**Unit Testing Frameworks**

Contents

[1 Document Control 3](#_Toc98418136)

[1.1 Change Record 3](#_Toc98418137)

[1.2 Reviewer 3](#_Toc98418138)

[1.3 Approver 3](#_Toc98418139)

[2 Document Purpose 3](#_Toc98418140)

[3 Testing Frameworks 3](#_Toc98418141)

[3.1 NUnit 3](#_Toc98418142)

[3.2 XUnit 4](#_Toc98418143)

[3.3 MSTest 4](#_Toc98418144)

[4 Core Differences 4](#_Toc98418145)

[4.1 Isolation of Tests 4](#_Toc98418146)

[4.2 Extensibility 5](#_Toc98418147)

[4.3 Initialization and De-initialization 5](#_Toc98418148)

[4.4 Assertion Mechanism 5](#_Toc98418149)

[4.5 Parallel Test Execution 5](#_Toc98418150)

[5 Comparison 6](#_Toc98418151)

[6 Conclusion 7](#_Toc98418152)

[7 Appendix 7](#_Toc98418153)

# Document Control

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Change reference** |
| 03/17/2022 | Prajeesh T S | 1.1 | Initial version |

## Reviewer

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Approval/Review Date** |
| **Anoop Jose** | Staff Software Architect |  |

## Approver

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Approval/Review Date** |
| **Anoop Jose** | Staff Software Architect |  |

# Document Purpose

This document provides details regarding details of available unit testing frameworks in C# and its comparison.

# Testing Frameworks

## NUnit

Nunit s an open-source testing framework ported from Junit. The latest version of NUnit is NUnit3 that is rewritten with many new features and has support for a wide range of .NET platforms.

NUnit has been downloaded more than 126 million times from NuGet.org. This shows the popularity of NUnit within the .NET user community. As of writing this article, there were close to 24000 questions tagged as NUnit on Stack overflow.

Using NUnit framework, tests can be executed serially as well as parallel. Parallel test execution is possible at assembly, class, or method level. When parallelism is enabled, by default four tests can be executed in parallel.

## XUnit

XUnit is an open-source testing framework based on the .NET framework. ‘x’ stands for the programming language, e.g., JUnit, NUnit, etc. The creators of NUnit created xUnit as they wanted to build a better framework rather than adding incremental features to the NUnit framework.

XUnit is created with more focus on the community; hence it is easy to expand upon.

xUnit is much more extensible when compared to NUnit and MSTest. The [Fact] attribute is used instead of the [Test] attribute. Non-parameterized tests are implemented under the [Fact] attribute, whereas the [Theory] attribute is used if you plan to use parameterized tests.

##### 

## MSTest

MSTest is the default test framework that is shipped along with Visual Studio. The initial version of MSTest (V1) was not open-source; however, MSTest V2 is open-source.

MSTest V2 has cross-platform support and can be extended using custom test attributes & custom asserts. Parallel test execution is supported by the MSTest framework where parallelism is possible at the Method level or Class level.

# Core Differences

## Isolation of Tests

XUnit framework provides much better isolation of tests in comparison to NUnit and MSTest frameworks. For each test case, the test class is instantiated, executed, and is discarded after the execution. This ensures that the tests can be executed in any order as there is reduced/no dependency between the tests. Executing each test as a separate instance minimizes the chances of one test causing the other tests to fail.

## Extensibility

When we do NUnit vs. XUnit vs. MSTest, extensibility plays an important role in choosing a particular test framework. The choice might depend on the needs of the project, but in some scenarios, extensibility can turn the tables around for a particular test framework. When compared to MSTest and NUnit frameworks, xUnit framework is more extensible since it makes use of [Fact] and [Theory] attributes. Many attributes that were present in NUnit framework e.g. [TestFixture], [TestFixtureSetup], [TestFixtureTearDown] [ClassCleanup], [ClassInitialize], [TestCleanup], etc. are not included in the xUnit framework.

## Initialization and De-initialization

The NUnit uses [SetUp], [TearDown] pairs whereas MSTest uses [TestInitialize], [TestCleanup] pairs for setting up the activities related to initialization & de-initialization of the test code.

On the other hand, xUnit uses the class constructor for the implementation of steps related to test initialization and IDisposable interface for the implementation of steps related to de-initialization.

