



Testing Library Approval Request

Date Submitted: 11/14/2021

Team Hobby:

Colin Creasman

Daniel Bribiesca

Team Lead: Jacob Delgado

Long Nguyen

Rifat Hasan

Appendix

| | |
|---|---------------|
| 1 Introduction | 2 |
| 2 Analysis of Available Testing Frameworks | 2 - 4 |
| 2.1 xUnit Pros and Cons | |
| 2.2 MSTest Pros and Cons | |
| 2.3 NUnit Pros and Cons | |
| 3 Discussion | 5 - 18 |
| 3.1 Test Detection Speed | |
| 3.2 Testing Method | |
| 3.3 Speed Test (success cases) | |
| 3.4 Speed Test (fail cases) | |
| 3.5 Speed Test (mixed cases) | |
| 3.6 Speed Test (exception cases) | |
| 3.7 Support Groups | |
| 3.8 Parallel Capability | |
| 3.9 User Experience | |
| 3.10 Testing Framework Testing Analysis | |
| 4 Conclusion | 19 |
| 5 References | 20 |

1 Introduction

“Hobby Project Generator” is a web application that will suggest different projects to the users based on their preferences. To ensure that the website is working properly to its best capabilities and outputting correct results, we would need to test some of the pieces of the website. An automated testing library will be used instead of manual testing to test code frequently, catch any bugs that appear, and overall increase productivity when testing these features. The testing library will need to have parallel capability, data driven methods capability, and have a quick test detection speed. This testing library will not be changing the functionality of the code, but rather be used as a service to make sure the code is running as intended by the client.

2 Analysis of available cloud services:

From our research, the main testing libraries we are looking at are:

- xUnit - Version 2.4.2+
- MSTest (Visual Studio Unit Testing Framework) - Version 2.2.8+
- NUnit - Version 3.13.2+

2.1 xUnit Pros and Cons

xUnit is one of the more recently popular testing libraries, written by the creator of NUnit.

Pro:

- Open-source testing framework that is available on github and is free to use
- Made by creators of NUnit (experienced team)
- Geared towards community-based framework
- One of the largest growing communities
- Can test both client and server side
- Very fast duration speeds

Cons:

- New instance is created for every test:
 - Easily cluttering up for big projects
 - Slower execution of tests
- Traditional attributes found in NUnit were removed
- Non-orthodox naming convention causes experienced coders to learn another naming convention

2.2 MSTest Pros and Cons

Microsoft's testing library that is built into visual studio.

Pro:

- Is embedded in Visual Studio IDE so no further installation is necessary.
- Free and easy setup with minor starting steps
- Fully supported community support
- Open-source and has cross-platform capabilities
- Cross-platform support
- End-to-end development for work management consisting of project planning and version control
- Can test both client and server side

Cons:

- Performance has been reported to be slower than the other testing frameworks
- Third party extensions can overlap with the framework and make it hard for users to use properly (inconvenient interoperability)
- Lacks some features that NUnit and other frameworks have such as ExplicitAttribute

2.3 NUnit Pros and Cons

NUnit is one of the earliest testing libraries and was made by the creator of xUnit.

Pros:

- Allows parallel test execution (4 tests at a time)
- Large community base (24,000 tagged questions on Stackoverflow)
- Most popular framework of the three (126 million downloads from NuGet.org)
- Flexible customization of naming test cases
- Capability of creating custom test attributes through subclassing
- Can test both client and server side

Cons:

- Old software that has been somewhat replaced by xUnit
- Slow duration times for various cases
- Setup attribute is needed to run

Table 1: Analysis between xUnit, MSTest, NUnit

| Metric \ Language (weight) | xUnit | MSTest | NUnit |
|--|--------------|---------------|--------------|
| Speed Test [passed cases] (0.15) | 0.95 | 0.80 | 0.45 |
| Speed Test [failed cases] (0.15) | 0.80 | 0.75 | 0.55 |
| Speed Test [passed & failed cases] (0.20) | 0.95 | 0.85 | 0.80 |
| Speed Test [exception cases] (0.10) | 0.95 | 0.85 | 0.70 |
| Support Groups (0.15) | 0.55 | 0.50 | 0.60 |
| Parallel Capability (0.20) | 0.45 | 0.55 | 0.60 |
| User Experience (0.05) | 0.55 | 0.65 | 0.45 |
| Total (weighted) = 1: | 5.20 | 4.95 | 4.15 |

Legend:

Each metric is assessed on a scale of 0 - 1.

