

Writing Unit Tests

Dung Le Hoang

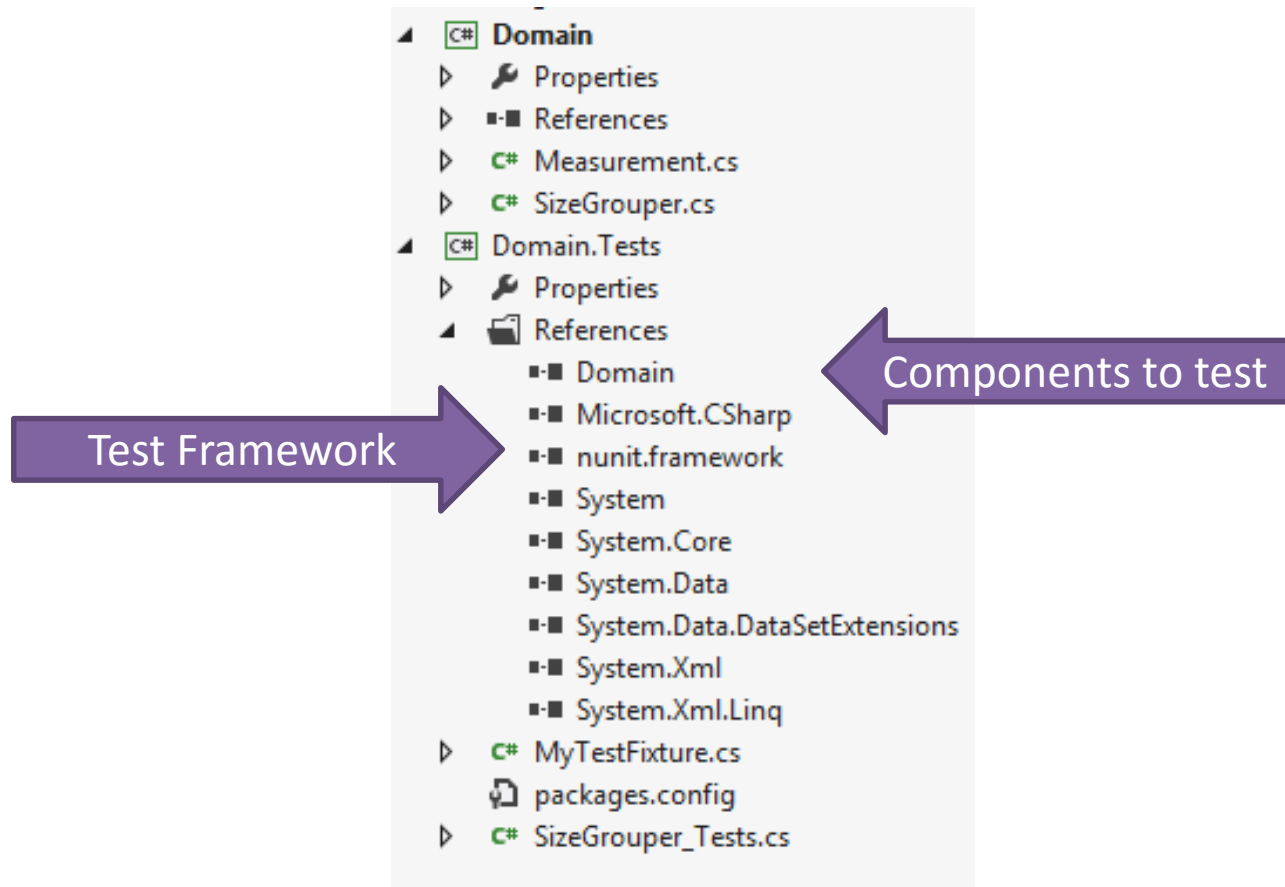


Agenda

- 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

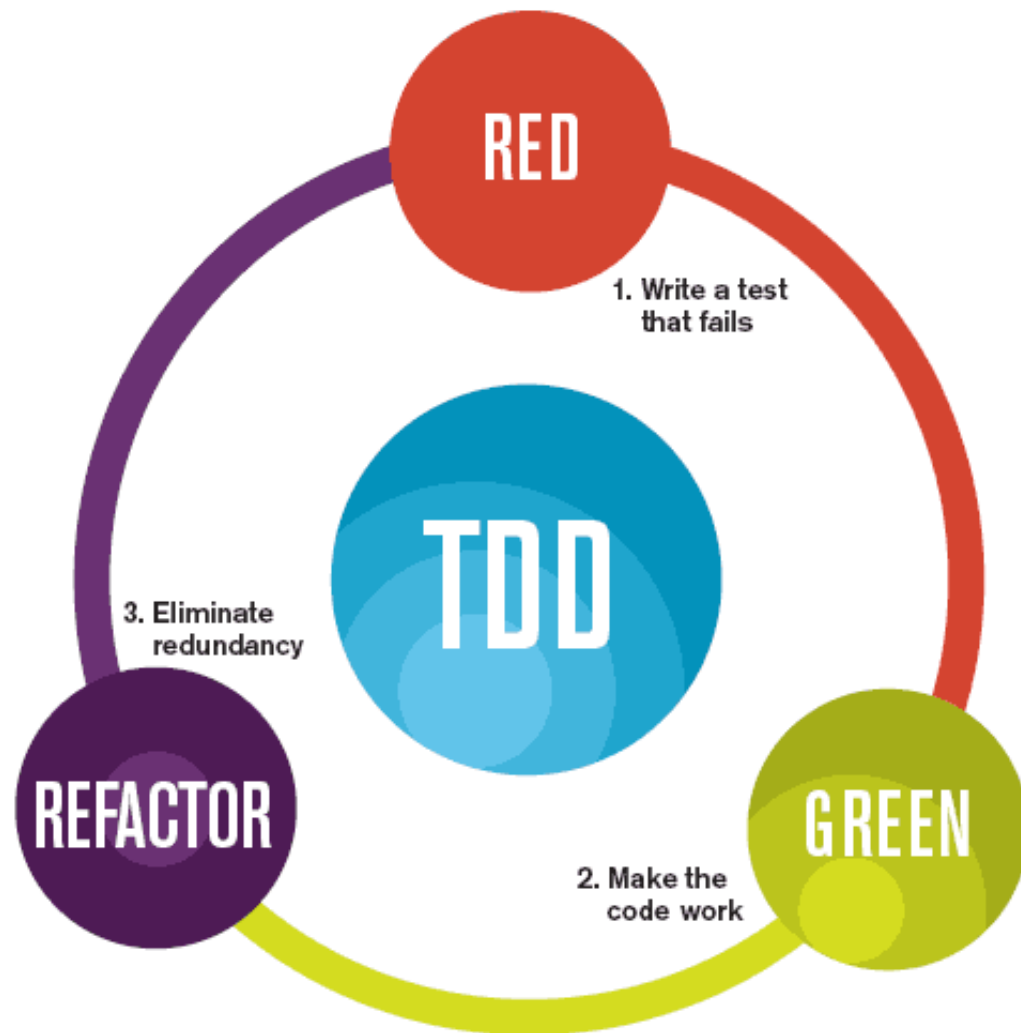
The screenshot displays the NUnit application window titled "Domain.Tests.dll - NUnit". The interface includes a menu bar (File, View, Project, Tests, Tools, Help) and a sidebar with "Tests" and "Categories" views. The "Tests" view shows a tree structure of test items: "G:\Temp_Videos to watch this week" (checked), "Domain" (checked), "Tests" (checked), "MyTestFixture" (checked), "MyFirstTest" (checked), "SizeGrouper_Tests" (unchecked), and several "Grouping_list_of_" items (unchecked). The main area shows a "Run" button, a "Stop" button, and a progress bar consisting of 10 green squares. Below the progress bar, the status is reported as "Passed: 1 Failed: 0 Errors: 0 Inconclusive: 0 Invalid: 0".

