NASH TECH
The Power to Innovate

Writing Unit Tests

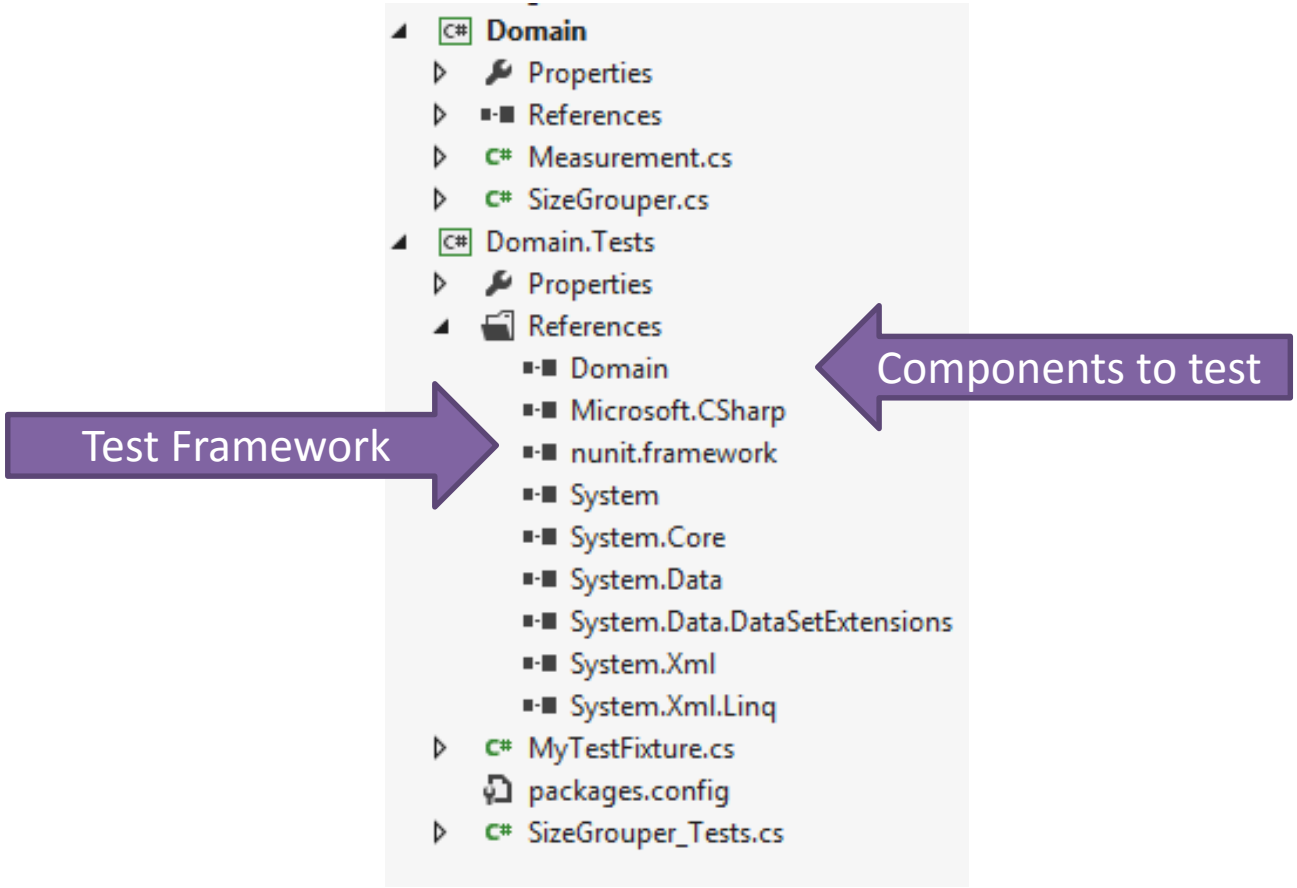
Dung Le Hoang

Agenda

- Test Projects
- Test Attributes
- Test Fixtures
- Test Runners
- Test Tips

