**Unit Testing Theory:**

**NUnit Theory:**

For detailed information about NUnit, please visit the NUnit Offical Website [NUnit.org](https://nunit.org/) .

**What Is NUnit?**

NUnit is a unit-testing framework for all .Net languages. Initially ported from JUnit, the current production release, version 3, has been completely rewritten with many new features and support for a wide range of .NET platforms. ( NUNIT is a framework used for testing .net code)

Advantages of NUNIT :

* It is used to increase the performance of the code by identify the bugs earlier.

**Attributes** (it is used identify the test)

NUnit uses custom attributes to identify tests. All NUnit attributes are contained in the NUnit.Framework namespace. Each source file that contains tests must include a using statement for that namespace and the project must reference the framework assembly, nunit.framework.dll.

This table lists most commonly used attributes in NUnit:

| **Attribute** | **Usage** |
| --- | --- |
| [Author Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/author.html) | Provides the name of the test author. |
| [Category Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/category.html) | Specifies one or more categories for the test. (to group the tests) |
| [Description Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/description.html) | Applies descriptive text to a Test, TestFixture or Assembly. |
| [Ignore Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/ignore.html) | Indicates that a test shouldn't be run for some reason. |
| [OneTimeSetUp Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/onetimesetup.html) | Identifies methods to be called once prior to any child tests.(it is called only once before all test methods) |
| [OneTimeTearDown Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/onetimeteardown.html) | Identifies methods to be called once after all child tests. .(it is called only once after all test methods) |
| [Order Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/order.html) | Specifies the order in which decorated test should be run within the containing fixture or suite. |
| [Property Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/property.html) | Allows setting named properties on any test case or fixture. |
| [SetUp Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/setup.html) | Indicates a method of a TestFixture called just before each test method.(it is called before each test method) |
| [TearDown Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/teardown.html) | Indicates a method of a TestFixture called just after each test method. .(it is called after each test method) |
| [Test Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/test.html) | Marks a method of a TestFixture that represents a test. (it is used to denote the test method) |
| [TestCase Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/testcase.html) | Marks a method with parameters as a test and provides inline arguments. |
| [TestCaseSource Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/testcasesource.html) | Marks a method with parameters as a test and provides a source of arguments. |
| [TestFixture Attribute](https://docs.nunit.org/articles/nunit/writing-tests/attributes/testfixture.html) | Marks a class as a test fixture and may provide inline constructor arguments.  (it is used denote the test class) |

Please visit the following URL for detailed list of all NUnit attributes:

b[Attributes | NUnit Docs](https://docs.nunit.org/articles/nunit/writing-tests/attributes.html)

**NUnit Assertion Models:**

.(by Assert.That method Assert is used to pass the objects as an argument to test method)

**Reference URL:** [Assertions | NUnit Docs](https://docs.nunit.org/articles/nunit/writing-tests/assertions/assertions.html)

In NUnit 3.0, assertions are written primarily using the Assert.That method, which takes constraint objects as an argument. We call this the Constraint Model of assertions

In earlier versions of NUnit, a separate method of the Assert class was used for each different assertion.

**Classic model:**

* Classic model has Single or multiple methods.
* This Classic Model is still supported but since no new features have been added to it for some time,
* the constraint-based model must be used in order to have full access to NUnit's capabilities.

For example, the following code must use the constraint model. There is no real classic equivalent.

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

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

Assert.That(array, Has.Exactly(2).GreaterThan(1));

Assert.That(array, Has.Exactly(3).LessThan(100));

Where equivalents do exist, the two approaches will always give the same result, because the methods of the classic approach have all been implemented internally using constraints. For example...

Assert.AreEqual(4, 2+2);

Assert.That(2+2, Is.EqualTo(4));

**NUnit Constraint Model:**

**Reference URL:** [Constraint Model (Assert.That) | NUnit Docs](https://docs.nunit.org/articles/nunit/writing-tests/assertions/assertion-models/constraint.html)

**Constraint Model (Assert.That)**

The constraint-based Assert model uses a single method of the Assert class for all assertions. The logic necessary to carry out each assertion is embedded in the constraint object passed as the second parameter to that method.

