**Csunit**

* Standard unit testing features such as test, fixture, setup, teardown, ignore, expected exception, etc.
* Easy to use graphical user interface
* Recipes for combining several test assemblies into one test suite
* Addin for Visual Studio 2005 and Visual Studio 2008
* Executes tests based on the following frameworks:
  + csUnit
  + NUnit (version 2.4.7, .NET 2.0)
  + Microsoft unit testing (basic support, VS 2008)
* Search capabilites across tests, output, and statistics
* Statistics per test to create performance base line
* Categories to group tests for execution
* Support for applications targeting .NET 2.0, .NET 3.0, and .NET 3.5
* Works with any .NET language (C#, VB.NET, Managed C++, etc.)
* Runs on Windows Vista SP 1 or Windows XP SP 3
*  [www.csunit.org](http://www.csunit.org)
* 73 000 stiahnutí na sourceforge od roku 2002, najviac v 2007, za posledné 3 roky len necelých 3 000

**Fixie**

[Fixie](http://fixie.github.io/) is a relative newcomer to the .NET testing framework space. Just as with other testing frameworks such as NUnit or xUnit.net, Fixie allows test methods to be created and executed. The difference with Fixie is that it takes a conventions-based approach, which is a benefit as we do not need to use attributes to mark classes and methods as tests.

<https://visualstudiomagazine.com/articles/2015/04/22/fixie-c-sharp-testing.aspx>

**Mbunit**

https://code.google.com/p/mb-unit/  
http://stackoverflow.com/questions/3678783/mbunit-vs-nunit

The Gallio Automation Platform is an open, extensible, and neutral system for .NET that provides a common object model, runtime services and tools (such as test runners) that may be leveraged by any number of test frameworks.

MbUnit is an extensible unit testing framework for the .NET Framework that takes in and goes beyond xUnit pattern testing. MbUnit is part of the Gallio bundle.

Even though NUnit now includes the most popular MbUnit advanced features, MbUnit is still more feature-rich, for example:

* [Contract verifiers](http://gallio.org/wiki/doku.php?id=mbunit%3acontract_verifiers)
* [XML assertions](http://vkreynin.wordpress.com/2010/07/18/test/)
* [Serialization assertions](http://gallio.org/wiki/doku.php?id=mbunit%3aassertions#serialization)
* [External data sources (CSV, XML, etc)](http://gallio.org/wiki/doku.php?id=mbunit%3adata-driven_testing%3aexternal_data_sources)
* [Parallelizable tests](http://blog.bits-in-motion.com/2009/03/announcing-gallio-and-mbunit-v306.html)

Gallio/MbUnit has still many features which do not exist in NUnit yet or in any other testing framework AFAIK: contract verifiers, combinatorial data tests, xml and serialization assertions, structural equality comparer, custom behaviors, etc.

**Moq**

**What is Moq?**  
[Moq](http://www.clariusconsulting.net) is promoted by its creators as easier to learn and use than other Mock Object Frameworks such as Rhino Mocks and TypeMock Isolator. Moq is the only mocking library for .NET developed to take advantage of Linq expression trees and lambda expressions. Thus making it the most productive, type-safe and refactoring-friendly mocking library available.

Moq supports mocking interfaces as well as classes. There are some requirements and limitations on classes. The class can’t be sealed. Also, the method being mocked must be marked as virtual. You cannot mock static methods.

**Moq Features**  
Moq offers the following features according to [Moq’s Google code web site](http://code.google.com/p/moq)

1. Strong-typed: no strings for expectations, no object-typed return values or constraints  
2. Unsurpassed Visual Studio intellisense integration: everything supports full Visual Studio intellisense, from setting expectations, to specifying method call arguments, return values, etc.  
3. No Record/Replay idioms to learn. Just construct your mock, set it up, use it and optionally verify calls to it. You may not verify mocks when they act as stubs only, or when you are doing more classic state-based testing by checking returned values from the object under test.  
4. Granular control over mock behavior with a simple MockBehavior enumeration. No need to learn what’s the theoretical difference between a mock, a stub, a fake, a dynamic mock, etc.  
5. Mock boths interfaces and classes  
6. Override expectations: can set default expectations in a fixture setup, and override as need on tests  
7. Pass constructor arguments for mocked classes  
8. Intercept and raise events on mocks  
9. Intuitive support for out/ref argument

<http://blogs.interknowlogy.com/2012/02/26/introduction-to-moq-framework/>  
<http://www.agile-code.com/blog/mocking-with-moq/>

**NBi**

**Just plain xml**

No need of C#, Java or any other development language to test your BI Solution. Your tests are written in xml by the means of a useful and intuitive syntax.

**All your components**

NBi helps you to create tests targeting your databases, cubes, etls and reports with a *unique* framework. If you're working the Microsoft BI stack, it means that you can test your SQL Server Database, SSIS, SSAS and SSRS components with this framework.