Setting Up A Test Project

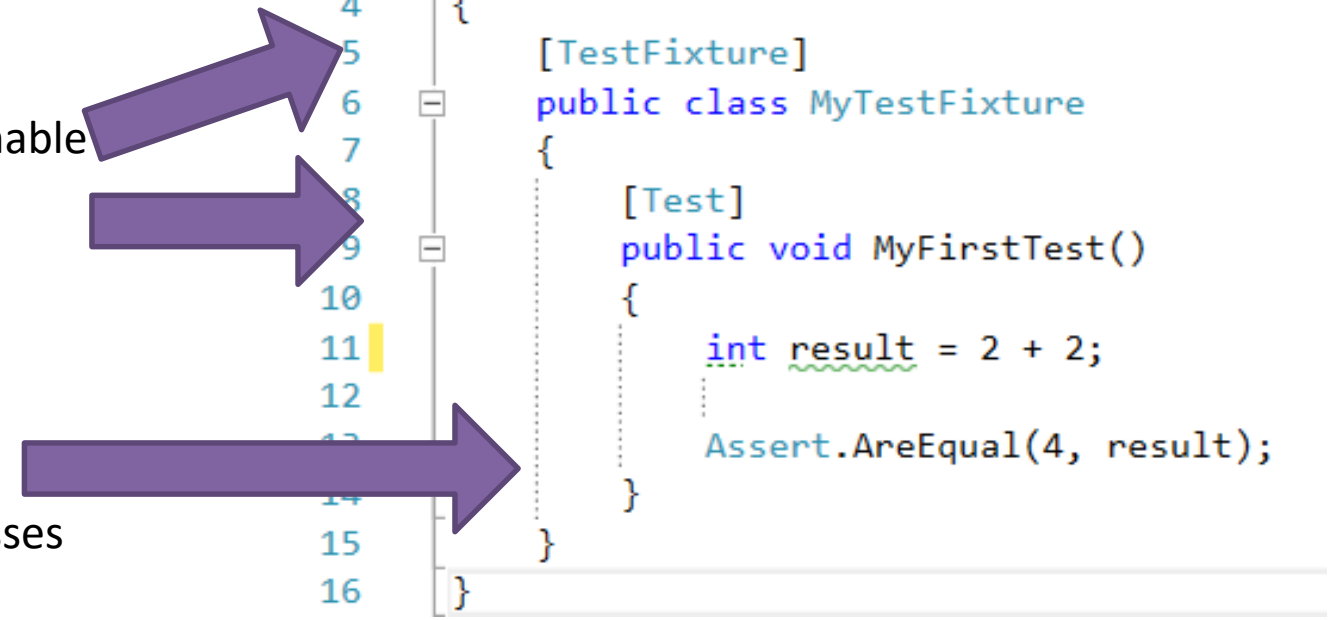
- Tests live in a separate class library project



A First Test

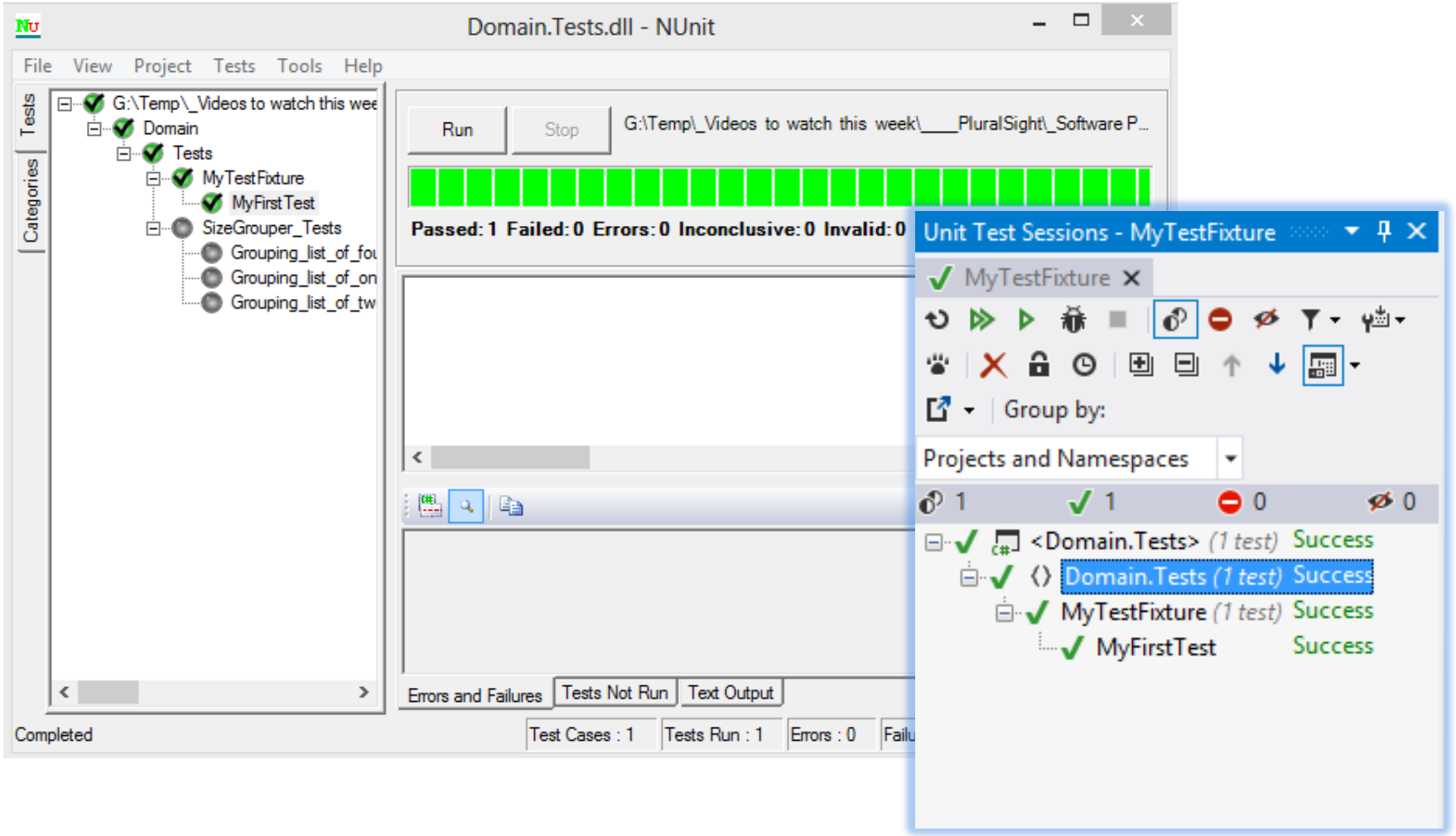
Attributes enable unit test functionality

