Cheatography

C# Unit-Tests - Basics Cheat Sheet

by Sérgio Ferreira (AlienEngineer) via cheatography.com/89520/cs/20379/

Arrange - Act - Assert

```
public void MyTest() {
   // Arrange - Only setup code needed by the act
step

   // Act - Only the action(s) under test

   // Assert - Verifi cation of the excepted
behavior
}
```

Tip: If any of these parts are greatly bigger than the others, look for refactoring your tests.

Test Class Organization

```
class SubjectTests {

   // fields
   int callCount = 0;
   // help methods
    private Subject MakeSu bject() =>
       new Subject();

   // test methods
    public void Test1() { }
    public void Test2() { }
    public void Test3() { }
}
```

This is just a convention. Don't leave help methods and field scattered all around the test methods.

Solitary vs Sociable

Solitary

Type of test that tests a unit without the involvement of other units.

Mocks all dependencies of the subject under test.

Sociable

Type of test that uses multiple units to verify a given behavior.

Mock only hard to manage dependencies. (e.g. external resources)

source: Working Effectively with Unit Tests by Jay Fields

? Unit-Test

We have control over all it's parts.

Runs in any order.

Doesn't depend on another test.

Doesn't produce side-effects.

Asserts observable behavior.

Tip: if any of these is false then it's not a unit-test.

Test - What's your name?

```
// Convention #1
public void Creati ng_ a_u ser _st ore s_i t_i -
n_t he_ dat abase() { ... }
public void Creati ng_ a_u ser _wi tho ut_ nam -
e_t hro ws_ exc ept ion() { ... }

// Convention #2
public void Create Use r_S tor esI nDa tab ase() {
... }
public void Create Use r_W ith out Nam e_T hro -
wsE xce ption() { ... }

// Convention #3
public void Given_ use r_w hen _cr eat ing _th -
en_ its _st ore d_i n_d ata base() { ... }
public void Given_ use r_w hen _ha s_n o_n ame -
_th en_ thr ows _ex cep tion() { ... }
```

The name of the test should have 3 parts:

- The behavior under test;
- The constraints;
- The expected behavior.

Actions on Loops

```
public void Test1() {
    for(var x in listOfInt) {
        Ass ert.Th at( Get Val ue(x), Is.True));
   }
}
```

Tip: Multiple asserts and action taken within a loop on the same test makes us ignore some cases in case of a failure.

By **Sérgio Ferreira**
(AlienEngineer)

cheatography.com/alienengineer/

Published 28th August, 2019.
Last updated 3rd September, 2019.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

C# Unit-Tests - Basics Cheat Sheet

by Sérgio Ferreira (AlienEngineer) via cheatography.com/89520/cs/20379/

Avoid some Expectations

```
// Thats how it's done
mock.V eri fy(...)

// If possible use specific value, is int.MaxValue a valid expectation?
Assert.Th at(x, Is.Gre ate rTh an(10))

// Might be null
var result = GetObj ect();
Assert.Th at( res ult.Pr operty, Is.True))

// Might throw exception somewhere other than action
[Expec tEx cep tion()]

// Look for content not types
Assert.Is Ins tan ceO fTy pe( result, typeof (So meD at
a Mod el));
```

Avoid != Never do it

Don't ignore the signs!

Sign	Outcome
A big arrange section: large dto, many parameters or many mocks.	Subject under test might be doing to much.
Tests to data model object (dto).	Reveals missing tests. DTOs will get their coverage from usage.
Tests to Exceptions.	Reveals missing tests. Exceptions will be tested by their usage.
Big test file	Can indicate duplication or the subject under test is doing to much.
Json, xml, etc	Formatted strings of any kind reveal coupling. *Except tests to formatters.*

Big file : Any file greater than 500 lines
Big section : More than 10 lines.
Many parameters : More than 3.
Many Mocks : More than 3.
Large dto: More than 10 properties.

Parameterized (NUnit)

```
[Test]
public void Test1( [Va lues(1, 2, 3)] int value) {
}
```

Parameterized (xUnit)

```
[Theory]
[Inlin eDa ta(1)]
[Inlin eDa ta(2)]
[Inlin eDa ta(3)]
public void Test1(int value) { }
```

Parameterized (MS Tests v2)

```
[DataTestMethod]
[DataR ow(1)]
[DataR ow(2)]
[DataR ow(3)]
public void Test1(int value) { }
```

Isolate - Shared data

```
static int value = 0;
public void Test1() {
    value = 10;
    doS ome thi ng( value);
}
public void Test2() {
    doS ome thi ng( value);
}
```

Static mutable state will eventually kill one or more tests.

Isolation - Thread safe tests

```
static object lockObject = new object();
public void Test1() {
    loc k(l ock Object) {
       // thread safe code
   }
}
public void Test2() {
    loc k(l ock Object) {
       // thread safe code
   }
}
```

Avoid this! Dealing with thread safety in tests adds another layer of complexity.

By **Sérgio Ferreira**
(AlienEngineer)

cheatography.com/alienengineer/

Published 28th August, 2019.
Last updated 3rd September, 2019.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com