**Unit Testing**

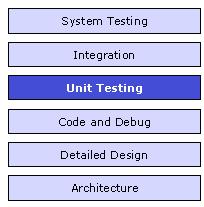
Every software is composed of various modules. Each module is composed of various classes. Classes composed of various functions. Function is the smallest unit of code in the application.

When we test individual function behavior without touching any other functions and determine whether it works exactly as per the requirements or not that is called Unit Testing.

Unit testing is a kind of testing done at the developer side.

advantages of Unit Testing:

1. Defects found early in development life cycle
2. Reliable Code
3. Maintainable code
4. Faster testing by only single click of action



NUnit is a unit testing framework for .NET. It is the most used framework for writing unit test cases.

NUnit is a unit testing framework for performing unit testing based on the .NET platform.

NUnit is open source.

NUnit is very easy to use. It only provides some custom attributes and some static Assert classes. With the combination of custom attributes and static classes, we can write unit test cases easily.

Custom attributes provides hint to NUnit test runners that these classes or functions contains unit testing code.

Some of the custom attributes are:

* TestFixture
* Setup
* TearDown
* Test
* Category
* Ignore
* TestCase
* Repeat
* MaxTime

Assert classes is used to test the conditions. If condition is satisfied then test is pass else fail.

**Test Runners** are UI tool which actually run NUnit test cases and show the result of test cases whether they are passed or failed.

A test case body is divided into three sections "AAA".

"AAA" denotes the Arrange, Act, and Assert.

**Arrange:** In Arrange section, we will initialize everything which we are required to run the test case. It includes any dependencies and data needed.

**Act:** In Act section, we called the business logic method which behavior we want to test.

**Assert:** Specify the criteria for passing the test case. If these criteria passed, that means test case is passed else failed.

**First NUnit Test Case Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | [TestCase]  public void When\_PremiumCustomer\_Expect\_10PercentDiscount()  {      //Arrange      Customer premiumCustomer = new Customer      {          CustomerId = 1,          CustomerName = "George",          CustomerType = CustomerType.Premium      };        Order order = new Order      {          OrderId = 1,          ProductId = 212,          ProductQuantity = 1,          Amount = 150      };        CustomerOrderService customerOrderService = new CustomerOrderService();        //Act      customerOrderService.ApplyDiscount(premiumCustomer, order);        //Assert      Assert.AreEqual(order.Amount, 135);  } |

In Arrange section, we initialize two variables premiumCustomer and order. In the next line, we create an instance of our CustomerOrderService class.

In Act section, we called actual method of CustomerOrderService class with variables initialized in Arrange section.

In Assert section, we check our expected response. Is amount equals to 135 or not?

Custom Attributes Description

**TestFixture**

The class that is to be tested using Nunit should be decorated with TextFixture.

**Test**

This attribute identifies the method to be tested. If we do not write this attribute then we can't to identify the test in Testexplorer.

**SetUp**

This attribute is used when you want to execute a piece of code in each test case. It identifies a method to be executed each time before a TestMethod/Test is executed.

**TearDown**

After completely executing each test if you want to execute a piece of code then you have to write this code under TearDown attribute.

**Note**  
If two SetUp classes are there the class that was written first will be executed first and then after that, the class that is written next to it will execute. (Top to bottom approach).  
  
Similarly, in tear down the order is reversed -- it will follow bottom to top approach.

**Ignore Attribute**

The Ignore attribute is required to indicate that a test should not be run on a particular method.

1. [TestFixture]
2. **public** **class** Program
3. {
4. [Test]
5. [Ignore("This method is skipping")]
6. **public** **void** Test()
7. {
8. ...
9. }
10. }

**ExpectedException Attribute**

This attribute is used to test methods that need to throw an exception in some cases. The cases may be FileNotFoundException,DividebyZeroException etc.

1. [TestFixture]
2. **public** **class** Program {
3. [Test]
4. [ExpectedException(**typeof**(DivideByZeroException))]
5. **public** **void** Test() {
6. **int** i = 10, j = 0, x;
7. x = i / j;
8. }
9. }

Order Attribute

Sometimes we need to execute test methods in a particular order. These test method are dependent on each other. For that, NUnit provides the Order attribute.

**[Test]**

**[Order(1)]**

**public void When\_AgeGreaterAndEqualTo60\_Expects\_IsSeniorCitizeAsTrue()**

**{**

**....**

**}**

**[Test]**

**[Order(2)]**

**public void When\_AgeLessThan60\_Expects\_IsSeniorCitizeAsFalse()**

**{**

**....**

**}**