Unit Testing

Summer of Tech

Pre-requisites

Should be installed and ready to go before the workshop

Hardware

1. Access to a computer with internet access (Wifi access will be provided)
2. At least 5GB free disk space
3. Bring your power chord or make sure your battery last for at least an hour
4. Bring a mouse

Software

1. Github account
2. Git SCM installed and available via command line
3. .Net Core SDK (v2.1.3)
4. Visual Studio Code or Visual Studio Community Edition. If you install Visual Studio code, you will the following plugins
    1. C# plugin
    2. .NET Core Test Explorer plugin
    3. Rust Test Lens

Resources

1. Soft copy of the presentation slides: https://s3-ap-southeast-2.amazonaws.com/sot2018-collateral/SoT+2018+-+Unit+testing+the+first+step+to+continuous+delivery.pdf
2. Soft copy of this document: https://s3-ap-southeast-2.amazonaws.com/sot2018-collateral/Workshop_v1.pdf
3. XUnit documentation: https://xunit.github.io/docs/getting-started-dotnet-core#write-first-tests
4. Moq documentation: https://documentation.help/Moq/
5. Fixture documentation: https://github.com/AutoFixture/AutoFixture/wiki/Cheat-Sheet
6. Quick commands for VSCode: *Ctrl+Shift+P*
7. Common git commands
    1. List all local branches
       ```
       git branch
       ```
    2. List all remote branches
       ```
       git branch -r
       ```
    3. Reset current branch (lose all work)
       ```
       git reset —-hard
       ```
    4. Reset current branch (unstage work but leave it intact)
       ```
       git reset
       ```
    5. Commit all your work
       ```
       git add .
       git commit -m “Some commit message”
       ```
    6. Git status
       ```
       git status
       ```
    7. Delete branch
       ```
       git branch -d name_of_branch
       ```
8. Dotnet commands
    1. Run unit tests
       ```
       dotnet test
       ```
    2. Build code
       ```
       dotnet build
       ```
    3. Add Nuget package
       ```
       dotnet add package name_of_package
       ```

Exercises

For all exercises we'll be using a GitHub repository, lets clone it using the following command:

```
git clone https://github.com/ruskindantra/SoT_2018.git
```

Exercise 1 (15 mins)

Synopsis

We will be writing some unit tests for a package available off Nuget. This package has about 3 intentional errors, lets try finding them.

General rules

1. It's about a car
2. The car can be "Started" and "Stopped"
    1. The car cannot be started/stopped twice
3. The car can "MoveForward" and "MoveBackward"
    1. Every time it moves forward
        1. It increases the "MetresTraveled" by 0.1
        2. It consumes one litre of petrol
    2. Every time it moves backwards
        1. It decreases the "MetresTraveled" by 0.1
        2. It consumes one litre of petrol
4. The car be "Service"d

Steps

1. Via command line navigate to the directory into which you cloned the repository
2. Switch to the *exercise_1* branch using the following command:

```
git checkout -b exercise_1 origin/exercise_1
```

3. Open your IDE of choice load the solution/folder
4. There are 3 stubbed test methods, populate these with some code
5. ***Advanced:*** Add some more unit tests to expose the other issues

Exercise 2 (15 minutes)

Synopsis

In this exercise we'll be improving our code base to put into practice the S and O principles of SOLID.

Steps

1. Switch to the *exercise_2* branch using the following command:

```
git checkout -b exercise_2 origin/exercise_2
```

2. There are some items marked as *TODO* in the *Car.cs* file under *Vehicle* namespace, lets give those a try
3. Note that there are a few different options on how we can go about doing this
4. Add some unit tests to exercise your changes
5. ***Advanced:*** Do you see any other opportunities where we can implement or improve on the code with respect to Single Responsibility and Open Close principles?

Exercise 3 (15 minutes)

Synopsis

In this exercise we'll be improving our code base to put into practice the L and I principles of SOLID.

1. Switch to the *exercise_3* branch using the following command:

```
git checkout -b exercise_3 origin/exercise_3
```

2. There are some items marked as *TODO* in the *Car.cs* file under *Vehicle* namespace, lets give those a try
3. ***Advanced:*** Consider the class *Car*, is that the only vehicle you can have? How can you apply the L and I principles to make that flexible?

Exercise 4 (35 minutes)

Synopsis

In this exercise we'll be moving our code towards a more robust style of programming using the Dependency inversion principle, this will be done in 4 stages.

1. Switch to the *exercise_4_s1* branch using the following command:

```
git checkout -b exercise_4_s1 origin/exercise_4_s1
```

2. There are some items marked as *TODO* in the *Car.cs* file under *Vehicle* namespace, lets give those a try
3. With the above change, you will also need to update your unit tests, there is a *TODO* there too
   1. While updating your unit tests, what have you noticed about the unit(s) you were testing?
   2. Were we actually writing unit tests?
4. Switch to the *exercise_4_s2* branch using the following command:

```
git checkout -b exercise_4_s2 origin/exercise_4_s2
```

5. In this stage we'll convert our unit tests (which were actually integration test) to proper unit test
6. Continue working in the *car* unit test class, complete the *TODOs*
   1. Run your unit tests, what have you noticed?
   2. Now imagine you have 10s or 100s of interfaces, will you be writing a dummy implementation for all?
7. Switch to the *exercise_4_s3* branch using the following command:

```
git checkout -b exercise_4_s3 origin/exercise_4_s3
```

8. Moq to the rescue, continue working in the unit test and complete the *TODOs*
9. ***Advanced:*** If time allows, introduction to Fixture
10. Switch to the *exercise_4_s4* branch using the following command:

```
git checkout -b exercise_4_s4 origin/exercise_4_s4
```

11. Use fixture in your unit tests, have a play with the rest of the code and the libraries

Conclusions

1. While coding, remember the SOLID principles
2. Unit testing good code is fun, if you are not having fun, refactor your code
3. There are several free libraries that can be leveraged if your code follows the SOLID principles, use them to your advantage