This test passes

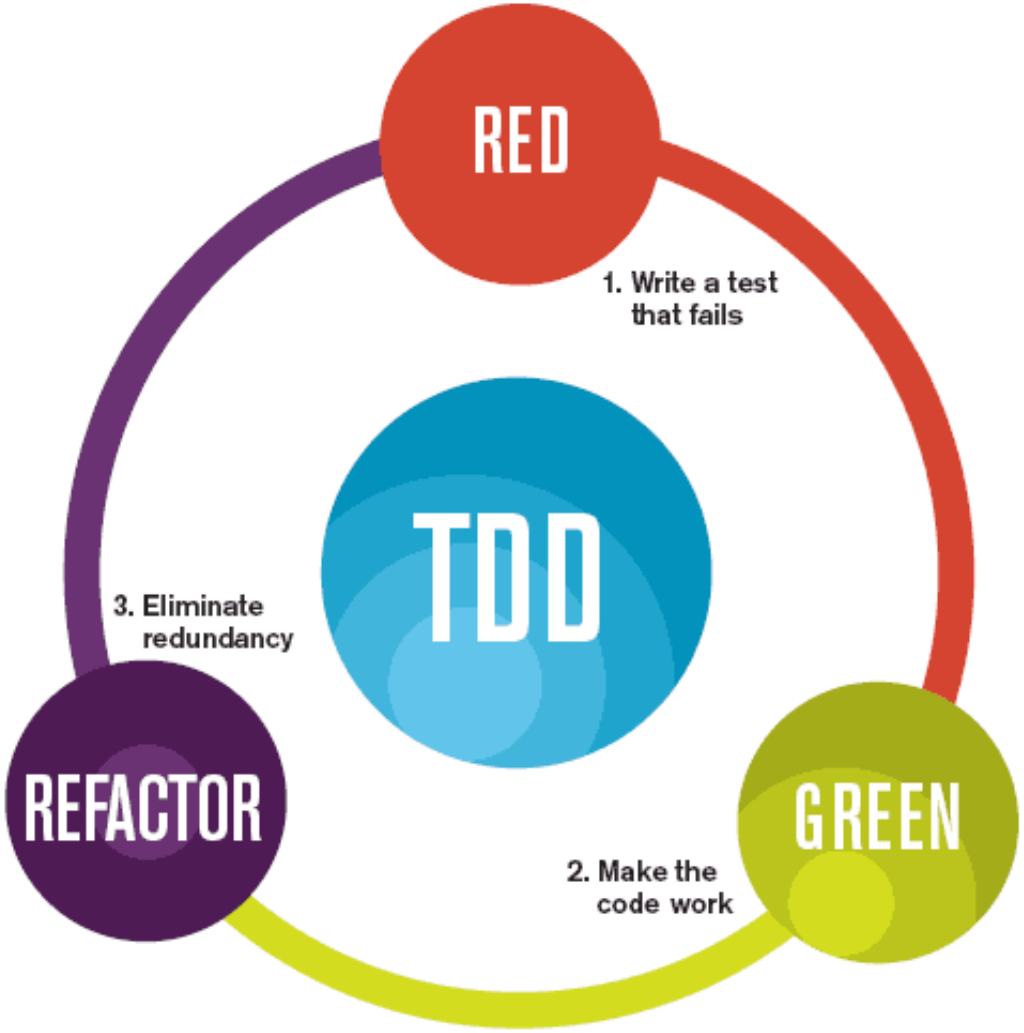


```
1   using NUnit.Framework;
2
3   namespace Domain.Tests
4   {
5       [TestFixture]
6       public class MyTestFixture
7       {
8           [Test]
9           public void MyFirstTest()
10          {
11              int result = 2 + 2;
12
13              Assert.AreEqual(4, result);
14          }
15      }
16  }
```