Here's a very simple assert using the constraint model:

Assert.That(myString, Is.EqualTo("Hello"));

The second argument in this assertion uses one of NUnit's syntax helpers to create an EqualConstraint. The same assertion could also be made in this form:

Assert.That(myString, new EqualConstraint("Hello"));

**NUnit Constraints:**

Reference URL: [Constraints | NUnit Docs](https://docs.nunit.org/articles/nunit/writing-tests/constraints/Constraints.html)

This page lists all the constraints available in NUnit.

* [Alphabetical List of Constraints](https://docs.nunit.org/articles/nunit/writing-tests/constraints/Constraints.html#alphabetical-list-of-constraints)
* [Constraints by Category](https://docs.nunit.org/articles/nunit/writing-tests/constraints/Constraints.html#constraints-by-category)

You can click on the constraint name in the below table to read more about the constraint. You can go though the constraints covered during the training.

Alphabetical List of Constraints

| **Constraint Name** |
| --- |
| [AllItemsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AllItemsConstraint.html) |
| [AndConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AndConstraint.html) |
| [AssignableFromConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AssignableFromConstraint.html) |
| [AssignableToConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AssignableToConstraint.html) |
| [AttributeConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AttributeConstraint.html) |
| [AttributeExistsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AttributeExistsConstraint.html) |
| [AnyOfConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AnyOfConstraint.html) |
| [BinarySerializableConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/BinarySerializableConstraint.html) |
| [CollectionContainsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionContainsConstraint.html) |
| [CollectionEquivalentConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionEquivalentConstraint.html) |
| [CollectionOrderedConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionOrderedConstraint.html) |
| [CollectionSubsetConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionSubsetConstraint.html) |
| [CollectionSupersetConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionSupersetConstraint.html) |
| [DelayedConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/DelayedConstraint.html) |
| [DictionaryContainsKeyConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/DictionaryContainsKeyConstraint.html) |
| [DictionaryContainsKeyValuePairConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/DictionaryContainsKeyValuePairConstraint.html) |
| [DictionaryContainsValueConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/DictionaryContainsValueConstraint.html) |
| [EmptyCollectionConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyCollectionConstraint.html) |
| [EmptyConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyConstraint.html) |
| [EmptyDirectoryConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyDirectoryConstraint.html) |
| [EmptyStringConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyStringConstraint.html) |
| [EndsWithConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EndsWithConstraint.html) |
| [EqualConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EqualConstraint.html) |
| [ExactCountConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/ExactCountConstraint.html) |
| [ExactTypeConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/ExactTypeConstraint.html) |
| [FalseConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/FalseConstraint.html) |
| [FileOrDirectoryExistsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/FileOrDirectoryExistsConstraint.html) |
| [GreaterThanConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/GreaterThanConstraint.html) |
| [GreaterThanOrEqualConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/GreaterThanOrEqualConstraint.html) |
| [InstanceOfTypeConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/InstanceOfTypeConstraint.html) |
| [LessThanConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/LessThanConstraint.html) |
| [LessThanOrEqualConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/LessThanOrEqualConstraint.html) |
| [NaNConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NaNConstraint.html) |
| [NoItemConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NoItemConstraint.html) |
| [NotConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NotConstraint.html) |
| [NullConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NullConstraint.html) |
| [OrConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/OrConstraint.html) |
| [PropertyConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/PropertyConstraint.html) |
| [PropertyExistsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/PropertyExistsConstraint.html) |
| [RangeConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/RangeConstraint.html) |
| [RegexConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/RegexConstraint.html) |
| [ReusableConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/ReusableConstraint.html) |
| [SameAsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SameAsConstraint.html) |
| [SamePathConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SamePathConstraint.html) |
| [SamePathOrUnderConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SamePathOrUnderConstraint.html) |
| [SomeItemsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SomeItemsConstraint.html) |
| [StartsWithConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/StartsWithConstraint.html) |
| [SubPathConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SubPathConstraint.html) |
| [SubstringConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SubstringConstraint.html) |
| [ThrowsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/ThrowsConstraint.html) |
| [ThrowsNothingConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/ThrowsNothingConstraint.html) |
| [TrueConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/TrueConstraint.html) |
| [UniqueItemsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/UniqueItemsConstraint.html) |
| [XmlSerializableConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/XmlSerializableConstraint.html) |

