The Art of Unit Testing- Robert Osherove

# What is a Unit Test?

A **unit test** is an automated piece of code that invokes the **unit of work** being tested, and then checks some assumptions about a single end result of that unit.

* A unit test is almost always written using a unit testing framework.
* It can be written easily and runs quickly.
* It’s trustworthy, **readable**, and maintainable.
* It’s consistent in its results, as long as production code hasn’t changed.

# Other Terminology

A **unit of work** is the sum of actions that take place between the invocation of a public method in the system and a single noticeable end result by a test of that system.

The class or code that you’ll write tests for is called the **System Under Test** (SUT). In other resources, this is sometimes referred to as the Code Under Test or Class Under Test (CUT).

# Characteristics of a Unit Test

It should be automated and repeatable.

* It does not involve manual steps or interaction.

It should run quickly.

* Meaning, less than 100 MS
* Slow tests make it much more likely that tests won’t be run.
* Slow tests are likely using external dependencies and aren’t truly unit tests.

It should be easy to implement.

* If a test is hard to write, the underlying code is likely the problem.
* Look for opportunities to introduce interfaces and seams, and to remove unneeded dependencies, and move functionality into smaller units.

It should be relevant tomorrow.

* Date dependencies should be tested, but in an automated way.

It should be consistent in its results .

* It always returns the same result if you don’t change anything between runs.

It should have full control of the unit under test.

* This is achieved by isolating the dependencies and external references, and thorough review of the code’s functionality.

It should be fully isolated (runs independently of other tests).

* There should be no dependency on other tests.
* Should maintain its own setup and tear down.

When it fails, it should be easy to detect what was expected and determine how to pinpoint the problem.

* The test’s name, and the output returned from the assert statements, are key here.

Anyone should be able to run it at the push of a button.

* Test runs are available without opening the solution, via the use of unit testing frameworks (depending on your testing framework).
* MS Test makes this process much harder (lacks a GUI tool, but there is MSTEST.exe)

# What a Unit Test Is Not

It is not an integration test!

**Integration testing**is testing a unit of work without having full control over all of it and using one or more of its ***real dependencies***, such as time, network, database, threads, random number generators, and so on.

Integration tests are very important, but they fulfill a different role than a unit test.

* These are typically intended to test the full functionality of an application.
* Unit tests focus on an individual piece of interaction or functionality.

If your test uses a database or an external service, it’s **not going to be fast and repeatable**.

# Naming Standards

Taken from the CXM Wiki’s Testing- Conventions & Structure Document

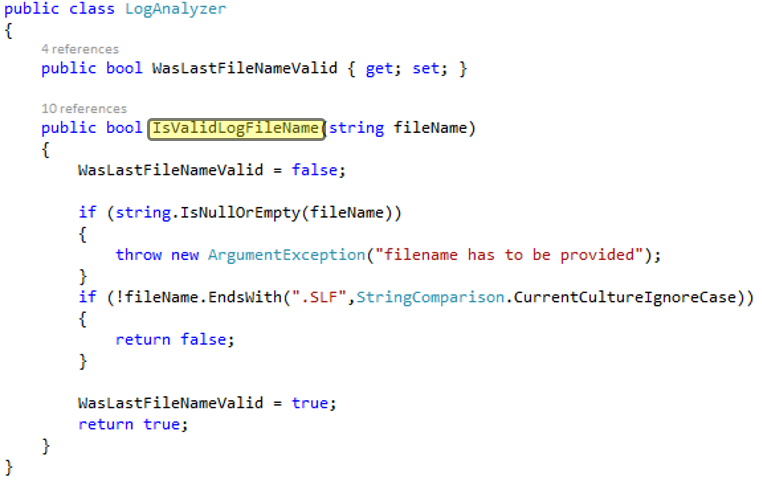
The name of the unit test method should follow the following pattern:

***Method\_StateUnderTest\_ExpectedBehavior***

This means:

* The name of the method.
* Followed by the condition being tested.
* Followed by the expected outcome.
* With each section being separated by an underscore (\_).

# Code Examples: System Under Test



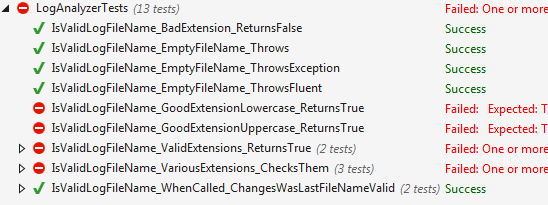
## Tests

Machine generated alternative text: [Test)
public void IsValidLogFileName_GoodExtensionLowercase_ReturnsTrue()
{I
LogAnalyzer analyzer = new LogAnalyzerQ;
bool result = analyzer. IsValidLogFileName(”filewithgoodextension. slf”);
Assert .True(result);

Machine generated alternative text: [rest]
ref ere rces
public void IsValidLogFileName_GoodExtensionUppercase_ReturnsTrue()
{
LogAnalyzer analyzer = new LogAnalyzerQ;
bool result = analyzer.IsValidLogFileNante(”filewithgoodextension.SLF”);
Assert.True(result);

## Test Output

Using these naming methods, we can easily determine expected results, why tests are failing, and how to fix them



# Assemblies and Test Names

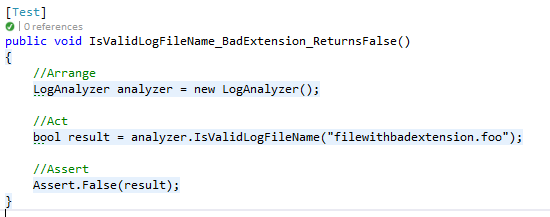
* Test assemblies should have the same namespace as their code counterparts, with “Tests” or “UnitTests” added
* The test file’s name should correspond with the name of the file that contains the class you are testing.

Add properly named tests, decorated with the [TestMethod] attribute.

* Remember: **MethodName\_StateUnderTest\_ExpectedBehavior.**

# Methodolgy: Arrange, Act, Assert

1. *Arrange* objects, creating and setting them up as necessary.
2. *Act* on the object, which will be the System Under Test
3. *Assert* that something is as expected.



# Isolation Frameworks

An *isolation framework* is a set of programmable APIs that makes creating fake objects much simpler, faster, and shorter than hand-coding them.

But what is a fake object?

Fakes are used to create a controllable substitute for a dependency in your program

* Anything required to be passed in or accessed by your System Under Test.
* Services, Database calls, other classes, etc.
* Can be called Stubs, Fakes, or Mocks.
* Usage of these terms varies and often leads to debate or flame wars.

# Negative Testing

* Negative testing is a method to ensure that we can gracefully handle an abnormal situation.
* We write code that handles the expected “happy path” of user and interaction.
* ….What happens when things go wrong?
* Add tests that have unexpected input, which simulate services or data being unavailable or null, or have actively malicious input. Or just stupid users.
* …You may find your code becomes more secure and stable as a result.

# Final Thoughts from Harlan Sanders (no really)

The easy way is efficacious and speedy, the hard way arduous and long. But, as the clock ticks, the easy way becomes harder and the hard way becomes easier. And as the calendar records the years, it becomes increasingly evident that the easy way rests hazardously upon shifting sands, whereas the hard way builds solidly a foundation of confidence that cannot be swept away.