NUnit vs. MbUnit vs. MSTest vs. xUnit.net [closed]

Asked 11 years ago Active 6 months ago Viewed 113k times

384

125

There are quite a lot of unittesting frameworks out there for .NET. I found this little feature comparison: http://xunit.github.io/docs/comparisons.html

Now I am to choose the best one for us. But how? Does it matter? Which one is most future proof and has a decent momentum behind it? Should I care about the features? While xUnit seems to be most modern and specifically designed for .NET, NUnit again seems to be the one that is widely accepted. MSTest again is already integrated into Visual Studio ...

unit-testing nunit mstest mbunit xunit.net

edited Oct 26 '15 at 19:20
Jim Bolla
7,323 25 49

asked Nov 4 '08 at 7:20
bitbonk
25.4k 28 160 256

closed as not constructive by casperOne Nov 18 '12 at 14:39

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

If this question can be reworded to fit the rules in the help center, please edit the question.

11 That comparison table is years out of date. For example, NUnit also has Assert.Throws etc, and everything in the Assertions table is the old API. The new Assert.That(..., Is....) fluent syntax is much nicer, and has been around for a good while now. – Jim Cooper May 18 '11 at 11:11

11 Do you know of any table that is more up to date? – bitbonk May 18 '11 at 13:49

1 Late 2013, moved from xUnit.net => NUnit. Also note that xUnit.NET (the project) != xUnit (the category, of which NUnit is a member) – DeepSpace101 Oct 30 '13 at 23:38

3 @Sid why you moved from xUnity.net => NUnit? – Alexander Logger Nov 3 '13 at 11:25

2 Similar question asked in 2014 Visual Studio 2013 MSTest vs NUnit – Michael Freidgeim Feb 3 '17 at 10:21

7 Answers

196

I know this is an old thread, but I thought I'd post a vote for xUnit.NET. 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 no longer define TestFixtures and Tests...you specify Facts and Theories about your 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 you a lot of control over how the methods those attributes decorate should be executed. While xUnit.NET in its default form allows you to write test classes that are similar to NUnit test fixtures with their test methods, you are not confined to this form of unit testing at all. You are free to extend the framework to support BDD-style Concern/Context/Observation specifications, as depicted here.

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 you 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.

Other approaches to testing may also be implemented with xUnit.NET...the possibilities are pretty limitless. Combined with another very forward looking mocking framework, Moq, the two create a very flexible, extensible, and powerful platform for implementing automated testing.

edited May 6 at 18:55

clows
157 4 11

answered Jun 5 '09 at 20:18

jrista
28k 10 79 123

34 While this was true a year+ ago, NUnit has since added most of the attributes in question. In NUnit you can write tests either way. – Mark Levison Nov 16 '10 at 16:21

8 Its not so much about which attributes are available, but how they can be used. xUnit.NET was designed from the ground up to be a highly flexible and extensible framework that did not lock you into any particular method of testing, and does not require you to regularly update the core framework to get the latest capabilities. – jrista Nov 22 '10 at 4:57

9 1. Sightly different names on attributes won't make much of a point. 2. NUnit was extensible and it continues to be extensible :? 3. data row parameters for tests are supported in nunit. Time ago they were supported in an extension :) 4. Nunit combined with Moq creates the same thing. 5. For BDD I'd say specflow, that integrates easily with many unit testing frameworks. – graffic Dec 1 '11 at 11:02

8 I like the sound of xUnit, however it has zilch documentation :(– Colonel Panic Aug 9 '12 at 13:34

9 xUnit has no documentation whatsoever! eg: Try finding what `Trait` really does or if you can group different tests within single parent test (eg. all `tests` within a `testfixture`). nUnit creates a great hierarchical view instead of xUnit's flat view of tests. Plus the nomenclature makes no sense - facts and theory? Be realistic! Those are better called tests and data. – DeepSpace101 Nov 20 '12 at 7:53

133

NUnit is probably the most supported by the 3rd party tools. It's also been around longer than the other three.

I personally don't care much about unit test frameworks, mocking libraries are IMHO much more important (and lock you in much more). Just pick one and stick with it.

answered Nov 4 '08 at 7:29

Alexander Kojevnikov
15.8k 5 44 45

3 what is your top mock library pick? – dplante Jun 5 '09 at 20:32

30 I like Moq, RhinoMocks is also good. – Alexander Kojevnikov Jun 8 '09 at 7:27

5 It might also be worthwhile to check Pex and Moles, the moles part especially is useful for mocking. – Charles Prakash Dasari Oct 23 '10 at 23:59

4 MSPec with FakeItEasy... making test cases more readable – Robie Jan 15 '12 at 0:29

5 MSpec with NSubstitute and AutoFixture is my choice. – Daniel Hilgarth Mar 19 '12 at 11:27

108

I wouldn't go with MSTest. Although it's probably the most future proof of the frameworks with Microsoft behind it's not the most flexible solution. It won't run stand alone without some hacks. So running it on a build server other than TFS without installing Visual Studio is hard. The visual studio test-runner is actually slower than Testdriven.Net + any of the other frameworks. And because the releases of this framework are tied to releases of Visual Studio there are less updates and if you have to work with an older VS you're tied to an older MSTest.