Constraints by Category

Collection Constraints

| **Constraint Name** |
| --- |
| [AllItemsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AllItemsConstraint.html) |
| [AnyOfConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AnyOfConstraint.html) |
| [CollectionContainsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionContainsConstraint.html) |
| [CollectionEquivalentConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionEquivalentConstraint.html) |
| [CollectionOrderedConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionOrderedConstraint.html) |
| [CollectionSubsetConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionSubsetConstraint.html) |
| [CollectionSupersetConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/CollectionSupersetConstraint.html) |
| [DictionaryContainsKeyConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/DictionaryContainsKeyConstraint.html) |
| [DictionaryContainsKeyValuePairConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/DictionaryContainsKeyValuePairConstraint.html) |
| [DictionaryContainsValueConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/DictionaryContainsValueConstraint.html) |
| [EmptyCollectionConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyCollectionConstraint.html) |
| [ExactCountConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/ExactCountConstraint.html) |
| [NoItemConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NoItemConstraint.html) |
| [SomeItemsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SomeItemsConstraint.html) |
| [UniqueItemsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/UniqueItemsConstraint.html) |

Comparison Constraints

| **Constraint Name** |
| --- |
| [GreaterThanConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/GreaterThanConstraint.html) |
| [GreaterThanOrEqualConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/GreaterThanOrEqualConstraint.html) |
| [LessThanConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/LessThanConstraint.html) |
| [LessThanOrEqualConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/LessThanOrEqualConstraint.html) |
| [RangeConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/RangeConstraint.html) |

Compound Constraints

| **Constraint Name** |
| --- |
| [AndConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AndConstraint.html) |
| [NotConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NotConstraint.html) |
| [OrConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/OrConstraint.html) |

Condition Constraints

| **Constraint Name** |
| --- |
| [EmptyConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyConstraint.html) |
| [FalseConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/FalseConstraint.html) |
| [NaNConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NaNConstraint.html) |
| [NullConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/NullConstraint.html) |
| [TrueConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/TrueConstraint.html) |

File and Directory Constraints

| **Constraint Name** |
| --- |
| [EmptyDirectoryConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyDirectoryConstraint.html) |
| [FileOrDirectoryExistsConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/FileOrDirectoryExistsConstraint.html) |
| [SamePathConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SamePathConstraint.html) |
| [SamePathOrUnderConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SamePathOrUnderConstraint.html) |
| [SubPathConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SubPathConstraint.html) |

String Constraints

| **Constraint Name** |
| --- |
| [EmptyStringConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EmptyStringConstraint.html) |
| [EndsWithConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/EndsWithConstraint.html) |
| [RegexConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/RegexConstraint.html) |
| [StartsWithConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/StartsWithConstraint.html) |
| [SubstringConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/SubstringConstraint.html) |

Type Constraints

| **Constraint Name** |
| --- |
| [AssignableFromConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AssignableFromConstraint.html) |
| [AssignableToConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/AssignableToConstraint.html) |
| [ExactTypeConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/ExactTypeConstraint.html) |
| [InstanceOfTypeConstraint](https://docs.nunit.org/articles/nunit/writing-tests/constraints/InstanceOfTypeConstraint.html) |

