Unit testing in real life

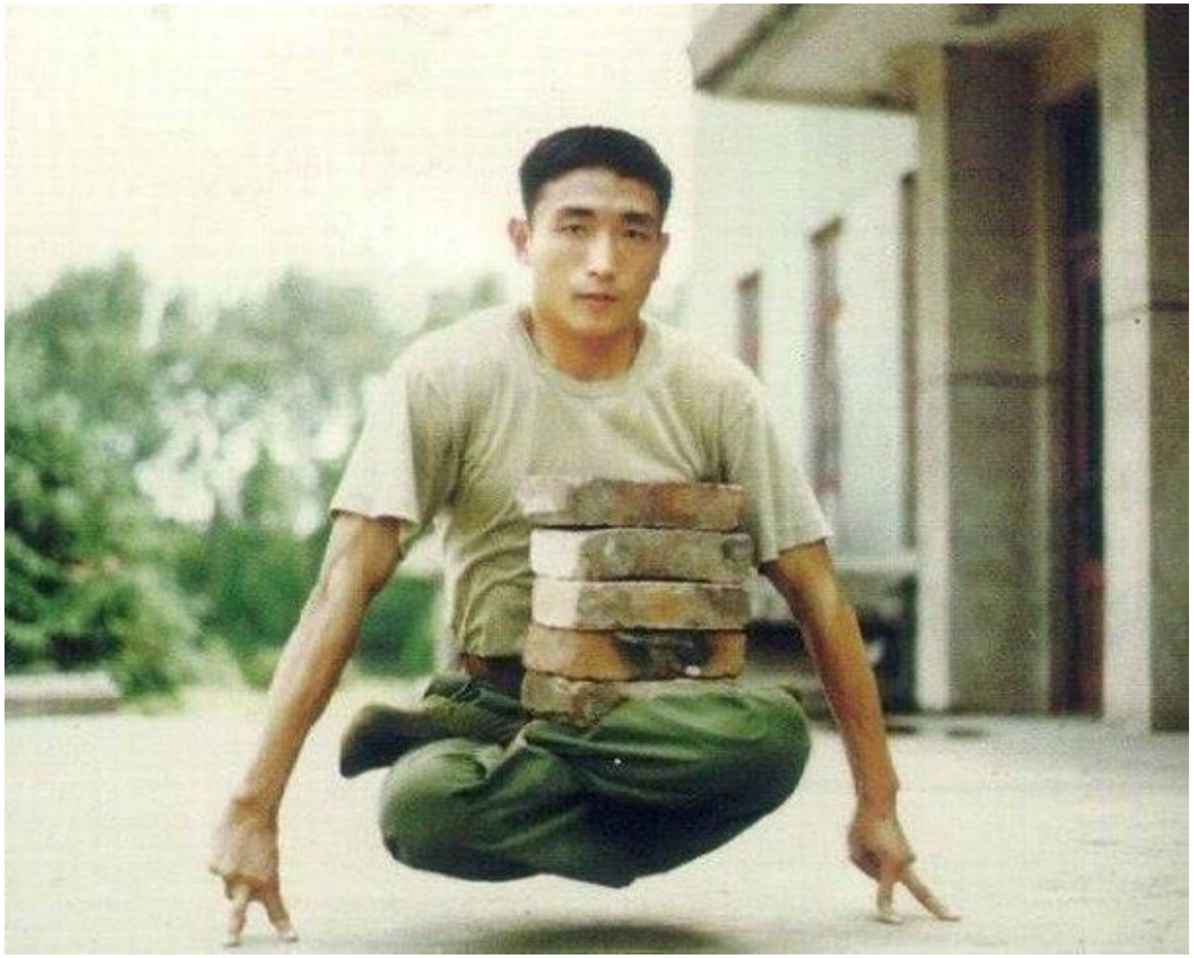
Vladimir Alekseichenko

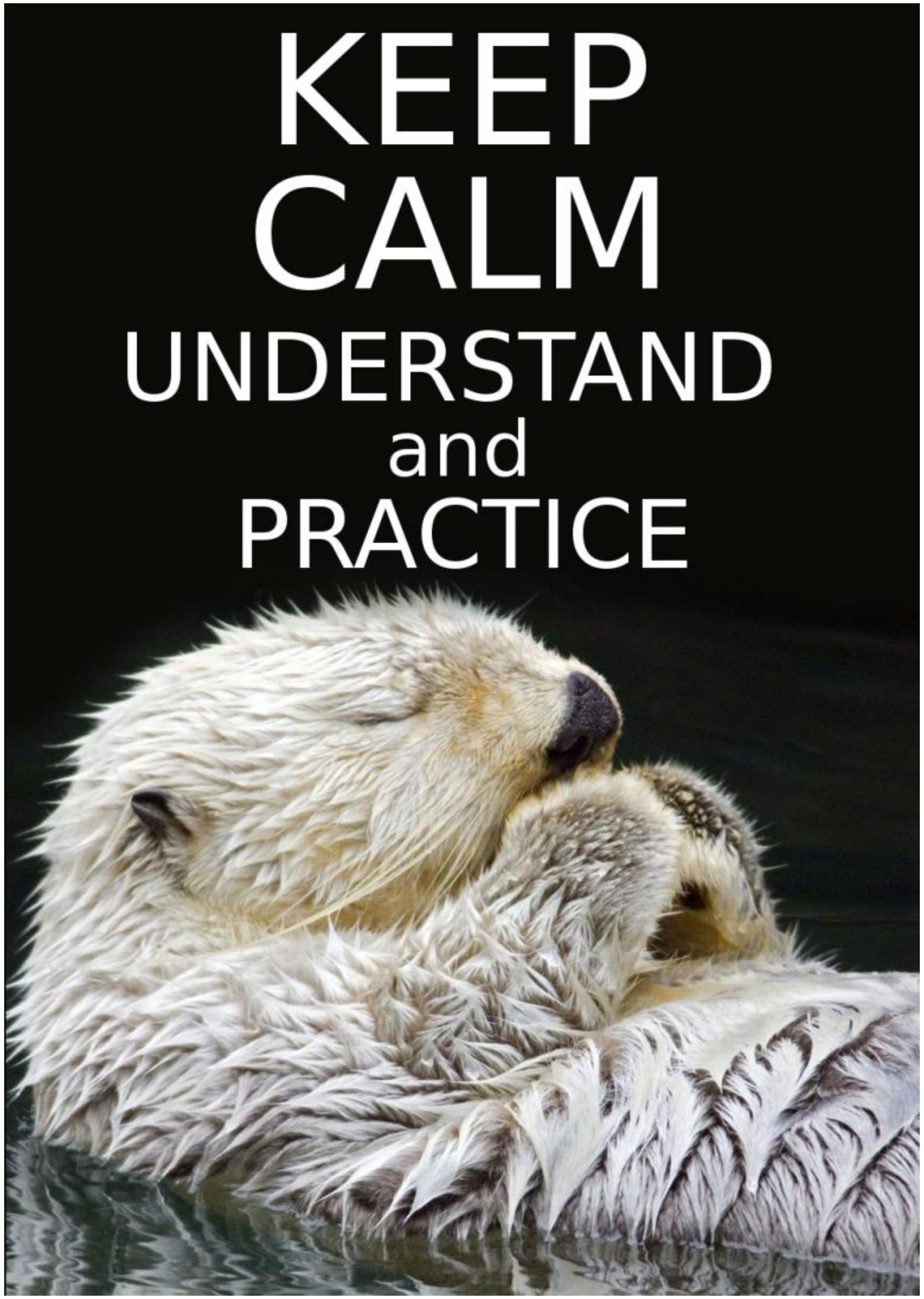
Agenda

- Why you don't use unit testing?
- Excuses for not testing
- Why even bother?
- Test automation pyramid
- Double objects
- The pillars of good tests