Write first Unit Test



Test Driven Development



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Assertions

- One behavior testing per unit test

```
1  [Test]
2  public void the_order_is_canceled()
3  {
4      var customer = CreateCustomer();
5      Assert.IsNotNull(customer);
6
7      customer.PlaceOrder();
8      Assert.IsTrue(customer.HasOrder);
9
10     customer.CancelOrder();
11     Assert.IsFalse(customer.HasOrder);
12 }
```

Test Code is important indeed

- Keep test code maintainable and DRY

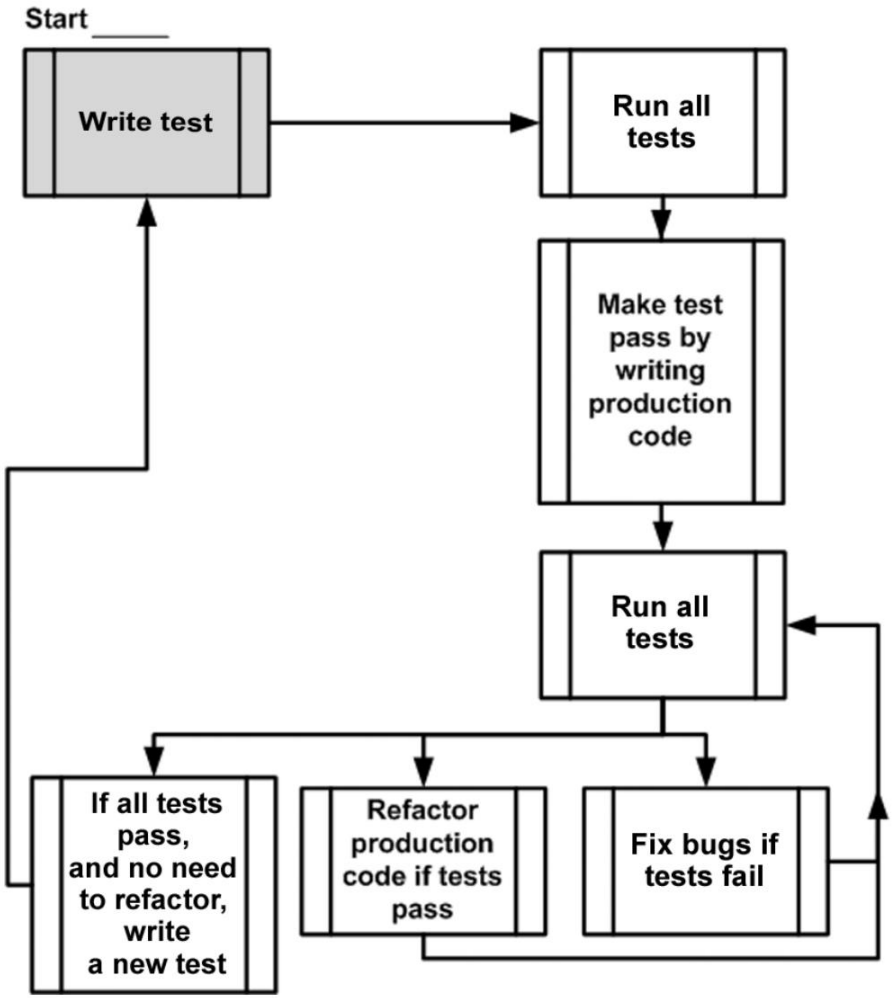
Test Qualities

- Repeatable
  - Tests shouldn't fail after 6 pm
- Independent
  - Tests shouldn't rely on state from another test

Test First … design

- Test only public members
  - Think like a client
  - Make the tests more robust

TDD process


Start
Write test
Run all tests
Make test pass by writing production code
Run all tests
If all tests pass, and no need to refactor, write a new test
Refactor production code if tests pass
Fix bugs if tests fail

Summary

- Write tests in a separate project
- Stick with Red – Green – Refactor
- Treat test code with respect
- Keep practicing and learning