DevOps Testing

Lab Tasks

Task 1: Prepare the Walk through 2
Task 2: Create a Unit Test project 3
Task 3: Create a Test class 4
Task 4: Create a Test method 4
Task 5: Build and run the test 4
Task 6: Add Testing to the Build 5
Task 7: Commit changes 6
Task 8: Fix the code and rerun the test 6
Task 9: Commit changes 7
References: 7

Task 1: Prepare the Walk through

1. Open Visual Studio
2. Open the project created in the CI lab.
3. In the **Solution Explorer**, right click on the Solution.
4. Select **Add -> New Project…**
5. Under **Installed**, expand **Visual C#**.
6. In the subsequent list of application types, click on **Class Library (.NET Framework).**
7. In the **Name** box, type **Calc.**
8. Click on **OK**.
9. The new Calc project will be created and displayed in the **Solution Explorer** with the **Class1.cs** file opened in the Code Editor.
10. Replace the original contents of **Class1.cs** with the next code.

```
using System;
namespace CalcNs
{
    /// <summary>
    /// Calc demo class.
    /// </summary>
    public class Calc
    {
        private double _val1;
        private double _val2;
        public Calc()
        {
        }
        public Calc(char operation, double val1, double val2)
        {
            _val1 = val1;
            _val2 = val2;
        }
        public double Val1
        {
            get { return _val1; }
        }
        public double Val2
        {
            get { return _val2; }
        }
        public double Div(double val1, double val2)
        {
            if (val2 == 0)
            {
                throw new DivideByZeroException("0 value on division");
            }
            return val1 / 0; // intentionally incorrect code
        }
        public double Mult(double val1, double val2)
        {
            if (val1 < 0)
            {
                throw new Exception("value 1 less than 0");
            }
            if (val2 < 0)
            {
```

```
                throw new Exception("value 2 less than 0");
            }
            return val1 * val2;
        }
        public double Subs(double val1, double val2)
        {
            if (val1 < 0)
            {
                throw new Exception("value 1 less than 0");
            }
            if (val2 < 0)
            {
                throw new Exception("value 2 less than 0");
            }
            return val1 - val2;
        }
        public double Sum(double val1, double val2)
        {
            if (val1 < 0)
            {
                throw new Exception("value 1 less than 0");
            }
            if (val2 < 0)
            {
                throw new Exception("value 2 less than 0");
            }
            return val1 + val2;
        }
    }
}
```

11. Go to **File** > **Save Class1.cs As…**
12. Change the name to **Calc**.
13. Click on **Save**.
14. On the **Build** menu, click **Build Solution**.
15. You now have a project that contains source code to test and tools to test it with. The namespace for Calc, is **Calc** and contains the public class **Calc** whose methods you will test in the following procedures.

Task 2: Create a Unit Test project

1. In the **Solution Explorer**, right click on the Solution.
2. Select **Add ->New Project…**
3. In the new window, select **Installed -> Visual C# -> Test** from the left list.
4. Select **Unit Test Project (.NET Framework)**.
5. In the **Name** box, enter **CalcTest**.
6. Click on **OK**.
7. The **CalcTest** project is added to the solution.
8. In the **CalcTest** project, add a reference to the Class solution.
    a. In **Solution Explorer**, right click to **CalcTest** > **References**.
    b. Click on **Add Reference...**
    c. In the **Reference Manager - CalcTest** dialog box, expand **Projects > Solution**.
    d. Check the **Calc** item.
    e. Click on **OK**.

Task 3: Create a Test class

1. In Solution Explorer, go to **CalcTest**.
2. Right click on the **UnitTest1.cs** file.
3. From the context menu, choose **Rename**.
4. Rename the file to **CalcTests.cs**.
5. If prompted choose **Yes** on the dialog that asks if you want to rename all references in the project to the code element 'UnitTest1'.
6. Add a using statement to the class to call the project under test without using fully qualified names. At the top of the class file, add:

```
using CalcNs;
```

Task 4: Create a Test method

In this procedure, we will write unit test methods to verify the behaviour of the **Div** method of the **Calc** class. The method is earlier listed above in this document.