3 Discussion

3.1 Test Detection Speeds

For a testing library to be beneficial to Team Hobby, the speed needs to be faster than doing manual unit tests where test cases would need to be written out before functions in the team's code could be tested.

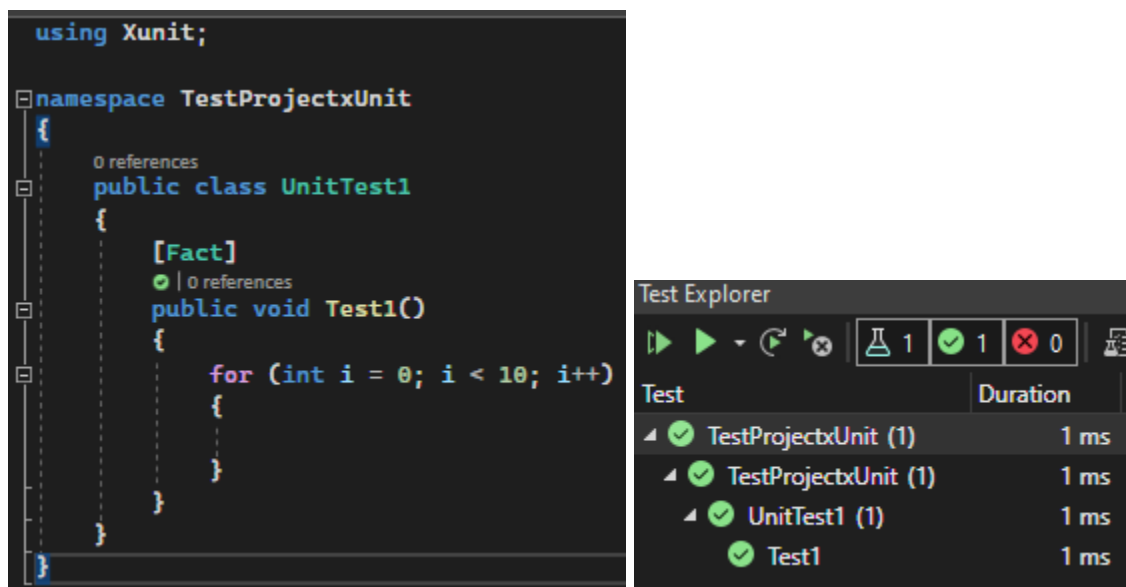
3.2 Testing method

To test each framework within its constraints, similar code was written up for each framework to get the most accurate duration times for each test run. The first picture of each section will include the code used to test the certain metric with its runtime results following after it. Multiple result pictures means that the duration time changed when using various OoM. The range of OoM was 1, 10, 100, and 1000. This sample size was chosen specifically for the size of Team Hobby's project.

3.3 Speed Test (Success cases only)

xUnit:

This framework had the best duration times out of all three, where from 1-1000 iterations, the duration time did not go above 1ms.



The image shows two side-by-side screenshots. The left screenshot is a code editor showing the following C# code:

```
using Xunit;

namespace TestProjectxUnit
{
    0 references
    public class UnitTest1
    {
        [Fact]
        0 references
        public void Test1()
        {
            for (int i = 0; i < 10; i++)
            {
            }
        }
    }
}
```

The right screenshot shows the Test Explorer window with the following table:

| Test | Duration |
|----------------------|----------|
| TestProjectxUnit (1) | 1 ms |
| TestProjectxUnit (1) | 1 ms |
| UnitTest1 (1) | 1 ms |
| Test1 | 1 ms |

MSTest:

For MSTest, regardless of loop size, the duration for completion was very low staying at 2ms until a million iterations were inputted in the for loop. To test the boundaries of this library, we added an additional for loop to have a nested loop with a runtime of $O(n^2)$. Up until 1000 iterations for both loops, the duration stood under 5ms. When another zero was added to both the loops in the nested loop, the duration time jumped to 95ms.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestProject_DAR
{
    [TestClass]
    0 references
    public class UnitTest1
    {
        [TestMethod]
        0 references
        public void SpeedTest1()
        {
            for (int i = 0; i < 1000; i++)
            {
                // ...
            }
        }
    }
}
```

The screenshot shows the Test Explorer window with the following details:

- Toolbar:** Includes icons for running tests (green play button), debugging (green play button with a bug), and test results (flask, green checkmark, red X).
- Test Results Summary:** Shows 1 passed (green checkmark), 1 failed (red X), and 0 errored (red X).
- Test List:**
 - TestProject_DAR (1) - 2 ms
 - TestProject_DAR (1) - 2 ms
 - UnitTest1 (1) - 2 ms
 - SpeedTest1 - 2 ms

50 ms

Nested for loop:

Test Explorer

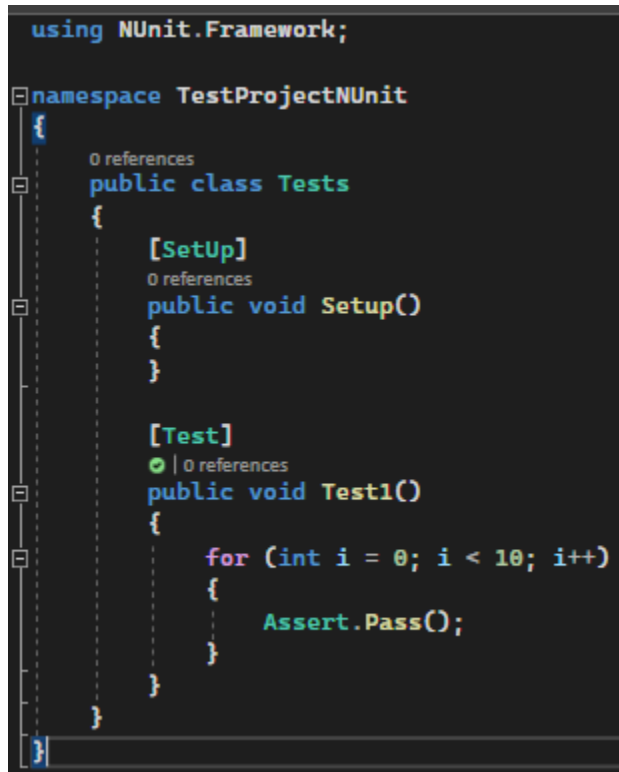
Run (green play), Debug (blue play), Test Results Summary (flask icon with 1 passed, 1 failed, 0 skipped)

| Test | Duration | Traits |
|---------------------|----------|--------|
| TestProject_DAR (1) | 95 ms | |
| TestProject_DAR (1) | 95 ms | |
| UnitTest1 (1) | 95 ms | |
| SpeedTest1 | 95 ms | |

144ms

NUnit:

As for NUnit, the speeds were not as good as MSTests with a base duration of ~20ms for iterations 1-1000. The three test explorer screenshots show the results for 1, 10, and 1000 as 100 and 10 results were the same. This difference of speeds could be because NUnit sits on top of another third party API in order to make it work with Visual Studio where MSTest is directly on VS.

A screenshot of a Visual Studio code editor with a dark theme. The code is written in C# and defines a test class. On the left side of the editor, there is a vertical toolbar with icons for file explorer, search, and other IDE functions. The code is as follows:

```
using NUnit.Framework;

namespace TestProjectNUnit
{
    0 references
    public class Tests
    {
        [SetUp]
        0 references
        public void Setup()
        {
        }

        [Test]
        0 references
        public void Test1()
        {
            for (int i = 0; i < 10; i++)
            {
                Assert.Pass();
            }
        }
    }
}
```


| Test Explorer | | | |
|---|----------|----|--|
| <div> <div> <div></div> <div></div> <div></div> </div> <div> <div>1</div> <div>1</div> <div>0</div> </div> </div> | | | |
| Test | Duration | T. | |
| TestProjectNUnit (1) | 19 ms | | |
| TestProjectNUnit (1) | 19 ms | | |
| Tests (1) | 19 ms | | |
| Test1 | 19 ms | | |

| Test Explorer | | | |
|---|----------|----|--|
| <div> <div> <div></div> <div></div> <div></div> </div> <div> <div>1</div> <div>1</div> <div>0</div> </div> </div> | | | |
| Test | Duration | T. | |
| TestProjectNUnit (1) | 23 ms | | |
| TestProjectNUnit (1) | 23 ms | | |
| Tests (1) | 23 ms | | |
| Test1 | 23 ms | | |

| Test Explorer | | | |
|---|----------|----|---------------|
| <div> <div> <div></div> <div></div> <div></div> </div> <div> <div>1</div> <div>1</div> <div>0</div> </div> </div> | | | |
| Test | Duration | T. | Error Message |
| TestProjectNUnit (1) | 25 ms | | |
| TestProjectNUnit (1) | 25 ms | | |
| Tests (1) | 25 ms | | |
| Test1 | 25 ms | | |

3.4 Speed Test (All failed cases)

xUnit:

Similar test results were given for all failed cases. The runtime slightly increased but this was expected as failed cases take longer than passed cases. A simple false statement was written since xUnit didn't have any "Fail" assertion functions. An iteration of 1000 gave us 9ms showing how fast xUnit's framework runs regardless of a failing scenario.