Overlaid on the main window is a smaller window titled "Unit Test Sessions - MyTestFixture". This window shows a list of test sessions with a toolbar for actions like rerun, pass, fail, and debug. The list shows the following results:

Session	Result
<Domain.Tests> (1 test)	Success
Domain.Tests (1 test)	Success
MyTestFixture (1 test)	Success
MyFirstTest	Success

At the bottom of the main window, there are tabs for "Errors and Failures", "Tests Not Run", and "Text Output". The status bar at the very bottom shows "Test Cases : 1 Tests Run : 1 Errors : 0 Failures : 0".

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

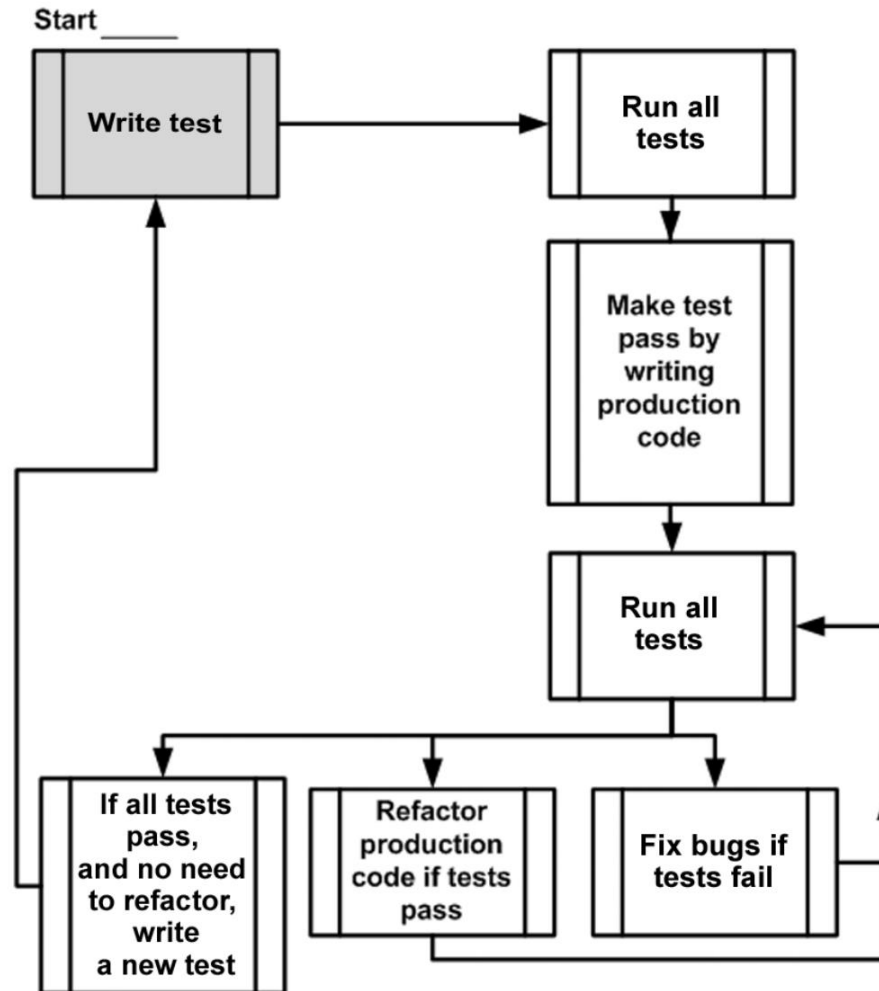
- Repeatable
 - Tests shouldn't fail after 6 pm
- Independent
 - Tests shouldn't rely on state from another test



Test First ... design

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

TDD process



Summary

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