By analysing the method under test, we determine that there are at least three behaviours that need to be checked:

1. The method throws an **DivideByZeroException** if the val2 is equals to 0.
2. It also throws **Exception** if the val2 is less than 0.
3. If the checks in **1.** and **2.** above are satisfied, the method divides the two values and returns the result.

In our first test, we verify that we have a valid val2.

1. Add the below Method to the **CalcTests** class.
2. Add the next method.

```
[TestMethod]
public void DivCorrect()
{
       double val1 = 11.99;
       double val2 = 4.55;
       double expected = 2.63;
       Calc calc = new Calc();
       double result = calc.Div(val1, val2);
       Assert.AreEqual(expected, result, 0.01, "Division correct");
}
```

The method is rather simple. We set up a new **Calc** object and then divide the 2 values. We use the Microsoft unit test framework for managed code **AreEqual** method to verify that the result of the method is what we expect.

Test method requirements

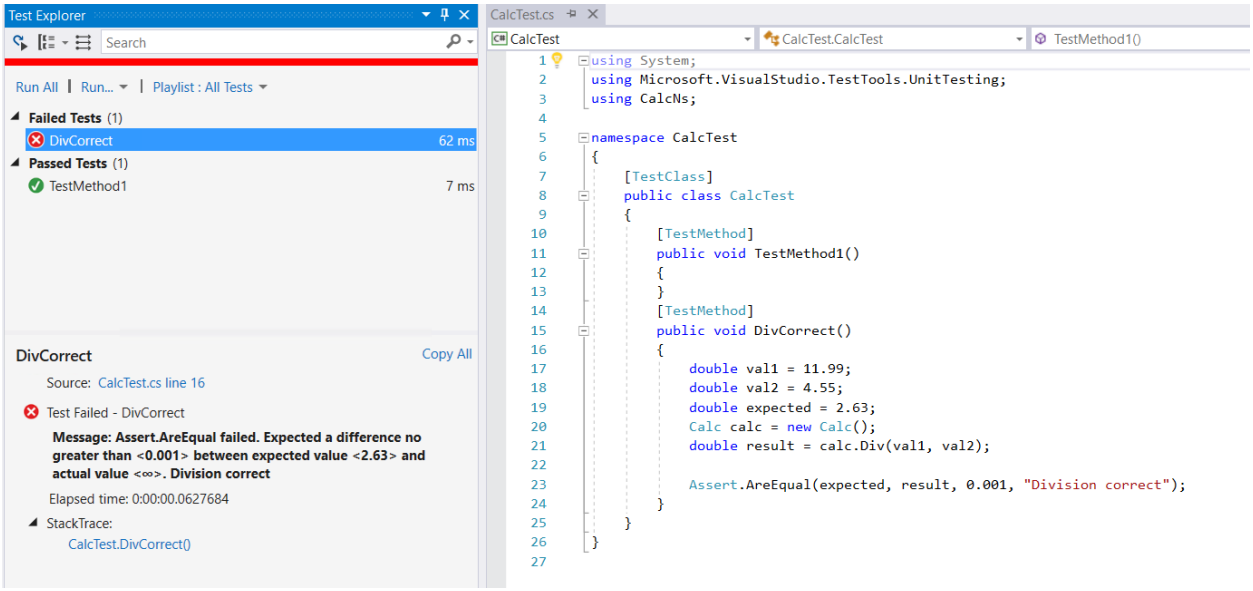
A test method must meet the following requirements:

- The method must be decorated with the **[TestMethod]** attribute.
- The method must return void.
- The method cannot have parameters.

Task 5: Build and run the test

1. Go to **Build** > **Build Solution** from the top methodto build the solution.
2. Open the **Test Explorer**.
    - Go to **Test -> Windows -> Test Explorer**.