Good practices make life easier

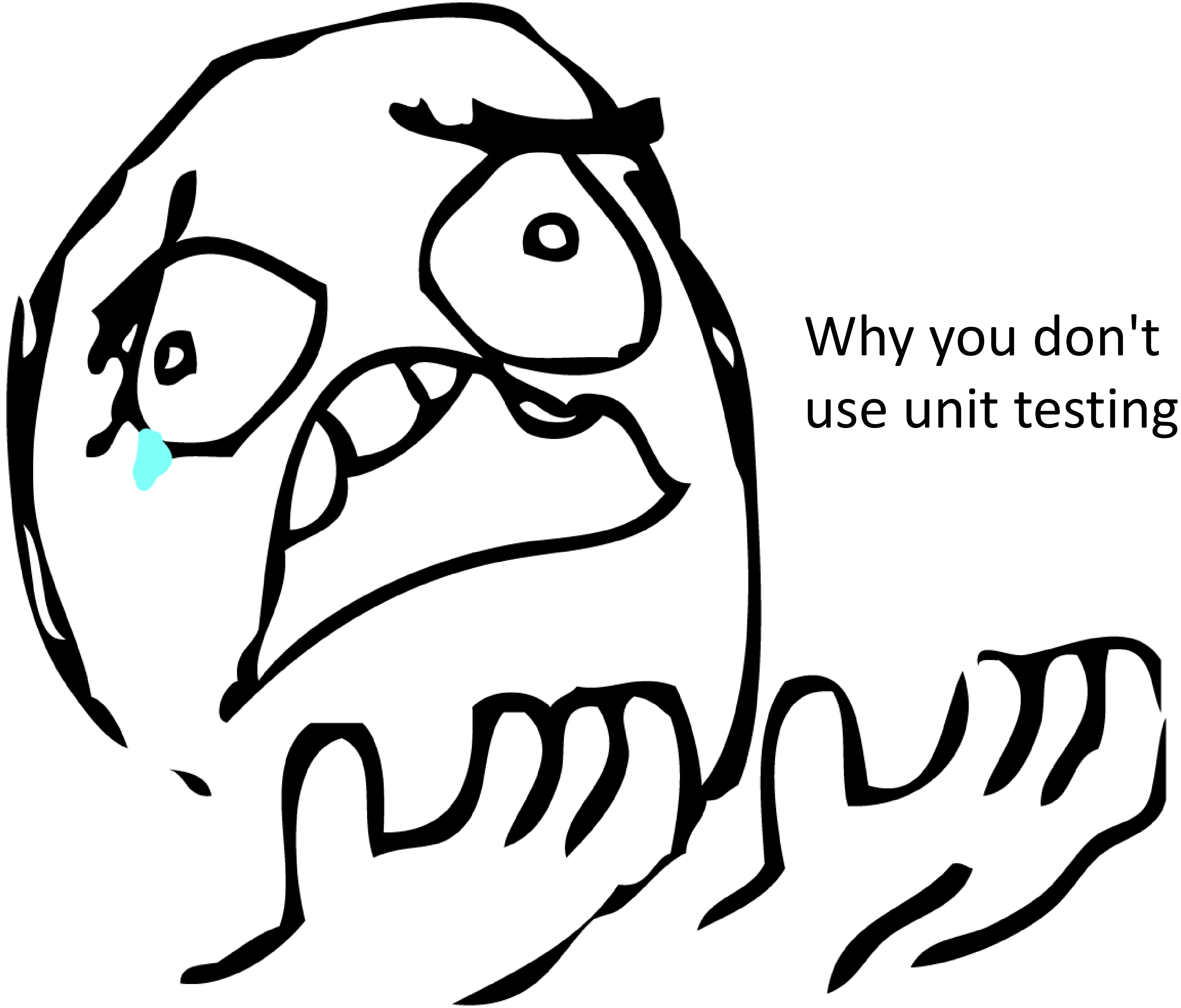
Good practices are not easy




KEEP
CALM
UNDERSTAND
and
PRACTICE

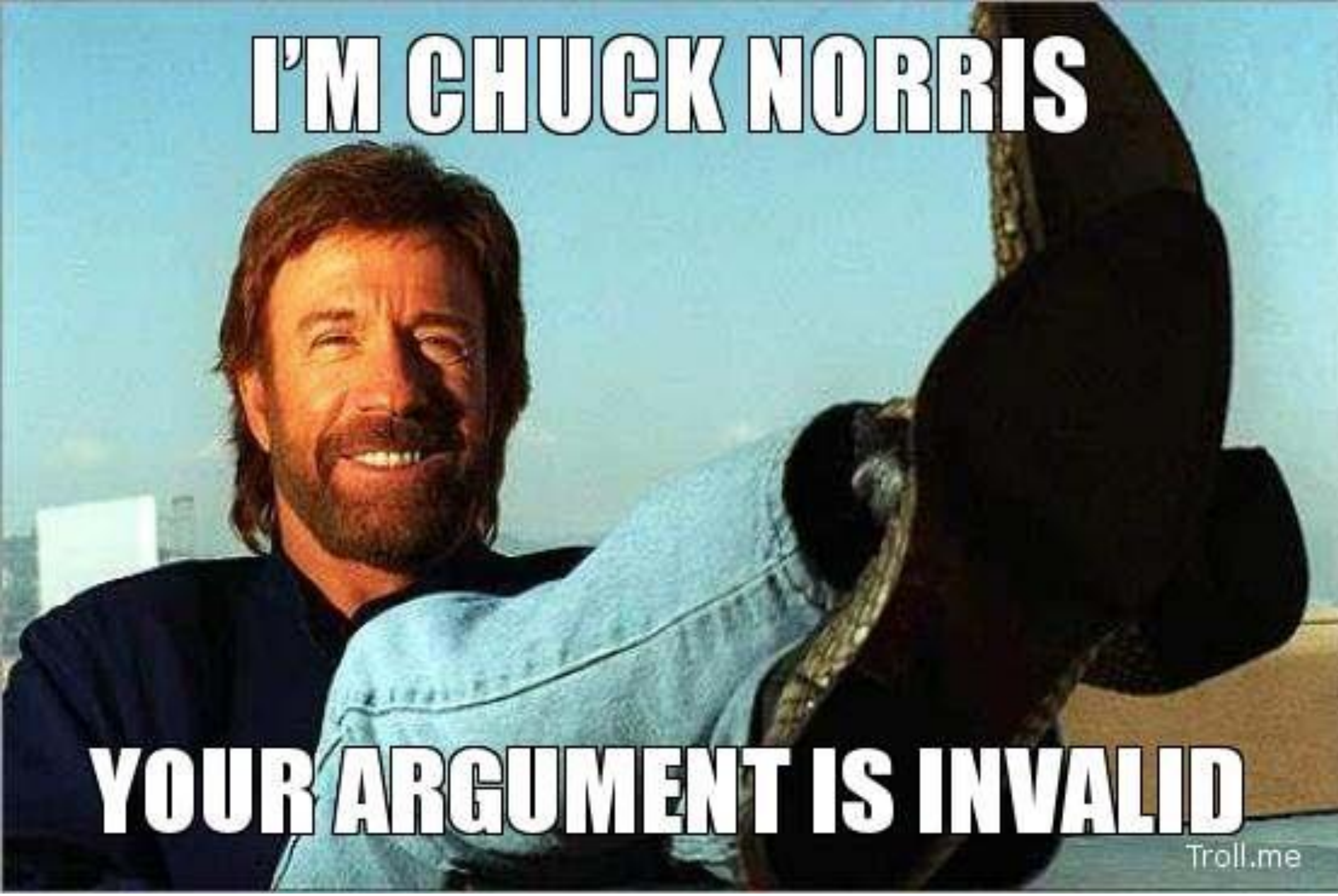
The question for you

http://das-team-nms.de/?page_id=204&paged=6



Why you don't use unit testing?

fckyea


I'M CHUCK NORRIS
YOUR ARGUMENT IS INVALID
Troll.me

Excuses for not testing

- It takes too long to run the tests.
- My legacy code is impossible to test.
- It's not my job to test my code.
- I don't really know how the code is supposed to behave so I can't test it.

Excuses for not testing *(continued)*

- I’m being paid to write code, not to write tests.
- I feel guilty about putting testers and QA staff out of work.
- My company won’t let me run unit tests on the live system.
- …

Why even bother?