**Automate generation**

NBi is shipped with a tool to facilitate the *automation of creation of test-cases*. Build hundreds tests in a couple of minutes and facilitate the maintenance of these tests.

http://www.nbi.io/

The main goal of this framework is to let users create tests with a declarative approach based on an Xml syntax. By the means of NBi, you don't need to develop C# code to specify your tests! Either, you don't need Visual Studio to compile your test suite. Just create an Xml file and let the framework interpret it and play your tests. The framework is designed as an add-on of NUnit but with the possibility to port it easily to other testing frameworks.

<http://fr.slideshare.net/CdricCharlier1>

<https://github.com/Seddryck/NBi>

https://seddryck.wordpress.com/category/quality/testing/nbi/

**NUnit**

<https://en.wikipedia.org/wiki/NUnit>

## NUnit Features

In order to test an application under NUnit, you write test code that is specially annotated using custom [Attributes](http://www.nunit.org/index.php?p=attributes&r=2.2.10). Your test code contains [Assertions](http://www.nunit.org/index.php?p=assertions&r=2.2.10), which demonstrate the correct working of the application.

If your application stores settings in [Configuration Files](http://www.nunit.org/index.php?p=configFiles&r=2.2.10), NUnit provides you with the ability to have settings for your test, which are different from those used in production.

In addition to running tests in a single assembly, NUnit provides support for tests organized as [Multiple Assemblies](http://www.nunit.org/index.php?p=multiAssembly&r=2.2.10) and for creating and running tests as [NUnit Test Projects](http://www.nunit.org/index.php?p=multiAssembly&r=2.2.10).

For those using NUnit on a Windows system with Visual Studio installed, [Visual Studio Support](http://www.nunit.org/index.php?p=vsSupport&r=2.2.10) is available.

<http://www.nunit.org/index.php?p=features&r=2.2.10>

<http://www.slideshare.net/ShirBrass/nunit-features-presentation>

<https://lukewickstead.wordpress.com/2013/02/09/howto-nunit-features/>

**Pex**

**Pex automatically generates test suites with high code coverage.** Right from the Visual Studio code editor, Pex finds interesting input-output values of your methods, which you can save as a small test suite with high code coverage. Microsoft Pex is a Visual Studio add-in for testing .NET Framework applications.

<http://www.pexforfun.com/Documentation.aspx#HowDoesPexWork>  
<http://stackoverflow.com/questions/2704669/is-pex-test-generation-really-useful-tool>

<http://www.codeproject.com/Articles/583520/UsingplusPexplusandplusMicrosoftplusCodeplusDigger>

Pex is one of the many wonderful things going on at Microsoft Research and it is intended to a tool to assist with automating white-box and unit-testing. It can help with generating all kinds of different inputs that can be thrown at a specific set of code and will display each of these execution branches along with the corresponding output of the function. It can provide an easy way for those that aren’t crazy about writing unit tests (or just aren’t very good at it) to simply test their code by letting the Pex Engine run through it.

Pex can provide an excellent way for you to find those small edge-cases that can so often plague software and the fact that the process is completely automated makes it even easier!

**Randoop**

feedback-directed random test generation, which generates each test, evaluates it, and determines whether to use it as a foundation for more tests. Randoop uses this technique.

+ pdf v downe

**Rhino Mocks**

Rhino Mocks will generate fake objects to replace the dependencies that you have, and then allow you to tell them, at runtime, how to behave. This functionality is very powerful, and it means that you can tell your fake objects, for each test, how to behave.

Rhino Mocks is a dynamic mock object framework for the .Net platform. Its purpose is to ease testing by allowing the developer to create mock implementations of custom objects and verify the interactions using unit testing.

http://www.wrightfully.com/using-rhino-mocks-quick-guide-to-generating-mocks-and-stubs/

**Specflow**

## What is SpecFlow?

SpecFlow allows .NET development teams to define, manage and execute automated acceptance tests as business readable specifications. It is based on [Gherkin](https://github.com/cucumber/gherkin/wiki) and part of the [Cucumber](https://cucumber.io/) eco system.

SpecFlow aims to bridge the communication gap between domain experts and developers. Acceptance tests in SpecFlow follow the BDD paradigm of defining specifications with examples, so that they are also understandable to business users. Acceptance tests can then be tested automatically as needed, while their specification serves as a living documentation of the system.

SpecFlow integrates with Visual Studio, but can be also used from the command line (e.g. on a build server).

[SpecFlow+](http://www.specflow.org/plus/) adds additional functionality to SpecFlow, such as a stand-alone IDE for Gherkin specifications, advanced reporting and much more.

