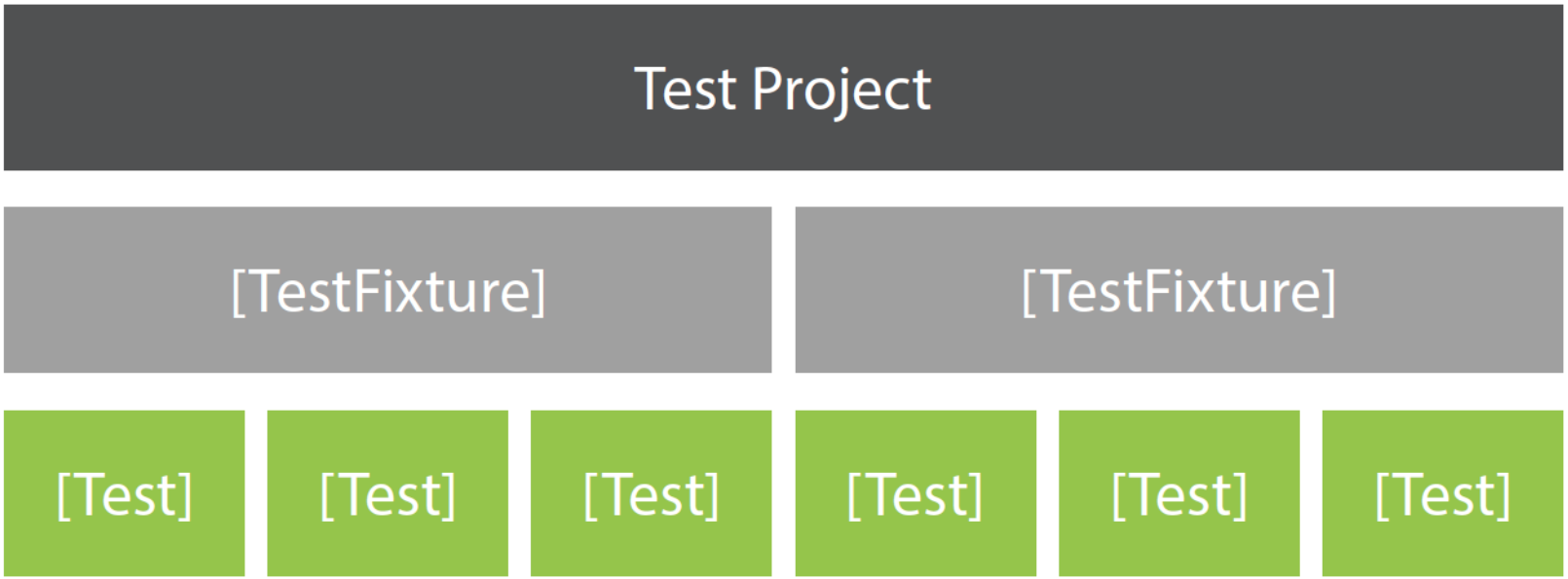
Testing Frameworks and Test Runners



NUnit Test Suite Organizational Structure



Designating Test Code

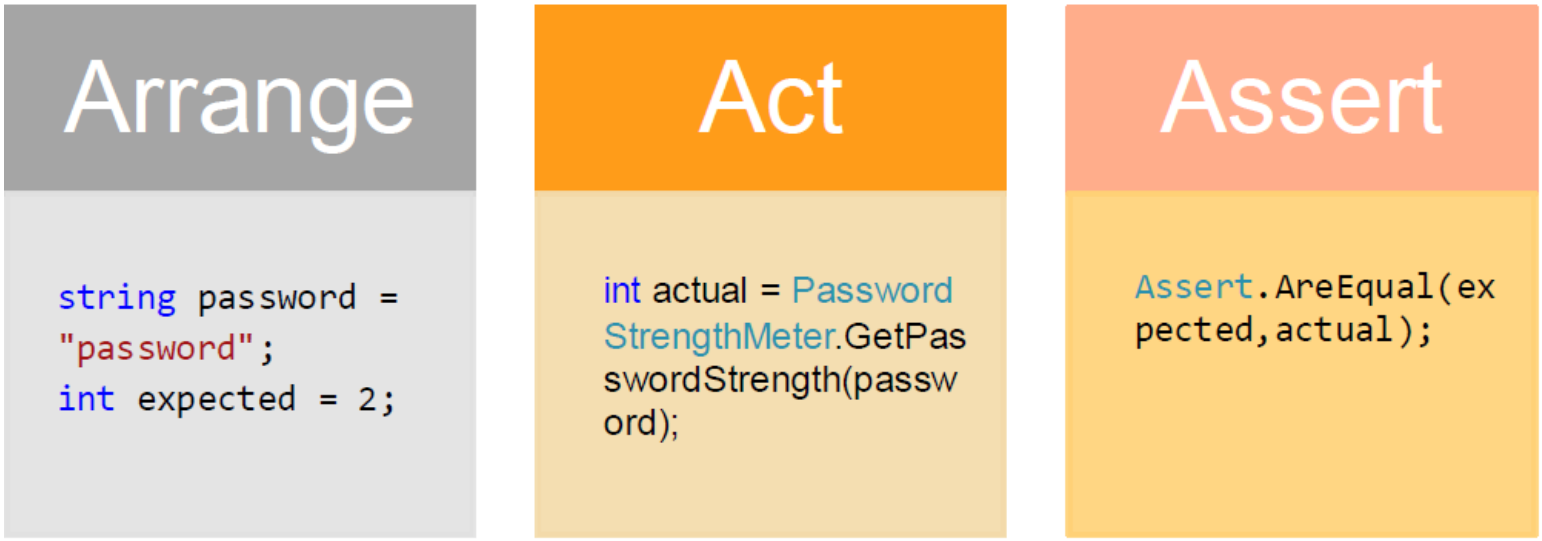


Designating Test Code

```
[TestFixture]