Managing fear

http://orecommunications.com/wordpress/2009/10/ouray-ice-festival-jan-7-10-2010

Testable architecture (robust)

http://rolexblog.blogspot.com/2012_12_01_archive.html

Be careful!

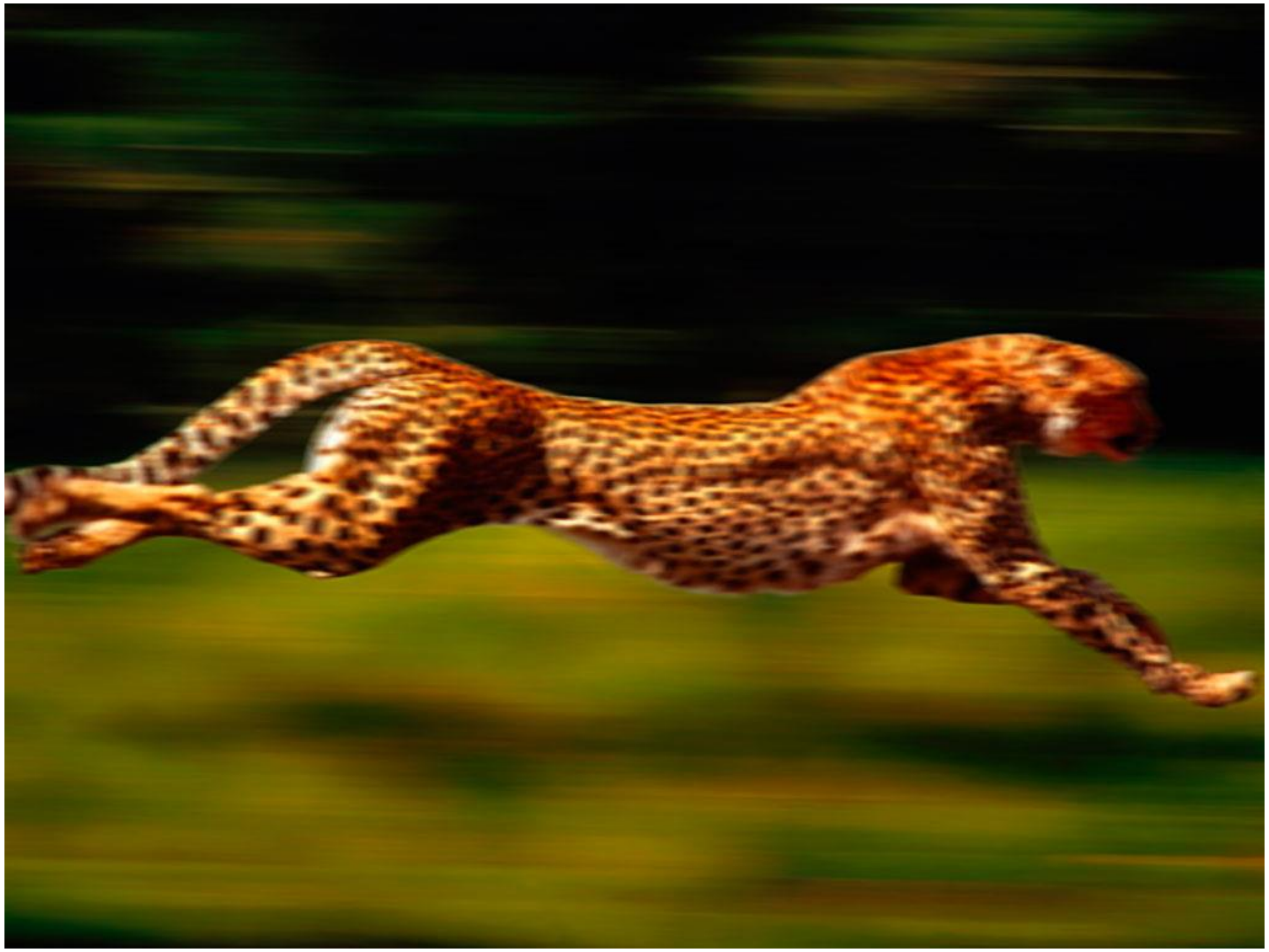
Good architecture always is **testable,**

but **not** all

testable architecture is **good**!