**Test Filters: Test filters represent a selection of test to be displayed or run or loaded.**

* **<filter,and,or,not,id,test,cat,class,method,namespace,prop,name>**

**Parameterized test (or) data-driven testing : sometimes test method needs to be executed using multiple parameters.**

**2 attributes:**

1. **TestCase : it is used to pass the parameters to the test methods.**
2. **TestCaseSourse : to specify from where to get the parameters.**

**Test-driven development : It is an approach to software development in which the unit tests are written before the business logic ,in other words before implementing the code developers write unit test.**

**TDD mantra : RED ,GREEN, REFRACTOR**

**How to perform TDD Test**

* Add a test.
* Run all tests and see if any new test fails.
* Write some code.
* Run tests and Refactor code.
* Repeat

**Reference URL:** [Test-driven development - Wikipedia](https://en.wikipedia.org/wiki/Test-driven_development)

Test-driven development (TDD) is a software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases. This is as opposed to software being developed first and test cases created later.

**Test-driven development cycle**

**1. Add a test**

The adding of a new feature begins by writing a test that passes if the feature's specifications are met. The developer can discover these specifications by asking about use cases and user stories. A key benefit of test-driven development is that it makes the developer focus on requirements before writing code. This is in contrast with the usual practice, where unit tests are only written after code.

**2. Run all tests.**

The new test should fail for expected reasons

This shows that new code is actually needed for the desired feature. It validates that the test harness is working correctly. It rules out the possibility that the new test is flawed and will always pass.

**3. Write the simplest code that passes the new test**

Inelegant or hard code is acceptable, as long as it passes the test. The code will be honed anyway in Step 5. No code should be added beyond the tested functionality.

**4. All tests should now pass**

If any fail, the new code must be revised until they pass. This ensures the new code meets the test requirements and does not break existing features.

**5. Refactor as needed**, using tests after each refactor to ensure that functionality is preserved

Code is refactored for readability and maintainability. In particular, hard-coded test data should be removed. Running the test suite after each refactor helps ensure that no existing functionality is broken.

**Examples of refactoring:**

moving code to where it most logically belongs

removing duplicate code

making names self-documenting

splitting methods into smaller pieces

re-arranging inheritance hierarchies

**Repeat**

The cycle above is repeated for each new piece of functionality. Tests should be small and incremental, and commits made often. That way, if new code fails some tests, the programmer can simply undo or revert rather than debug excessively. When using external libraries, it is important not to write tests that are so small as to effectively test merely the library itself unless there is some reason to believe that the library is buggy or not feature-rich enough to serve all the needs of the software under development cycle.

**MOQ :**

**Integration testing: Integration testing means checking if different modules are working fine when combined together as a group.**

**What is Mocking: duplicate code**

**Reference URL:** [unit testing - What is Mocking? - Stack Overflow](https://stackoverflow.com/questions/2665812/what-is-mocking)

Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behavior of the object you want to replace the other objects by mocks that simulate the behavior of the real objects. This is useful if the real objects are impractical to incorporate into the unit test.

In short, mocking is creating objects that simulate the behavior of real objects.

**Stub**: it is a small piece of code that takes the place of another component during testing.

At times you may want to distinguish between mocking as opposed to stubbing. There may be some disagreement about this subject but my definition of a stub is a "minimal" simulated object. The stub implements just enough behavior to allow the object under test to execute the test.

A mock is like a stub but the test will also verify that the object under test calls the mock as expected. Part of the test is verifying that the mock was used correctly.

To give an example: You can stub a database by implementing a simple in-memory structure for storing records. The object under test can then read and write records to the database stub to allow it to execute the test. This could test some behavior of the object not related to the database and the database stub would be included just to let the test run.

If you instead want to verify that the object under test writes some specific data to the database you will have to mock the database. Your test would then incorporate assertions about what was written to the database mock.

[Method calls - Unit Testing in C# (educationsmediagroup.com)](https://docs.educationsmediagroup.com/unit-testing-csharp/moq/method-calls)