```

using Xunit;

namespace TestProjectxUnit
{
    0 references
    public class UnitTest1
    {
        [Fact]
        0 references
        public void Test1()
        {
            for (int i = 0; i < 1000; i++)
            {
                Assert.False(true);
            }
        }
    }
}

```

| Test | Duration | T |
|-----------------------|----------|---|
| TestProjectbxUnit (1) | 2 ms | |
| TestProjectbxUnit (1) | 2 ms | |
| UnitTest1 (1) | 2 ms | |
| Test1 | 2 ms | |

| Test | Duration | T |
|-----------------------|----------|---|
| TestProjectbxUnit (1) | 9 ms | |
| TestProjectbxUnit (1) | 9 ms | |
| UnitTest1 (1) | 9 ms | |
| Test1 | 9 ms | |

MSTest:

The fail tests for MSTest were slightly slower than pass cases going up to 19ms for 1000 iterations. Just like the previous tests for MSTest, a nested loop was needed to get a larger duration time.

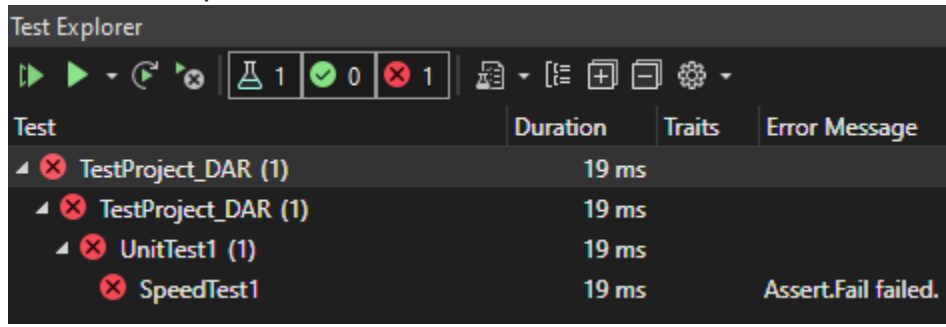
```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestProject_DAR
{
    [TestClass]
    0 references
    public class UnitTest1
    {
        [TestMethod]
        0 references
        public void SpeedTest1()
        {
            for (int i = 0; i < 1000; i++)
            {
                Assert.Fail();
            }
        }
    }
}
```

| Test | Duration | Traits | Error Message |
|---------------------|----------|--------|---------------------|
| TestProject_DAR (1) | 13 ms | | |
| TestProject_DAR (1) | 13 ms | | |
| UnitTest1 (1) | 13 ms | | |
| SpeedTest1 | 13 ms | | Assert.Fail failed. |

61ms

Nested for loop:



Test Explorer

| Test | Duration | Traits | Error Message |
|---------------------|----------|--------|---------------------|
| TestProject_DAR (1) | 19 ms | | |
| TestProject_DAR (1) | 19 ms | | |
| UnitTest1 (1) | 19 ms | | |
| SpeedTest1 | 19 ms | | Assert.Fail failed. |

67ms

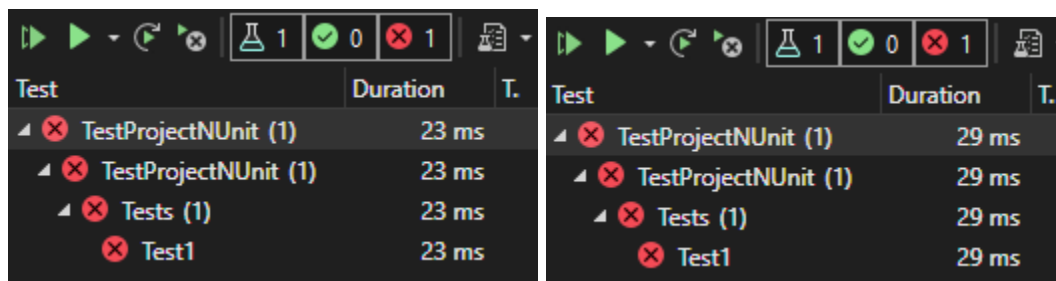
NUnit:

For NUnit the duration time was about 10ms slower than MSTest's time where the 1000 iterations took almost 30ms to complete but MSTest took almost half of that.