http://www.intellect-dt.ru/news/detail/283

Fast feedback

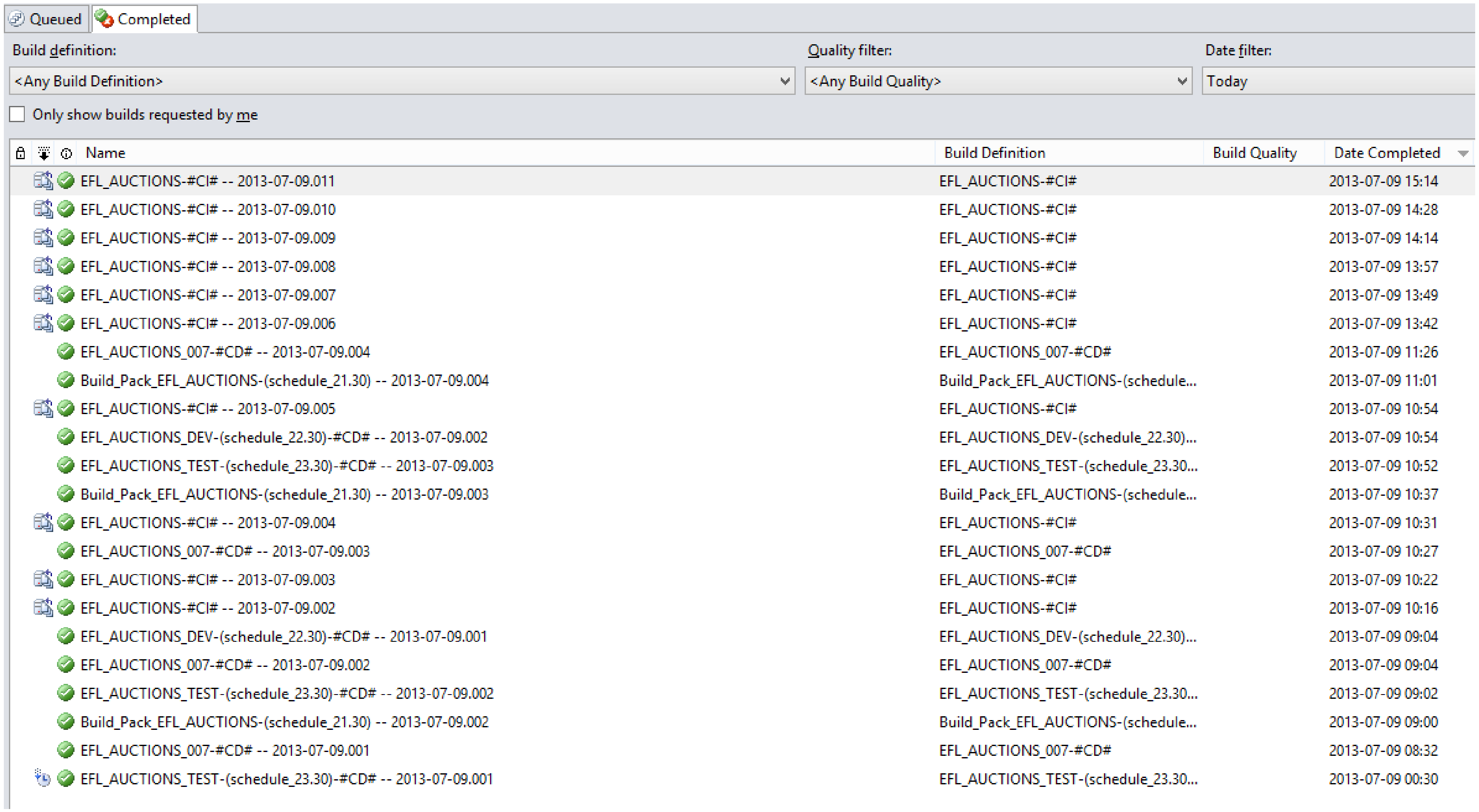


http://community.devexpress.com/blogs/aspnet/archive/2009/07/20/asp-net-performance-improvements-in-devexpress-2009-volume-2.aspx

Effective team communication

http://itsveryvisual.com/importance-clear-communication-work-place.html

Continuous integration



Build definition: <Any Build Definition>

Quality filter: <Any Build Quality>

Date filter: Today

☐ Only show builds requested by me

Name	Build Definition	Build Quality	Date Completed
EFL_AUCTIONS-#CI# -- 2013-07-09.011	EFL_AUCTIONS-#CI#		2013-07-09 15:14
EFL_AUCTIONS-#CI# -- 2013-07-09.010	EFL_AUCTIONS-#CI#		2013-07-09 14:28
EFL_AUCTIONS-#CI# -- 2013-07-09.009	EFL_AUCTIONS-#CI#		2013-07-09 14:14
EFL_AUCTIONS-#CI# -- 2013-07-09.008	EFL_AUCTIONS-#CI#		2013-07-09 13:57
EFL_AUCTIONS-#CI# -- 2013-07-09.007	EFL_AUCTIONS-#CI#		2013-07-09 13:49
EFL_AUCTIONS-#CI# -- 2013-07-09.006	EFL_AUCTIONS-#CI#		2013-07-09 13:42
EFL_AUCTIONS_007-#CD# -- 2013-07-09.004	EFL_AUCTIONS_007-#CD#		2013-07-09 11:26
Build_Pack_EFL_AUCTIONS-(schedule_21.30) -- 2013-07-09.004	Build_Pack_EFL_AUCTIONS-(schedule...		2013-07-09 11:01
EFL_AUCTIONS-#CI# -- 2013-07-09.005	EFL_AUCTIONS-#CI#		2013-07-09 10:54
EFL_AUCTIONS_DEV-(schedule_22.30)-#CD# -- 2013-07-09.002	EFL_AUCTIONS_DEV-(schedule_22.30)...		2013-07-09 10:54
EFL_AUCTIONS_TEST-(schedule_23.30)-#CD# -- 2013-07-09.003	EFL_AUCTIONS_TEST-(schedule_23.30...		2013-07-09 10:52
Build_Pack_EFL_AUCTIONS-(schedule_21.30) -- 2013-07-09.003	Build_Pack_EFL_AUCTIONS-(schedule...		2013-07-09 10:37
EFL_AUCTIONS-#CI# -- 2013-07-09.004	EFL_AUCTIONS-#CI#		2013-07-09 10:31
EFL_AUCTIONS_007-#CD# -- 2013-07-09.003	EFL_AUCTIONS_007-#CD#		2013-07-09 10:27
EFL_AUCTIONS-#CI# -- 2013-07-09.003	EFL_AUCTIONS-#CI#		2013-07-09 10:22
EFL_AUCTIONS-#CI# -- 2013-07-09.002	EFL_AUCTIONS-#CI#		2013-07-09 10:16
EFL_AUCTIONS_DEV-(schedule_22.30)-#CD# -- 2013-07-09.001	EFL_AUCTIONS_DEV-(schedule_22.30)...		2013-07-09 09:04
EFL_AUCTIONS_007-#CD# -- 2013-07-09.002	EFL_AUCTIONS_007-#CD#		2013-07-09 09:04
EFL_AUCTIONS_TEST-(schedule_23.30)-#CD# -- 2013-07-09.002	EFL_AUCTIONS_TEST-(schedule_23.30...		2013-07-09 09:02
Build_Pack_EFL_AUCTIONS-(schedule_21.30) -- 2013-07-09.002	Build_Pack_EFL_AUCTIONS-(schedule...		2013-07-09 09:00
EFL_AUCTIONS_007-#CD# -- 2013-07-09.001	EFL_AUCTIONS_007-#CD#		2013-07-09 08:32
EFL_AUCTIONS_TEST-(schedule_23.30)-#CD# -- 2013-07-09.001	EFL_AUCTIONS_TEST-(schedule_23.30...		2013-07-09 00:30