<http://www.specflow.org/documentation/FAQ/>

[SpecFlow](http://specflow.org/) is an open-source .NET tool that lets you write specifications using 100%-Cucumber-compatible Gherkin syntax, and has a number of advantages over Cucumber itself:

* *It integrates with Visual Studio*, which means you get File->New templates for creating new feature files
* More crucially, *it gives complete VS debugger support*, so you can set breakpoints on Given/When/Then lines in your .feature files and step through their execution
* You can implement your step definitions in *any .NET language*
* When you compile a project containing SpecFlow feature files, the output is an NUnit test assembly, so you can *use your favourite NUnit-compatible test runner or existing CI infrastructure* to run the specifications with no additional configuration.
* *It doesn’t have such a ridiculous name* :)It’s sad, but at one of my recent clients, several managers refused to take Cucumber seriously and wouldn’t pay attention to Cucumber specifications purely because of the name. That’s the real world for you!

<http://blog.stevensanderson.com/2010/03/03/behavior-driven-development-bdd-with-specflow-and-aspnet-mvc/>

**Specter**

Specter is an object-behaviour specification framework for .NET.   
It enables behaviour-driven development by requiring developers to write executable specifications for their objects, before actually implementing them.   
Technologically this is similar to test driven development, however the shift in nomenclature removes the psychological barrier of writing "tests" for code that does not exist. (Existing projects implementing this idea include RSpec for Ruby and NSpec for .NET)

Specter uses Boo meta-programming features and therefore allows very readable specifications to be written.

[**https://www.openhub.net/p/10342**](https://www.openhub.net/p/10342)

* Readable specification code [example](http://specter.sourceforge.net/examples/MiniBarSpec.boo.html)
* Specifications can be compiled even if the objects specified (the subjects) are not implemented yet
* Standalone specification runners
* Integration with existing NUnit tools

<http://specter.sourceforge.net/>

**TickSpec**

**Project Description**  
A lightweight Behaviour Driven Development (BDD) framework. Describe behaviour in plain text using the Gherkin business language, i.e. given, when, then. Easily execute the behaviour against matching F# tick methods (let ``tick method`` () = true) or attributed C# or F# methods.

[**https://tickspec.codeplex.com/**](https://tickspec.codeplex.com/)



[**http://www.slideshare.net/ptrelford/bdd-with-tick-spec**](http://www.slideshare.net/ptrelford/bdd-with-tick-spec)

**TPT**

<https://www.piketec.com/products/tpt.php>

### TPT - Time Partition Testing Tool

TPT is a model-based testing tool for testing embedded systems, especially the testing of control systems. TPT supports all important fields of the test process which are in detail the management of tests, test modelling, test execution, test assessment, and test documentation.

<https://www.polelink.com/index.php?lang=en&Itemid=245>

<https://en.wikipedia.org/wiki/TPT_%28software%29>

## Typemock

### The Next Unit Test Generation

It's our most powerful feature ever. And it's the first of its kind. Typemock Suggest creates test suggestions suitable for your code.  
Imagine having a suite of unit test that cover your brilliantly written code. Written at only a fraction of the time.  
  
Typemock Suggest can also cover your Legacy Code.

## Learns your code. Learns your changes

It doesn't seem possible… but it is. Typemock suggest analyzes your code to suggests relevant unit tests. These tests follow best practices. Arrange-Act-Assert. Isolations. Assertions.

<http://www.typemock.com/suggest>

**Visual T#**

<https://visualstudiogallery.msdn.microsoft.com/e1286265-56d4-4c34-ad58-5230d3f3d427>

**Visual T#** has been developed to simplify the design and the coding of unit tests for Microsoft.NET.

**xUnit.net**

xUnit.net is a free, open source, community-focused unit testing tool for the .NET Framework. Written by the original inventor of NUnit v2, xUnit.net is the latest technology for unit testing C#, F#, VB.NET and other .NET languages. xUnit.net works with ReSharper, CodeRush, TestDriven.NET and Xamarin. It is part of the [ASP.NET Open Source Gallery](http://www.outercurve.org/Galleries/ASPNETOpenSourceGallery) under the [Outercurve Foundation](http://www.outercurve.org/), licensed under [Apache 2](http://opensource.org/licenses/Apache-2.0) (an OSI approved license).

<https://xunit.github.io/>

Learn the latest in unit testing technology for C#, VB.NET (and other .NET languages) created by the original inventor of NUnit. xUnit.net is a free, extensible, open source framework designed for programmers that aligns more closely with the .NET platform.

<http://www.pluralsight.com/courses/description/xunitdotnet-test-framework>

### Great Community & Active Development

xUnit.net is free and open source. The code is hosted on github (800 commits, 30 contibutors) and was previously on codeplex. The core team is made of inspired evangelists and very active. The official twitter account has 1400 followers and it’s still growing. xUnit is extensible and a lot of extensions like AutoFixture are already available on nuget.org.

<http://www.codeproject.com/Articles/1011753/Moving-to-xUnit-net>

http://blog.ploeh.dk/2010/04/26/WhyImmigratingfromMSTesttoxUnit.net/