public class CalculatorTests
{
    [Test]
    public void ShouldAddTwoNumbers()
    {
        // Test code omitted
    }
}
```

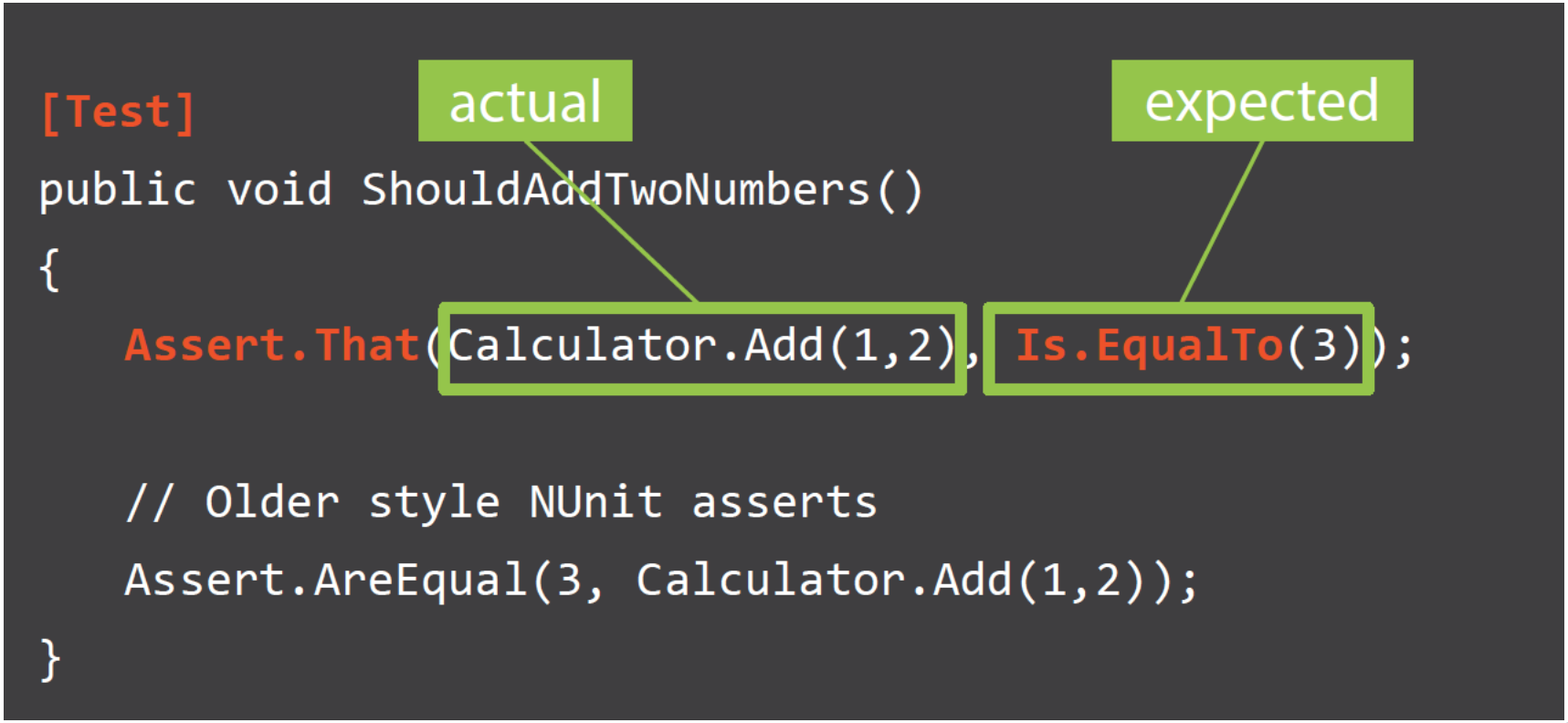
Creating your first unit test

Remember the “Three-A’s” pattern



What are Asserts?