I don't think it matters a lot which of the other frameworks you use. It's really easy to switch from one to another.

I personally use XUnit.Net or NUnit depending on the preference of my coworkers. NUnit is the most standard. XUnit.Net is the leanest framework.

edited Nov 29 '09 at 1:16

Turnkey
7,995 2 23 35

answered Nov 4 '08 at 9:25

Mendelt
31.9k 6 66 92

36 I've been dragged kicking and screaming to this same conclusion. I really wanted to use MSTest because of it's integration with Visual Studio, but that's also its weakness. I need to run tests on a non-Microsoft build server and there aint no way I am installing Visual Studio on it just to get that. It's a shame that Microsoft produces great tools and then renders them almost unattainable. – Tim Long Mar 10 '10 at 19:48

11 +1 for calling out the awfulness that is MSTest. At the end of the day, it doesn't matter which unit testing framework you use, just as long it as is not MSTest – Mike Mooney Jan 2 '12 at 16:50

21

Consider supplementing, not replacing, MSTest with another testing framework. You can keep Visual Studio MSTest integration while getting the benefits of a more full-featured testing framework.

For example, i use xUnit with MSTest. Add a reference to the xUnit.dll assembly, and just do something like this. Suprisingly, it just works!

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Assert = Xunit.Assert;  // <-- Aliasing the Xunit namespace is key

namespace TestSample
{
    [TestClass]
    public class XunitTestIntegrationSample
    {
        [TestMethod]
        public void TrueTest()
        {
            Assert.True(true);  // <-- this is the Xunit.Assert class
        }

        [TestMethod]
        public void FalseTest()
        {
            Assert.False(true);
        }
    }
}
```

answered Sep 26 '12 at 22:58

Matt Crouch
1,053 1 15 28

This technique may also work for NUnit, MBUnit, or other testing frameworks mentioned in other answers, but i haven't tried them. – Matt Crouch Sep

26 '12 at 23:01

1 do you think I could get parameterized tests to work with MSTest with this approach, Matt? – DevDave Mar 27 '13 at 14:14

@DevDave No. In his example he just used a class from another assembly. If you want parameterized tests you would need another testing framework that is specifically built to extend MSTest. – zoran404 Jun 3 '18 at 17:44

3 `Suprisingly, it just works!` You just called a static function from another assembly. Why are you surprised that it works? Also if you just need assertions that why not use an assembly specifically made for that? – zoran404 Jun 3 '18 at 17:47

9

Nunit doesnt work well with mixed-mode projects in C++ so I had to drop it

answered Jun 5 '09 at 20:27

Eric

14.1k 16 70 127

What did you end up picking? – Daniel Trebbien Jan 1 '12 at 17:51

3 I am not proud of this answer but I dropped unit testing for that project. I resorted to write a lot of validation procedures for runtime detection of errors – Eric Jan 3 '12 at 16:00

2 I'd hoped to use NUnit in mixed-mode but also found it inadequate, in the end I went for googletest which is an excellent C++ unit test framework and v simple to set up. – chillitom May 16 '13 at 13:46

8

It's not a big deal on a small/personal scale, but it can become a bigger deal quickly on a larger scale. My employer is a large Microsoft shop, but won't/can't buy into Team System/TFS for a number of reasons. We currently use Subversion + Orcas + MBUnit + TestDriven.NET and it works well, but getting TD.NET was a huge hassle. The version sensitivity of MBUnit + TestDriven.NET is also a big hassle, and having one additional commercial thing (TD.NET) for legal to review and procurement to handle and manage, isn't trivial. My company, like a lot of companies, are fat and happy with a MSDN Subscription model, and it's just not used to handling one off procurements for hundreds of developers. In other words, the fully integrated MS offer, while definitely not always best-of-bread, is a significant value-add in my opinion.

I think we'll stay with our current step because it works and we've already gotten over the hump organizationally, but I sure do wish MS had a compelling offering in this space so we could consolidate and simplify our dev stack a bit.

answered Dec 29 '08 at 18:46

user8133

206 1 4 5

2 ReSharper does have a compelling offering in this space! – Squirrel Mar 6 '09 at 14:35

7 Your answer is curious, and seems self-contradictory. You say you're a big Microsoft-shop, but won't use TFS (which is the whole point- you won't get the benefit of the vertical integration without it) and use an MSDN subscription model, but are using a non-MS approach. I'm lost, to be honest. Well out-of-date answer, unfortunately. – nicodemus13 May 10 '12 at 15:49

7

It's not a big deal, it's pretty easy to switch between them. MSTest being integrated isn't a big deal either, just grab testdriven.net.

Like the previous person said pick a mocking framework, my favourite at the moment is Moq.

answered Nov 4 '08 at 7:33

Bugg