```
using NUnit.Framework;

namespace TestProjectNUnit
{
    public class Tests
    {
        [SetUp]
        public void Setup()
        {
        }

        [Test]
        public void Test1()
        {
            for (int i = 0; i < 1; i++)
            {
                Assert.Fail();
            }
        }
    }
}
```



| Test | Duration | T. |
|----------------------|----------|----|
| TestProjectNUnit (1) | 23 ms | |
| TestProjectNUnit (1) | 23 ms | |
| Tests (1) | 23 ms | |
| Test1 | 23 ms | |

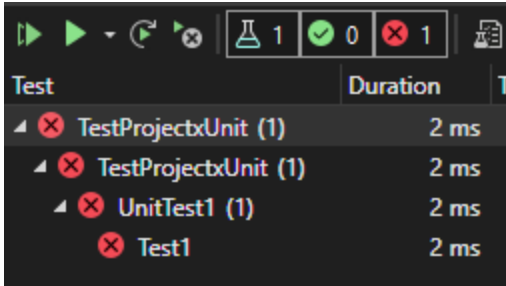
| Test | Duration | T. |
|----------------------|----------|----|
| TestProjectNUnit (1) | 29 ms | |
| TestProjectNUnit (1) | 29 ms | |
| Tests (1) | 29 ms | |
| Test1 | 29 ms | |

3.5 Mixed Speed Test (passed and failed cases)

xUnit:

```
using Xunit;

namespace TestProjectxUnit
{
    0 references
    public class UnitTest1
    {
        [Fact]
        0 references
        public void Test1()
        {
            for (int i = 0; i < 10; i++)
            {
            }
            for (int i = 0; i < 10; i++)
            {
                Assert.False(true);
            }
        }
    }
}
```



| Test | Duration | Status |
|----------------------|----------|--------|
| TestProjectxUnit (1) | 2 ms | Failed |
| TestProjectxUnit (1) | 2 ms | Failed |
| UnitTest1 (1) | 2 ms | Failed |
| Test1 | 2 ms | Failed |

MSTest:

The mixed tests were in between the first two tests. Nested looping was not checked as the duration time was already at a good amount at 1000 iterations. The results came out good, but were still lacking behind xUnit by about a dozen milliseconds.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestProject_DAR
{
    [TestClass]
    0 references
    public class UnitTest1
    {
        [TestMethod]
        0 references
        public void SpeedTest1()
        {
            for (int i = 0; i < 10000; i++)
            {
                Assert.Fail();
            }

            for (int i = 0; i < 10000; i++)
            {
            }
        }
    }
}

```

Test Explorer

1
 0
 1

| Test | Duration | Traits | Error Message |
|-----------------------|----------|--------|---------------------|
| ✖ TestProject_DAR (1) | 13 ms | | |
| ✖ TestProject_DAR (1) | 13 ms | | |
| ✖ UnitTest1 (1) | 13 ms | | |
| ✖ SpeedTest1 | 13 ms | | Assert.Fail failed. |

Test Explorer

1
 0
 1

| Test | Duration | Traits | Error Message |
|-----------------------|----------|--------|---------------------|
| ✖ TestProject_DAR (1) | 18 ms | | |
| ✖ TestProject_DAR (1) | 18 ms | | |
| ✖ UnitTest1 (1) | 18 ms | | |
| ✖ SpeedTest1 | 18 ms | | Assert.Fail failed. |

NUnit:

As usual, the tests for NUnit portrayed slower duration times than the other two frameworks. This time, NUnit was much closer to MSTest while xUnit was much faster than the other two.

```
using NUnit.Framework;

namespace TestProjectNUnit
{
    0 references
    public class Tests
    {
        [SetUp]
        0 references
        public void Setup()
        {
        }
        [Test]
        0 references
        public void Test1()
        {
            for (int i = 0; i < 1; i++)
            {
                Assert.Pass();
            }
            for (int i = 0; i < 1; i++)
            {
                Assert.Fail();
            }
        }
    }
}
```

| Test | Duration | T |
|----------------------|----------|---|
| TestProjectNUnit (1) | 19 ms | |
| TestProjectNUnit (1) | 19 ms | |
| Tests (1) | 19 ms | |
| Test1 | 19 ms | |

| Test | Duration | T |
|----------------------|----------|---|
| TestProjectNUnit (1) | 25 ms | |
| TestProjectNUnit (1) | 25 ms | |
| Tests (1) | 25 ms | |
| Test1 | 25 ms | |

