Unit testing is a software testing method where individual units or components of a software application are tested in isolation to ensure they work as expected. [A “unit” is typically the smallest testable part of an application, such as a function, method, procedure, module, or class](https://en.wikipedia.org/wiki/Unit_testing)

Unit testing in C# involves writing tests for individual units of code, such as methods or   
classes, to ensure they work correctly. Here’s a step-by-step guide to get you started:  
  
In the software development process Unit Tests basically test individual parts ( also called as Unit ) of code (mostly methods) and make it work as expected by programmer. A Unit Test is a code written by any programmer which test small pieces of functionality of big programs. Performing unit tests is always designed to be simple, A "UNIT" in this sense is the smallest component of the large code part that makes sense to test, mainly a method out of many methods of some class. Generally the tests cases are written in the form of functions that will evaluate and determine whether a returned value after performing Unit Test is equals to the value you were expecting when you wrote the function. The main objective in unit testing is to isolate a unit part of code and validate its to correctness and reliable.   
  
There are few reasons that can give you a basic understanding of why a developer needs to design and write out test cases to make sure major requirements of a module are being validated during testing,

* Unit testing can increase confidence and certainty in changing and maintaining code in the development process.
* Unit testing always has the ability to find problems in early stages in the development cycle.
* Codes are more reusable, reliable and clean.
* Development becomes faster.
* Easy to automate.

Test class requirements

The minimum requirements for a test class while writing Unit Test case is given below:

* If you are using Unit Test to write test case then the [TestClass] attribute is highly required in the Microsoft unit testing framework for any class that contains unit test methods that you would like to run in Visual Studio Test Explorer.
* Each and every test method that you want to run must having the [TestMethod]attribute above it.

Test method requirements

A test method must meet the given requirements:

* The method must be defined with the [TestMethod] attribute just above method name.
* The method must having return type void.
* The method cannot have any parameters.

NUnit provides a rich set of assertion methods to verify the behavior of your code. Here are some commonly used NUnit assertion methods along with examples and explanations:

**1. Assert.AreEqual**

Verifies that two values are equal.

**C#**

[Test]

public void AreEqual\_Test()

{

int expected = 5;

int actual = 2 + 3;

Assert.AreEqual(expected, actual);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**2. Assert.AreNotEqual**

Verifies that two values are not equal.

**C#**

[Test]

public void AreNotEqual\_Test()

{

int expected = 5;

int actual = 2 + 2;

Assert.AreNotEqual(expected, actual);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**3. Assert.IsTrue**

Verifies that a condition is true.

**C#**

[Test]

public void IsTrue\_Test()

{

bool condition = 5 > 3;

Assert.IsTrue(condition);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**4. Assert.IsFalse**

Verifies that a condition is false.

**C#**

[Test]

public void IsFalse\_Test()

{

bool condition = 5 < 3;

Assert.IsFalse(condition);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**5. Assert.IsNull**

Verifies that an object is null.

**C#**

[Test]

public void IsNull\_Test()

{

object obj = null;

Assert.IsNull(obj);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**6. Assert.IsNotNull**

Verifies that an object is not null.

**C#**

[Test]

public void IsNotNull\_Test()

{

object obj = new object();

Assert.IsNotNull(obj);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**7. Assert.Throws**

Verifies that a specific exception is thrown.

**C#**

[Test]

public void Throws\_Test()

{

Assert.Throws<ArgumentNullException>(() => MethodThatThrows(null));

}

public void MethodThatThrows(string input)

{

if (input == null)

throw new ArgumentNullException(nameof(input));

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**8. Assert.DoesNotThrow**

Verifies that no exception is thrown.

**C#**

[Test]

public void DoesNotThrow\_Test()

{

Assert.DoesNotThrow(() => MethodThatDoesNotThrow());

}

public void MethodThatDoesNotThrow()

{

// No exception thrown here

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**9. Assert.AreSame**

Verifies that two object references refer to the same object.

**C#**

[Test]

public void AreSame\_Test()

{

var obj1 = new object();

var obj2 = obj1;

Assert.AreSame(obj1, obj2);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**10. Assert.AreNotSame**

Verifies that two object references do not refer to the same object.

**C#**

[Test]

public void AreNotSame\_Test()

{

var obj1 = new object();

var obj2 = new object();

Assert.AreNotSame(obj1, obj2);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**11. StringAssert.Contains**

Verifies that a string contains a specified substring.

**C#**

[Test]

public void StringContains\_Test()

{

string str = "Hello, world!";

StringAssert.Contains("world", str);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**12. StringAssert.StartsWith**

Verifies that a string starts with a specified substring.

**C#**

[Test]

public void StringStartsWith\_Test()

{

string str = "Hello, world!";

StringAssert.StartsWith("Hello", str);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**13. StringAssert.EndsWith**

Verifies that a string ends with a specified substring.

**C#**

[Test]

public void StringEndsWith\_Test()

{

string str = "Hello, world!";

StringAssert.EndsWith("world!", str);

}

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

**14. Assert.That**

A more flexible assertion method that uses constraints.

**C#**

[Test]

public void That\_Test()

{

int[] array = { 1, 2, 3 };

Assert.That(array, Has.Exactly(1).EqualTo(3));

}

EMPLOYEE.CS

namespace EmployeeLibrary

{

public class Employee

{

public string Email { get; set; }

public string Password { get; set; }

public bool IsEmailValid()

{

return Email != null && Email.EndsWith("@gmail.com");

}

public bool IsPasswordValid()

{

// Example validation: password should be at least 8 characters long

return Password != null && Password.Length >= 8;

}

public bool IsEmailAndPasswordValid()

{

return IsEmailValid() && IsPasswordValid();

}

}

}

Program.cs

using NUnit.Compatibility;

using DemoProject;

using NUnit.Framework;

using Assert = NUnit.Framework.Assert;

namespace DemoProjectTest

{

[TestFixture]

public class EmployeeTests

{

[Test]

public void IsEmailValid\_EmailEndsWithGmail\_ReturnsTrue()

{

// Arrange

var employee = new Employee { Email = "test@gmail.com", Password = "password123" };

// Act

var result = employee.IsEmailValid();

// Assert

Assert.IsTrue(result);

}

[TestCase("test@yahoo.com", "password123")]

[TestCase("test@yahoo.com", "password123")]

[TestCase("test@yahoo.com", "password123")]

public void IsEmailValid\_EmailDoesNotEndWithGmail\_ReturnsFalse(String email,String pwd)

{

// Arrange

var employee = new Employee { Email=email,Password=pwd };

// Act

var result = employee.IsEmailValid();

// Assert

Assert.IsFalse(result);

}

[Test]

public void Email\_ShouldNotBeNull()

{

// Arrange

var employee = new Employee { Email = "test@gmail.com", Password = "password123" };

// Act

var email = employee.Email;

// Assert

Assert.IsNotNull(email);

}

[Test]

public void Password\_ShouldBeEqual()

{

// Arrange

var employee = new Employee { Email = "test@gmail.com", Password = "password123" };

// Act

var password = employee.Password;

// Assert

Assert.AreEqual("password123", password);

}

[Test]

public void Email\_ShouldContainGmail()

{

// Arrange

var employee = new Employee { Email = "test@gmail.com", Password = "password123" };

// Act

var email = employee.Email;

// Assert

StringAssert.Contains("@gmail.com", email);

}

}

}