Why even bother? *(continued)*

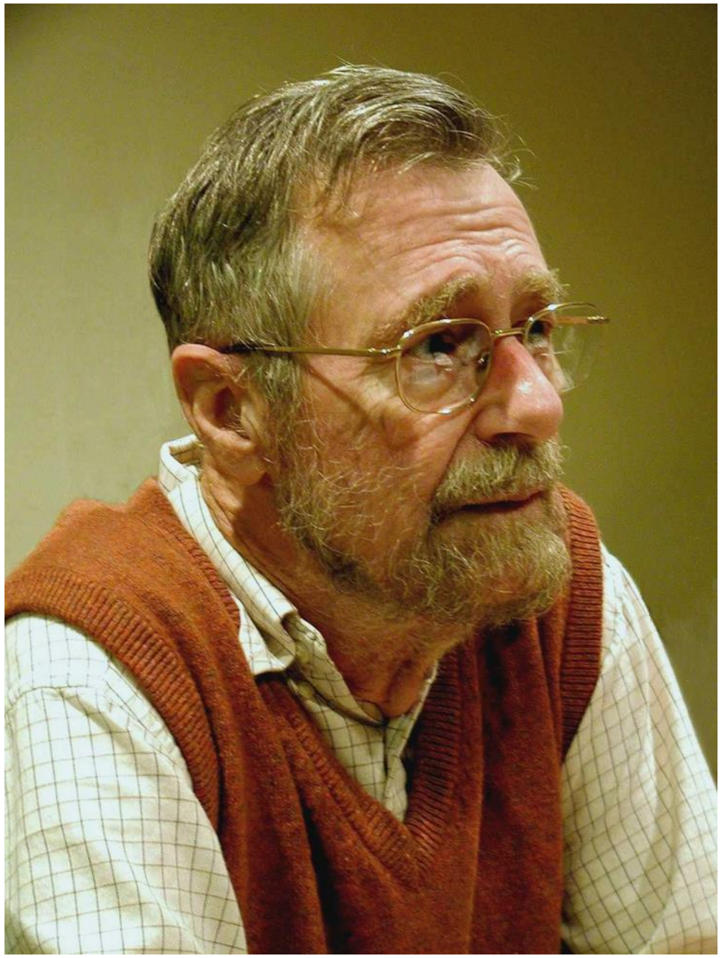
- Tests **reduce bugs** in new features.
- Tests **reduce bugs** in existing features.
- Tests are good **documentation**.
- Tests **reduce the cost** of change.
- Tests **improve design**.
- Tests **allow refactoring**.

Why even bother? *(continued)*

- Tests constrain features.
- Tests defend against other programmers.
- Testing is fun.
- Testing forces you to slow down and think.
- Testing makes development faster.
- Tests reduce **fear.**

Some quotes to remember

Edsger Dijkstra



http://en.wikipedia.org/wiki/Edsger_W._Dijkstra

Limits of testing

„Program testing can be used to show the presence of bugs, but never to show their absence!"

-- Edsger Dijkstra

Martin Fowler

http://en.wikipedia.org/wiki/Edsger_W._Dijkstra

„Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead.”

--Martin Fowler

Michael Feathers

http://www.flickr.com/photos/fraserspeirs/3395595360

Code without tests is **bad code**

- It doesn't matter how well written it is.
- It doesn't matter how pretty or object-oriented or well-encapsulated it is.

Without them, we really don't know if our code is getting better or worse.

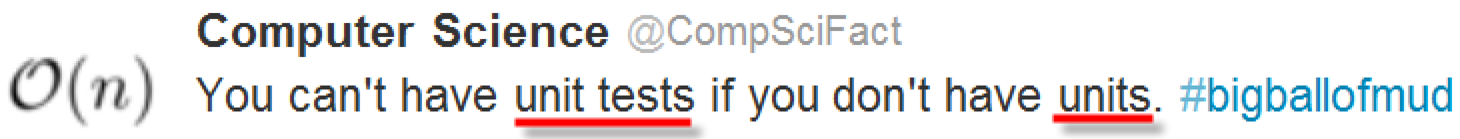
Working Effectively with Legacy Code by *Michael Feathers*

What is **unit testing**?

Unit tests are performed to prove that a piece of code does what the developer thinks it should do.