Asserts tell the test runner whether a test has passed or failed



Example

```
[TestMethod]
public void AddTwoNumbers_Success()
{
    // Arrange
    const int firstNumber = 1;
    const int secondNumber = 2;
    var calculator = new Calculator();

    // Act
    var result = calculator.Add(firstNumber, secondNumber);

    //Assert
    Assert.AreEqual(3,result);
}
```

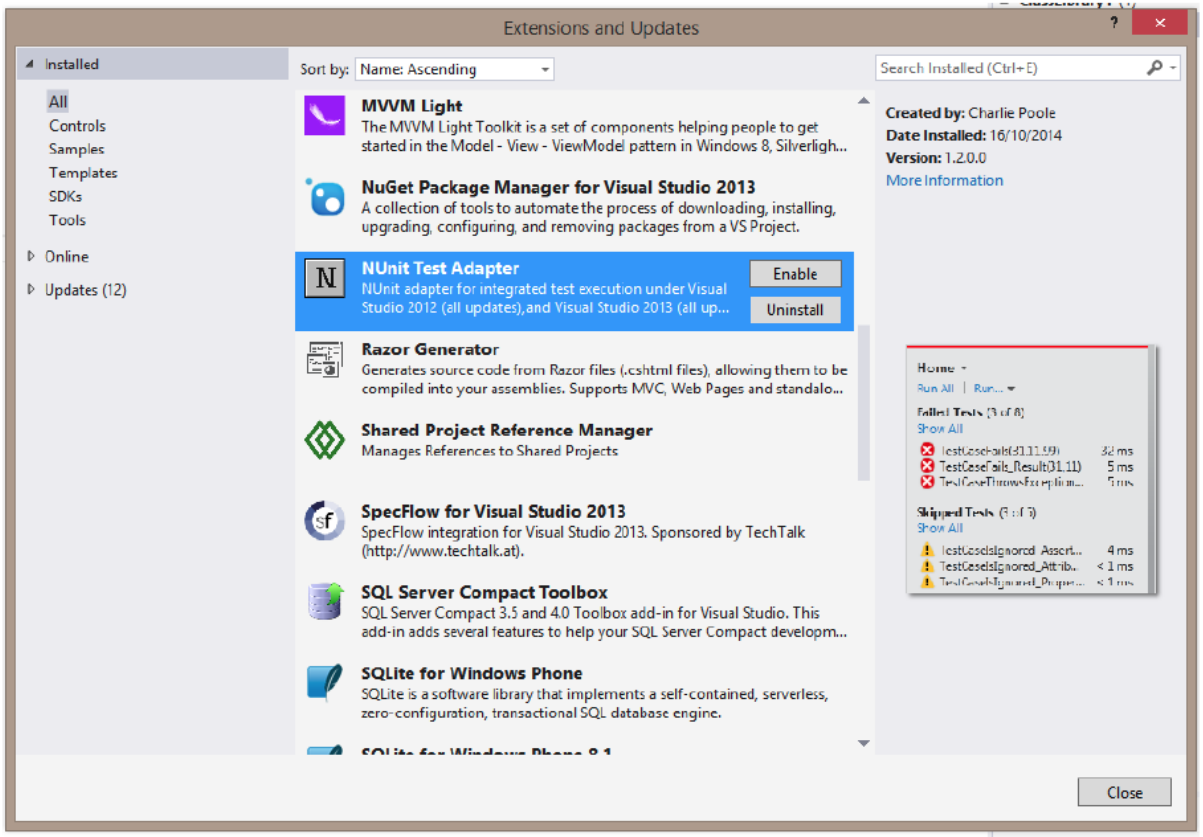
What Makes a Good Test?

- Independent & isolated
- Test single behaviour / logical thing
- Clear intent / readable
- Don't test the compiler
- Reliable & repeatable
- Production quality code
- Valuable

Naming Conventions

```
[TestFixture]
public class CalculatorTests
{
    [Test]
    public void ShouldAddTwoNumbers()
    {
        var sut = new Calculator();
        // System Under Test
    }
}
```

Enabling NUnit Test Execution in Visual Studio



Tools → Extensions and Updates

Writing an NUnit Test from Scratch

- Create new test project
- Reference production code project
- Install NUnit NuGet package
- Write a test class and methods

Three Part Naming

Test Organization

High Precision

Test one expectation per test

Multiple asserts on same object can be OK

Test should point to precise location of problem

Unit Testing Checklist

- Test name describes the scenario
- Contains arrange, act, assert
- Stays within one project layer
- Is a state, value, or interaction test
- Fakes all dependencies
- Mocks at most one dependency
- Favors the public API
- Asserts against one object
- Favors builder over setup methods

Q & A

Practice
Lesson 7

Home work