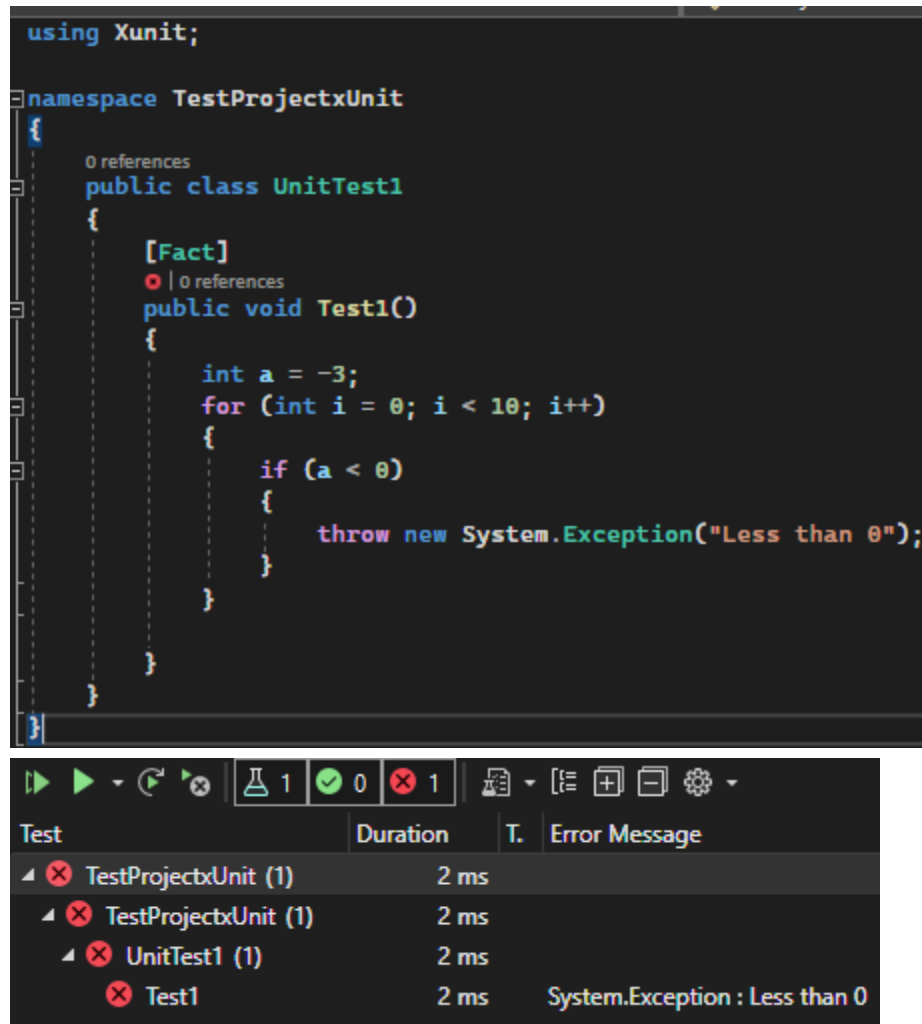
3.6 Speed Test (exception cases)

xUnit:

Exception testing for xUnit was no different than failed cases as it gave the same fast results. These were exceptionally faster than the other two frameworks.

```
using Xunit;

namespace TestProjectxUnit
{
    0 references
    public class UnitTest1
    {
        [Fact]
        0 references
        public void Test1()
        {
            int a = -3;
            for (int i = 0; i < 10; i++)
            {
                if (a < 0)
                {
                    throw new System.Exception("Less than 0");
                }
            }
        }
    }
}
```



| Test | Duration | T. | Error Message |
|----------------------|----------|----|--------------------------------|
| TestProjectxUnit (1) | 2 ms | 1 | |
| TestProjectxUnit (1) | 2 ms | 1 | |
| UnitTest1 (1) | 2 ms | 1 | |
| Test1 | 2 ms | 1 | System.Exception : Less than 0 |

MSTest:

