

Arrange - Act - Assert

```
public void MyTest() {  
    // Arrange - Only setup code needed by the act  
    step  
  
    // Act - Only the action(s) under test  
  
    // Assert - Verification of the expected  
    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 MakeSubject() =>  
        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.

Mocks 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 its 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 Create_a_user_stores_information_in_database() { ... }  
public void Create_a_user_with_the_name_entered_within_expection() { ... }  
  
// Convention #2  
public void Create_Using_Store_informationDatabase() { ... }  
public void Create_Using_With_out_Name_Error_within_exception() { ... }  
  
// Convention #3  
public void Given_use_when_createing_the_entire_stored_information_and_data_base() { ... }  
public void Given_use_when_having_hands_on_name_then_throws_exception() { ... }
```

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) {  
        Assert.That(GetValue(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.

Page 1 of 2.

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

⚡ Avoid some Expectations

```
// Thats how it's done
mock.Verify(...)

// If possible use specific value, is int.MaxValue a valid expectation?
Assert.That(x, Is.GreaterThan(10))

// Might be null
var result = GetObject();
Assert.That(result.Property, Is.True)

// Might throw exception somewhere other than action
[ExceptionOccurrence()]

// Look for content not types
Assert.IsInstanceOf(result, typeof(SomeObject));
a Model);
```

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 too 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 too much.
Json, xml, etc	Formatted strings of any kind reveal coupling. <i>Except tests to formatters.</i>

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 (xUnit)

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

👍 Parameterized (MS Tests v2)

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

⚡ Isolate - Shared data

```
static int value = 0;
public void Test1() {
    value = 10;
    doSomething(value);
}

public void Test2() {
    doSomething(value);
}
```

Static mutable state will eventually kill one or more tests.

⚡ Isolation - Thread safe tests

```
static object lockObject = new object();
public void Test1() {
    lock(lockObject) {
        // thread safe code
    }
}

public void Test2() {
    lock(lockObject) {
        // thread safe code
    }
}
```

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

👍 Parameterized (NUnit)

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



By Sérgio Ferreira
(AlienEngineer)

cheatography.com/alienengineer/

Published 28th August, 2019.

Last updated 3rd September, 2019.

Page 2 of 2.

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