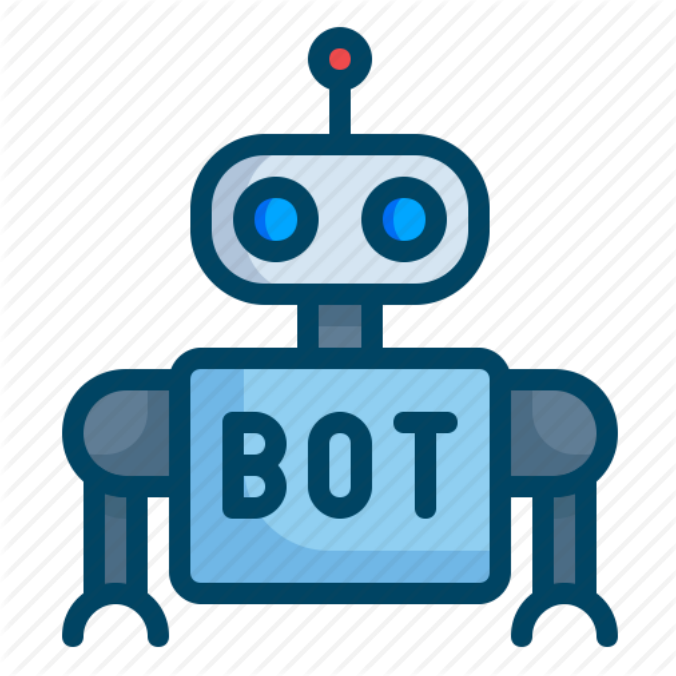
Unit what!?

Unit testing with C#

Armando Cifuentes González

Software Engineer in Test
iTexico

LinkedIn: *https://www.linkedin.com/in/arcigo/*
Github: *https://github.com/ArCiGo*
Twitter: *@_ArCiGo*
armandocifuentes_2@yahoo.com.mx



What is unit testing?

A unit test (component testing, according to the **ISTQB**) is a piece of a code (usually a method) that invokes another piece of code and checks the correctness of some assumptions afterward. If the assumptions turn out to be wrong, the unit test has failed. A unit is a method or function.

Component testing is often done in isolation from the rest of the system, depending on the software development lifecycle model and the system, which may require mock objects, service virtualization, harnesses, stubs, and drivers. Component testing may cover functionality (e.g., correctness of calculations), non-functional characteristics (e.g., searching for memory leaks), and structural properties (e.g., decision testing)

Objectives of unit testing

- Reducing risk
- Verifying whether the functional and non-functional behaviors of the component are as designed and specified
- Building confidence in the component's quality
- Finding defects in the component
- Preventing defects from escaping to higher test levels

Test basis

- Detailed design
- Code
- Data model
- Component specifications

Test objects

- Components, units or modules
- Code and data structures
- Classes
- Database modules

Typical defects and failures

- Incorrect functionality (e.g., not as described in design specifications)
- Data flow problems
- Incorrect code and logic

Benefits of unit testing

- Makes the Process Agile
- Quality of Code
- Finds Software Bugs Early
- Facilitates Changes and Simplifies Integration
- Provides Documentation
- Debugging Process
- Design
- Reduce Costs

How to start writing unit tests?

Follow the next steps.-

- Create an abstract class for the class to be tested (i. e. **YourAwesomeClass**). The name of the abstract class should have the suffix **Test**. This class will only have shared members and **SetUp** and **TearDown** methods. This last step is optional if your target class doesn't require any initialization or shared members.
- Create the classes with the name of the methods to be tested (**Methods**). These classes will inherit from the base class, so you don't need to reinvent the wheel initializing variables, constants, etcetera.
- The name of the class must be descriptive and clear to understand.

- Inside of the class, create the methods with the name of the possible scenarios that will occur. What are your **InitialConditions** and what will be your **ExpectedResults**?
- *Optional 1:* Use mocks for your tests.
- *Optional 2:* Change the name of the namespace (it could be, for example, namespace **YourWorkspace.Test.ClassTests**, in plural)
- *Note 1:* If there isn't anything to set up, omit the abstract class.

Pattern

```
using NUnit.Framework;

namespace YourWorkspace.Test.YourAwesomeClassTest
{
    public abstract class YourAwesomeClassTest
    {
        // Your setup
    }

    [TestFixture]
    public class SomeMethodToTest : YourAwesomeClassTest
    {
        [Test]
        public void OnePossibleScenarioOfTheMethod_ExpectedResult()
        {
            // Your code
        }

        // Other methods
    }

    // Other classes
}
```

Example

```
using NUnit.Framework;

namespace YourWorkspace.Test.CalculatorTests
{
    public abstract class CalculatorTest
    {
        protected double x = 234.12, y = 134;
        protected string xString = "Hi", yString = "Something";
    }

    [TestFixture]
    public class Addition : CalculatorTest
    {
        [Test]
        public void SumTwoNumbers_ReturnsTrue()
        {
            var calc = new Calculator();

            Assert.AreEqual(calc.Addition(x, y), 368.12);
        }

        [Test]
        public void SumTwoNumbers_ReturnsFalse()
        {
            var calc = new Calculator();

            Assert.AreEqual(calc.Addition(x, y), 5);
        }

…
```

Example

```
...

        [Test]
        public void SumTwoRandomVariables_ThrowsException()
        {
            var calc = new Calculator();

            Assert.Throws<Exception>(() => {
                calc.Addition(xString, y);
            });
        }

        //Other possible scenarios
    }

    //Your other classes to test
}
```

Advantages of this convention:

- Code tests well organized
- Method names more descriptive
- Easy to maintain

Disadvantages of this convention:

- Nothing

Q&A

Bibliography

- **Defining unit testing, step by step**. The Art of Unit Testing (with examples in C#). *Roy Osherove*. Manning Publications Co. 2014
- **Component Testing**. Certified Tester. Foundation Level Syllabus. *International Software Testing Qualifications Board*. 2018.
- **8 benefits of Unit Testing**. *DZone*. 2019: https://dzone.com/articles/top-8-benefits-of-unit-testing
- **Unit Testing**. *The C# Unosquare Labs Best Practices*. 2017: https://github.com/unosquare/best-practices/tree/master/C%23#unit-testing


reminding you to drink water
@drinkwaterho
DONT
FORGET
TO
DRINK
WATER
YOU
STUPID
BITCH
6:44 PM · Nov 29, 2019 · Twitter for iPhone

Thanks… :D