MSTest gave the same duration times as the failed cases.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestProject_DAR
{
    [TestClass]
    0 references
    public class UnitTest1
    {
        [TestMethod]
        0 references
        public void Exception_Test()
        {
            int amount = -3;
            for (int i = 0; i < 10; i++)
            {
                if (amount < 0)
                {
                    throw new System.Exception("Less than 0");
                }
            }
        }
    }
}

```

| Test Explorer | | | |
|-------------------------|----------|----|---|
| | | | |
| Test | Duration | T. | Error Message |
| ⚡ ❌ TestProject_DAR (1) | 15 ms | | |
| ⚡ ❌ TestProject_DAR (1) | 15 ms | | |
| ⚡ ❌ UnitTest1 (1) | 15 ms | | |
| ❌ Exception_Test | 15 ms | | Test method TestProject_DAR.UnitTest1.Exception_Test threw exceptions |

NUnit:

NUnit was also very similar to the failed cases, still coming in last in terms of duration time.


```

namespace TestProjectNUnit
{
    1 reference
    public class Tests
    {
        [SetUp]
        0 references
        public void Setup()
        {
        }

        [Test]
        0 references
        public void Test1()
        {
            var test = new Tests();
            for (int i = 0; i < 1000; i++)
            {
                Assert.Catch<ApplicationException>(() => test.Add(4, 5));
                Assert.Throws<ApplicationException>(() => test.Add(4, 5));
            }
        }