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

What is unit?



Computer Science @CompSciFact

You can't have unit tests if you don't have units. #bigballofmud

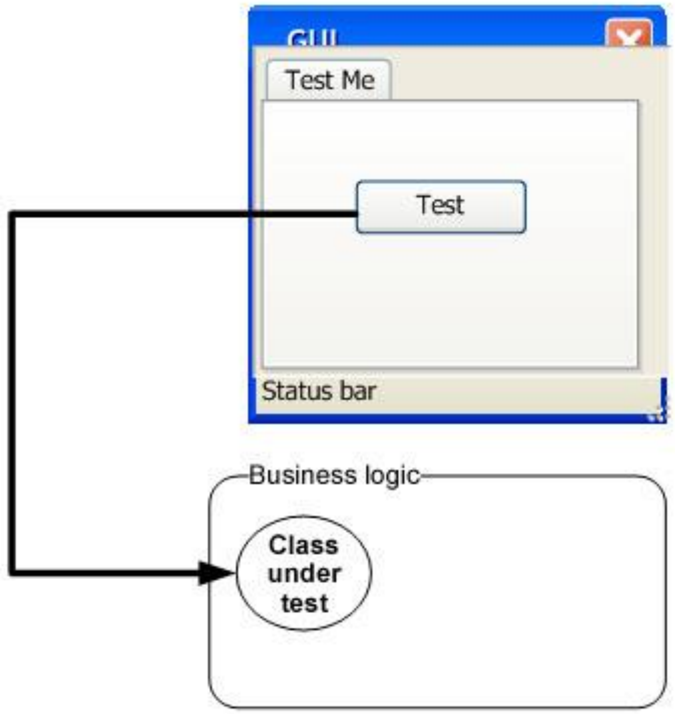
In object-oriented systems, these units **typically** are classes and methods.

Unit

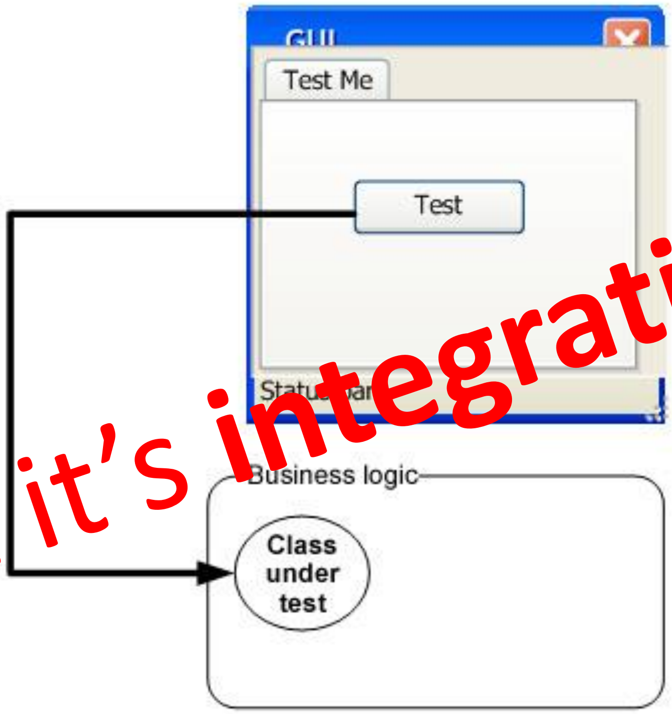
Casually refers to low-level test cases written in the same language as the production code, which directly access its objects and members.

http://c2.com/cgi/wiki?UnitTest

Class under test



Class under test



…but it's **integration** test!

