FRONTRANGESYSTEMS
custom software solutions
frontrangesystemscom

Unit Test Guidelines

A guide for awesome unit tests.

These guidelines are by no means a complete list; neither are they intended to be a rigid set of unbreakable laws. They're guidelines and this is a starting point.

Idempotent

All tests shall clean up after themselves. There should be no difference if a test has been run once or a thousand times.

Execution order is irrelevant

Unit tests should not rely on a previous unit test to run successfully. Any data setup or preconditions should be handled either in the `ClassInitialize` or the `TestInitialize` methods.

Code coverage

As a general rule, code coverage is expected to be a minimum average of 80%. Code coverage for some classes will be higher, and some will be lower, but an average of 80% is the minimum per assembly. An extremely low coverage percentage is typically a symptom of code smell; e.g. the class is doing too much, is too tightly coupled with other classes, or needs some refactoring in order to become testable.

Name test after expected result

Method names in unit tests help developers who are either outside the team, or unfamiliar with the particular piece of code familiarize themselves quickly with the tests.

Tests should only be testing one thing in one class

In general, with all code, methods should be short in order to reduce cyclomatic complexity and increase maintainability. The same is true for unit tests. Small test methods ensure that when a test fails, only one thing made it fail. Reduce or eliminate dependencies by using mock objects, and keep things simple. The best solutions to complex problems are simple, not complex.

Tests should be small and fast

The system will eventually have thousands of unit tests that will run on every check in. Long running unit tests equals more broken builds when a developer checks in some breaking code, and is too impatient to wait for the result and goes home for the day. I've never personally done this; no, wait, I have.

Keep testing at the unit level

Unit tests should never touch resources outside the current class. Unit tests that touch the file system, database, web service or some external resource can fail due to reasons outside the class. Permissions, service configuration, etc. can all be failure reasons, even though the code functions as expected. These types of tests are integration tests and should be kept separate, not run in a CI build and

Test for success

No, this isn't the title of the latest self-help book.

We should be testing the expected results of methods. This is typically what we do without even thinking about it when we run through the UI.

Test for failure / negative / exceptions

Tests should also be written for any potential failures or exceptions. Test for items we know are out of bounds, and make sure the code handles that situation as expected.

Provide a random generator

Sending random strings and numbers through unit tests assures that they don't get the same developer data every time. This approach will also assist in checking maximum and minimum values and lengths within objects.

Only write valuable tests

Writing unit test for the sake of writing tests provides little value to the code base. Our goal in writing unit tests is to allow developers to easily add or modify features with confidence that their changes won't break any areas of the system.