        2 references | 0/1 passing
        private int Add(int v1, int v2)
        {
            return v1 + v2;
        }
    }
}

```

| Test | Duration | T. | Error Message |
|----------------------|----------|----|---|
| TestProjectNUnit (1) | 27 ms | | |
| TestProjectNUnit (1) | 27 ms | | |
| Tests (1) | 27 ms | | |
| Test1 | 27 ms | | Expected: instance of <System.ApplicationException> But was: null |

3.7 Support Groups

In order to work with the testing software efficiently, a large community outreach is helpful when running into popular bugs or issues. “Hobby Project Generator” favors older libraries since more questions and forums have been posted on them throughout the years they have been in service.

This is why although xUnit is community-focused, it is outmatched in terms of the number of tagged questions on stackoverflow and other sites by NUnit and MSTest. MSTest used to be private until recent changes and now the community is second to NUnit.

StackOverflow tagged questions:

- xUnit: 7,500 questions
- NUnit: 24,000 questions
- MSTest: 10,000 questions

Although NUnit has the most tagged questions, this doesn't mean that it has the most active community as it has been around for the longest time. NUnit has the most documentation, while xUnit and MSTest are gaining more popularity as time goes by.

3.8 Parallel Capability

Parallel capability is significant as productivity can increase when multiple tests are being done simultaneously rather than having to run each test one after another. Since there will be many unit tests to test each functional function on the automated testing library, parallelism gives an edge on how fast multiple tests can be completed without having to wait and decrease productivity.

| | xUnit.net | MSTest | NUnit |
|----------------------------------|-----------|--------|-------|
| No parallelization | ✓ | ✓ | ✓ |
| Class scoped parallelization | ✓ | ✓ | ✓ |
| Method scoped parallelization | | ✓ | ✓ |
| Parameter scoped parallelization | | | ✓ |

Figure 1: Parallelization scopes of xUnit, MSTest, and NUnit (jsakamoto, 2021)

All three frameworks have both non-parallelization and parallelization capabilities. By default, xUnit runs all instances in parallel to each other because every method is a different instance. To combine two methods, the `Collections` attribute is used to run the tests without parallelism. However, xUnit is constrained to class scoped parallelization. On the other hand, NUnit does have the capability of parallelism at assembly, class, and method level if it is enabled in the settings. Lastly, MSTest received the lowest score as although it does support parallelism, it is only at the method and class level, giving the other two frameworks an edge when it comes to parallel testing.

Given the team's hardware capabilities, xUnit would not compensate for the team's goals in parallelization since there are CPU cores that are not being used to its full

capacity. Furthermore, in some cases running in parallelization through the parameter scope rather than the class scope results in much faster tests. Although this is not always the case, “Hobby Project Generator” will need a testing library that has method parallelization capabilities, while parameter scoped parallelization will most likely not be needed due to the size of this project.

3.9 User experience

User experience is significant as it would help maneuver around the testing library and make it easier to use. The UX of the testing framework has very little weight on the overall score of the libraries because the advantages are miniscule and subjective. Any of the three frameworks could be aesthetically pleasing to different users, so simplicity and clarity was sought when grading each framework.

xUnit provides a very innovative way of displaying and typing test attributes. Unlike the orthodox method of naming attributes with brackets, xUnit utilizes both the old attribute naming style and takes on a new style with no brackets. Each set of frameworks have their own utility belt for various attributes. For example, xUnit can be said that the new outlook of how their testing library is formatted gives it a cleaner look, while others believe that the new changes make it difficult for experienced users to learn another syntax. This subjectivity is the reason why this category is very low in the grading scale above.

For a program to be data driven, the input data needs to be manipulable to where the program is able to run the same way with different inputs. Typically, the program logic is what controls the program, but data driven programming makes it so that the data itself is in control. All frameworks had some sort of data driven programming implementations embedded into the libraries.

3.10 Testing Framework Testing Analysis

Initially, we suspected that MSTest would secure the number one spot for this section as it is a native testing library that runs straight on top of Visual Studio. NUnit and xUnit on the other hand, were thought of being in a close second with each other as a third party application would need to be implemented to run on top of the APIs that allow the testing libraries to run on Visual Studio. However, this lack of direct integration and need for extra plugins was wrong as xUnit had much closer duration times than NUnit, being first in all the speed tests conducted. This could be attributed to xUnit being the newest released framework from creators that have experience and have been through the trials and errors of creating NUnit.

4 Conclusion

The growth of features from Microsoft on MSTest has set it as a good pick for “Hobby Project Generator” as it provides high test speeds, direct integration with Visual Studio, and other benefits. However, the leading pick for Team Hobby is xUnit as it provides the functions needed to test the team’s code and is the most updated framework of the three. Although the differences are very subtle, the key difference that sets xUnit apart from its counterparts is that it is more widely being an industry favorite and provides very fast duration speeds for all benchmark tests given. The attributes xUnit uses is a little different than MSTest and NUnit but all three virtually have the same methods and functions. The only feature that might hurt productivity from this framework is the non-orthodox naming convention, but this can be picked up quite easily. Apart from this, xUnit runs like any other testing library with minor distinctions. Both NUnit and MSTest are good testing libraries that others may enjoy better, but with MSTest’s long duration speed for failed cases, and NUnits needing to be downloaded as a third party software, makes xUnit the best choice with MSTest coming at a close second.

5 References:

1. Hamid Mosalla. "Xunit – Part 4: Parallelism and Custom Test Collections." *Hamid Mosalla*, 26 Jan. 2020, <https://hamidmosalla.com/2020/01/26/xunit-part-4-parallelism-and-custom-test-collections/>.
2. Mathurin, Chris. "NET Core 2: Why Xunit and Not Nunit or MSTest." *DEV Community*, DEV Community, 30 Mar. 2019, <https://dev.to/hatsrumandcode/net-core-2-why-xunit-and-not-nunit-or-mstest--aei>.
3. "Nunit vs. Xunit vs. MSTest: Comparing Unit Testing Frameworks in C#." *LambdaTest*, 17 Nov. 2021, <https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/>.
4. "Nunit vs. Xunit vs. MSTest: Comparing Unit Testing Frameworks in C#." *LambdaTest*, 17 Nov. 2021, <https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/>.
5. Vinugayathri. "Why Should You Use Xunit? A Unit Testing Framework for .NET." *Clarion Tech*, <https://www.clariontech.com/blog/why-should-you-use-xunit-a-unit-testing-framework-for-.net>.
6. "Test Automation Frameworks." *Smartbear.com*, <https://smartbear.com/learn/automated-testing/test-automation-frameworks/>.
7. Advolodkin, Nikolay, and Nikolay Advolodkin on February 17. "MsTest vs Nunit: Which Should You Use and Why?" *Ultimate QA*, 20 Nov. 2020, <https://ultimateqa.com/mstest-vs-nunit/>.
8. "Automated Unit Testing: MSTest vs Xunit vs Nunit." *AnAr Solutions Pvt. Ltd.*, 27 Nov. 2020, <https://anarsolutions.com/automated-unit-testing-tools-comparison/>.
9. "Getting Started: .NET Framework with Visual Studio." *XUnit.net*, <https://xunit.net/docs/getting-started/netfx/visual-studio>.
10. "Most Complete MSTEST Framework Tutorial Using .NET Core." *LambdaTest*, 19 Nov. 2021, <https://www.lambdatest.com/blog/most-complete-mstest-framework-tutorial-using-net-core-2/>.