Cheatography

C# Unit-Tests - Advanced Cheat Sheet

by Sérgio Ferreira (AlienEngineer) via cheatography.com/89520/cs/20391/

Implementation testing

```
public void Test() {
  // Arrange
  var mock = new Mock<IActionInterface>();
  var subject = new MySubject(mock.Object);

  // Act
  subject.DoSomething();

  // Assert
  mock.Verify(e => e.Store(It.IsAny<int>()),
Times.Once);
}
```

- Locks tests with the dependency:
  - The test method is aware of IActionInterface.
  - Creates duplicated code.
- Changes to MySubject constructor forces changes throughout all the tests.

Verify vs Callbacks

```
// If one of these equality comparisons fail we
just know that MyAction wasn't called with those
arguments.
mock.Verify(e => e.MyAction(
  It.Is<ParameterType>(param =>
    param.Field1 == "some value" &&
    param.Field2 == 3),
  Times.Once);
// Instead:
ParameterType calledArgs = null;
mock
  .Setup(e => e.MyAction(It.IsAny<ParameterType>
()))
  .Callback<ParameterType>(param => calledArgs =
param);
Assert.That(calledArgs, Is.Not.Null);
Assert.That(calledArgs.Field1, Is.EqualTo("some
value"));
Assert.That(calledArgs.Field2, Is.EqualTo(3));
```

Tip: Avoid stating in your tests how things were done, instead strive to describe what happened. **Observable Behavior**

Behavior testing

```
private int _someVariable;

private MySubject MakeSubject()
{
  var mock = new Mock<IActionInterface>();

  mock.Setup(e => e.Action(It.IsAny<int>()))
      .Callback<Int>(s => _someVariable= s);

  return new MySubject(mock.Object);
}

private bool WasStored(int value) =>
  _someVariable == value;

[Test]
public void SomeTestMethod()
{
  // Arrange
  var subject = MakeSubject();

  // Act
  subject.DoSomething();

  // Assert
  Assert.True(WasStored(5));
}
```

- Changes to dependencies are done in one place only.
- Focuses on Behavior not implementation.

Dependencies - Common missed tests

What happens when the dependency..

- throws exception?
- returns null?
- returns wrong format? (e.g. expected json)

Tip: Testing is about making sure we have answers to questions. Testing is done when we don't have doubts, therefore no more questions.

By **Sérgio Ferreira** (AlienEngineer)

cheatography.com/alienengineer/

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

Cheatography

C# Unit-Tests - Advanced Cheat Sheet

by Sérgio Ferreira (AlienEngineer) via cheatography.com/89520/cs/20391/

Web Apis

```
internal abstract class WebApiTests {
  private WebApplicationFactory<Startup> _factory;

  protected WebApiTests() =>
    _factory = new WebApplicationFactory<Startup>
();

  protected HttpClient GetClient() =>
    _factory.CreateClient();
}
class MyApiControllerTests : WebApiTests {
  public async Task requestTest() {
    // Arrange
    var client = GetClient();

    // Act
    var response = await
client.GetAsync("api/someuri");

    // Assert
    response.EnsureSuccessStatusCode();
    var data = await
response.Content.ReadAsAsync<Model>();

    // Assert data ...
  }
}
```

using **Microsoft.AspNetCore.Mvc.Testing** library for auto mocking out the http layer.

Files

```
// Hide file access through Stream Factory
public interface IStreamFactory
{
  Stream OpenStream();
  void CloseStream(Stream stream);
}
// On test file
private IStreamFactory MakeStreamStorage()
{
  // Create one isolated MemoryStream per test run!
  _stream = new MemoryStream();
  var mock = new Mock<IStreamFactory>();
  mock.Setup(s => s.OpenStream()).Returns(_st-
ream);
  return mock.Object;
}
```

Test arrange with files

```
void Test() {
  // slow & can fail
  var data = File.ReadAllText(@"Data.json");
  // slow & can fail
  var x = JsonConvert.DeserializeObject<MyModel>(-
data);
  // None of the above lines have anything to do
with the test. It's all about Arrange section.
}
```

Tip: If you feel the need to do this, probably your subject under test is doing to much. See **Don't ignore the signs!**

By **Sérgio Ferreira** (AlienEngineer)

cheatography.com/alienengineer/

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