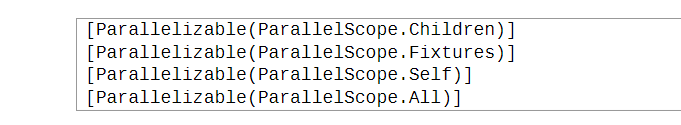
## Assertion Mechanism

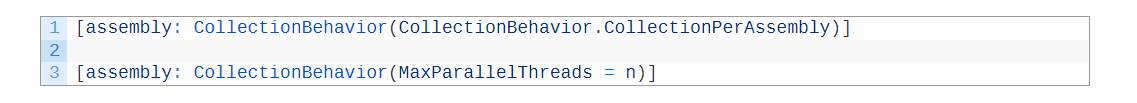
xUnit framework makes use of Assert.Throws instead of [ExpectedException] which is used in NUnit and MSTest. The drawback of using [ExpectedException] is that the errors might not be reported if they occur in the wrong part of the code. For example, if assert has to be raised for Security Exception, but Authentication Exception occurs, [ExpectedCondition] will not raise assert.

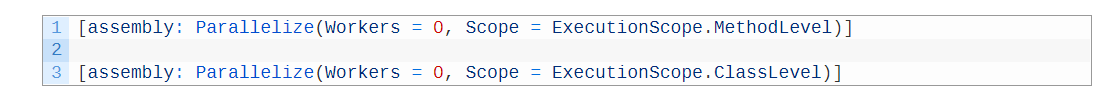
## Parallel Test Execution

All the three C# unit testing frameworks support parallel test execution and are well-suited for Selenium automation testing as throughput plays a major role in automation testing. Below are the ways in which parallelism can be achieved in each of the test frameworks.

* NUnit – Parallelism is possible at the level of Children (child tests are executed in parallel with other tests), Fixtures (descendants of test till the level of Test Fixtures can execute in parallel), Self (test itself can be executed in parallel with other tests), and All (test & its descendants can execute in parallel with others at the same level)



* xUnit – Parallelism by putting the test classes into a single test collection or by executing an ‘n’ number of threads in parallel. ****
* **MSTest –** Parallelism at Method level as well as Class level



# Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **NUnit** | **MSTest** | **XUnit** |
| License | MIT | Creative Commons Attribution 3.0 International Public License | MIT |
| Performance | Good | Average | Good |
| Isolated Tests | Yes | Yes | Yes |
| Extensible test attributes | NO (Test and Test Case attributes are sealed) | NO | YES (Theory and Fact attributes are extensible) |
| Parallel test execution | Yes [Children, Fixtures and All] | Yes [Method and Class Level] | Yes [Assembly Level] |
|  |  |  |  |
| Marks a test method/individual test | [Test] | [TestMethod] | [Fact] |
| Indicates that a class has group of tests | [TextFixture] | [TestClass] | N/A |
| Contains the initialization code which is triggered before each test | [Setup] | [TestInitialize] | Constructor |
| Contains the clean-up code which is triggered after each test | [TearDown] | [TestCleanup] | IDisposable.Dispose |
| Contains per collection fixture setup and tear down | N/A | N/A | ICollectionFicture<T> |
| Ignoretest case | [Ignore(“reason”)] | [Ignore] | [Fact(Skip=”reason”)] |
| Categorize test cases or classes | [Category(“”)] | [TestCatagory(“”)] | [Trait(“Category”)] |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Conclusion

While most of the other testing frameworks mentioned are all pretty much the same, xUnit.NET has taken a pretty unique, modern, and flexible approach to unit testing. It changes terminology, so you we longer define TestFixtures and Tests...we specify Facts and Theories about our code, which integrates better with the concept of what a test is from a TDD/BDD perspective.

xUnit.NET is also EXTREMELY extensible. Its FactAttribute and TraitAttribute attribute classes are not sealed, and provide overridable base methods that give us a lot of control over how the methods those attributes decorate should be executed. While xUnit.NET in its default form allows us to write test classes that are similar to NUnit test fixtures with their test methods, we are not confined to this form of unit testing at all. We are free to extend the framework to support BDD-style Concern/Context/Observation specifications, as depicted.

xUnit.NET also supports fit-style testing directly out of the box with its Theory attribute and corresponding data attributes. Fit input data may be loaded from excel, database, or even a custom data source such as a Word document (by extending the base data attribute.) This allows to capitalize on a single testing platform for both unit tests and integration tests, which can be huge in reducing product dependencies and required training.

#### 

# Appendix

* [NUnit vs. XUnit vs. MSTest: Comparing Unit Testing Frameworks In C# - DEV Community](https://dev.to/himanshusheth004/nunit-vs-xunit-vs-mstest-comparing-unit-testing-frameworks-in-c-24ha)
* [NUnit vs. XUnit vs. MSTest: Comparing Unit Testing Frameworks In C# (lambdatest.com)](https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/)