3. You will see the **DivCorrect** method listed in the **Not Run Tests** section from the **Test Explorer**.
4. In **Test Explorer**, click **Run All** to run the test. As the test is running the status bar at the top of the window is animated. At the end of the test run, the bar turns green if all the test methods pass, or red if any of the tests fail.
5. In this case, the test does fail.
    - Select the method in **Test Explorer** to view the details at the bottom of the window, as per the screenshot below. Also note the presence of **Red Circles with X's** in them in the **Class.cs** class file to indicate a fail.



Task 6: Add Testing to the Build

1. Go to your VSTS portal.
2. Click **Pipelines -> Builds**.
3. Select the build created in the CI lab.
4. Click on **Edit** from the top Builds menu.
5. Add a task.
    - Click on <+> beside the Agent job 1.
    - Select the **Test** tab.
    - Select **Visual Studio Test**.
    - Click on **Add**.
6. Modify the new task.
    - Select VsTest – testAssemblies.
    - Complete:
        - Select tests using = **Test assemblies**.
        - Test files = ****\$(BuildConfiguration)*test*.dll !**\obj****
        - Search folder = **$(System.DefaultWorkingDirectory)**.
    - In the execution options section
        - Code coverage enabled = **Checked**.
7. Save the build.

- Click on **Save & Queue** from the top menu.
- Click on **Save** from the list
- Add a Comment
- Click on **Save**.

Task 7: Commit changes

1. Back to the solution in Visual Studio.
2. In the **Solution Explorer**, right click on the solution.
3. Click on **Commit…**
4. Add a message.
5. Click on **Commit All**.
6. Click on **Sync** from the new yellow message or from the Home page of the Team Explorer.
7. Click on **Push**.
8. Back to the VSTS.
9. Go to **Pipelines -> Builds**.
10. Click on the Build created.
11. See there is a new build in process.
    - Validate the build fails.

Task 8: Fix the code and rerun the test

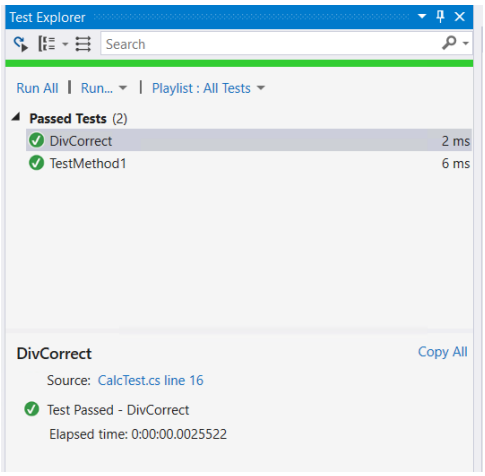
Analyse the test results

The test result output in Test Explorer contains a message that describes the failure.
For the **AreEquals** method, message displays you what was expected (**the Expected <XXX>parameter**) and what was actually received (**the Actual<YYY> parameter**).
A re-examination of the **Div** code shows that the unit test has succeeded in finding a bug. The Div method was trying to divide by 0.

1. Correct the bug.
    - Go to **Calc** class in the **Calc** project.
    - Go to the **div** method.
    - Update the return line with the next code.

```
return val1 / val2;
```

2. Rerun the test
    - In **Test Explorer**, click on **Run All** to rerun the test. The test is passed and the red turns green.



Task 9: Commit changes

1. In the **Solution Explorer**, right click on the solution.
2. Click on **Commit…**
3. Add a message.
4. Click on **Commit All**.
5. Click on **Sync** from the new yellow message or from the Home page of the Team.
6. Click **Push**.
7. See there is a new build in process.
    - Validate the build fails.

References:

https://msdn.microsoft.com/en-us/library/ms243176.aspx
https://microsoft.github.io/PartsUnlimited/testing/200.5x-Testing-UnitTestforMngdCode.html