http://imgur.com/gallery/peXGg

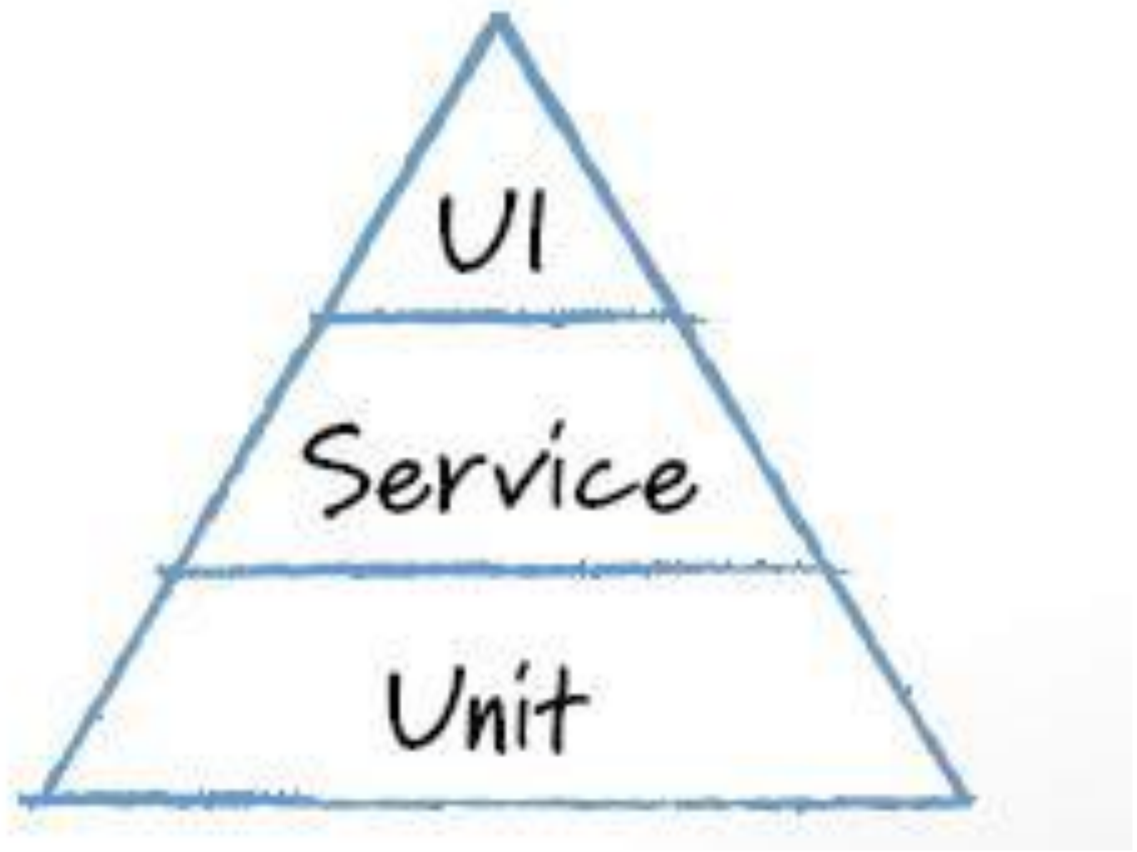
We need to go deeper

http://imgur.com/gallery/peXGg

Mike Cohn

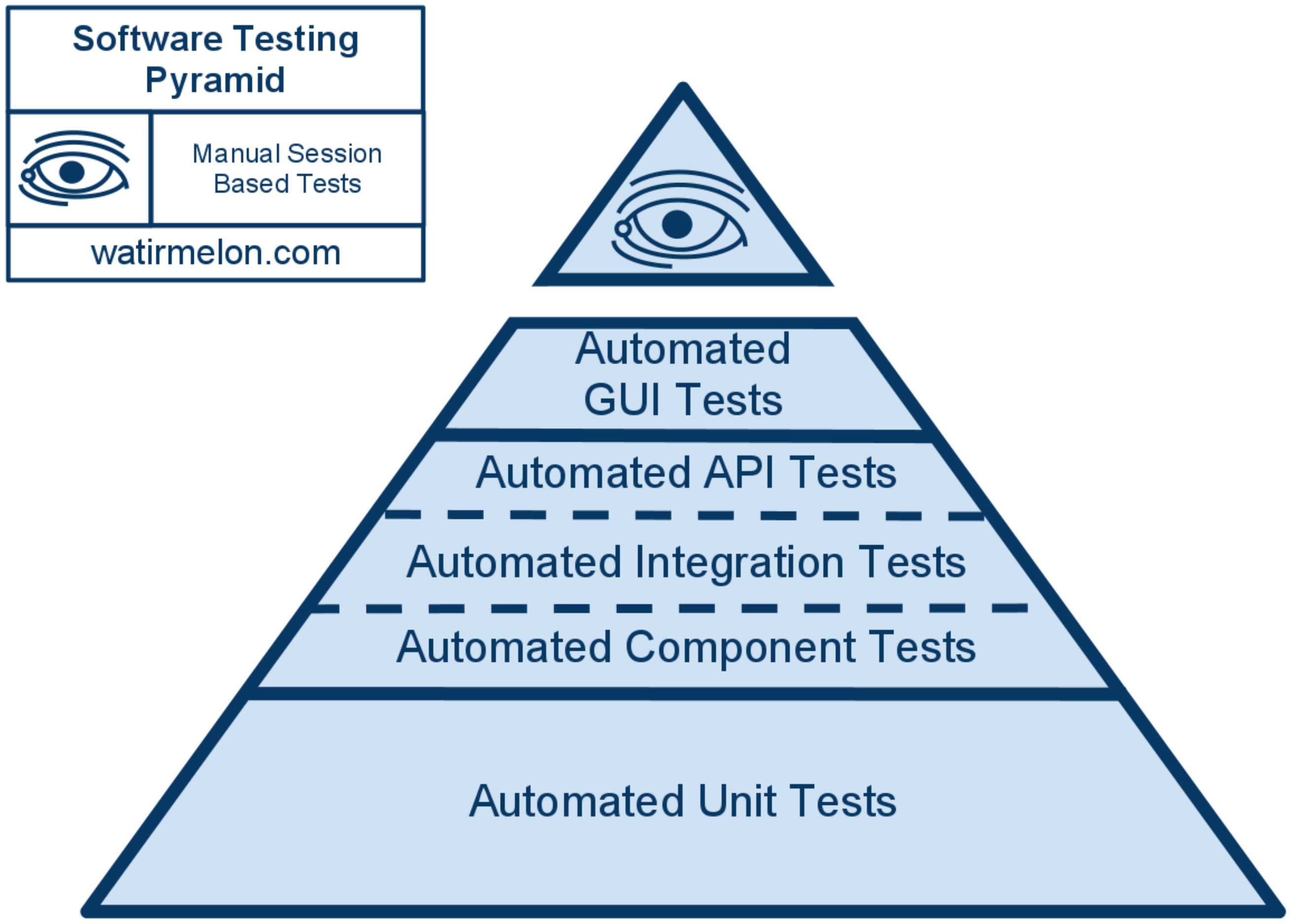
http://en.wikipedia.org/wiki/Mike_Cohn

Test automation pyramid
Mike Cohn



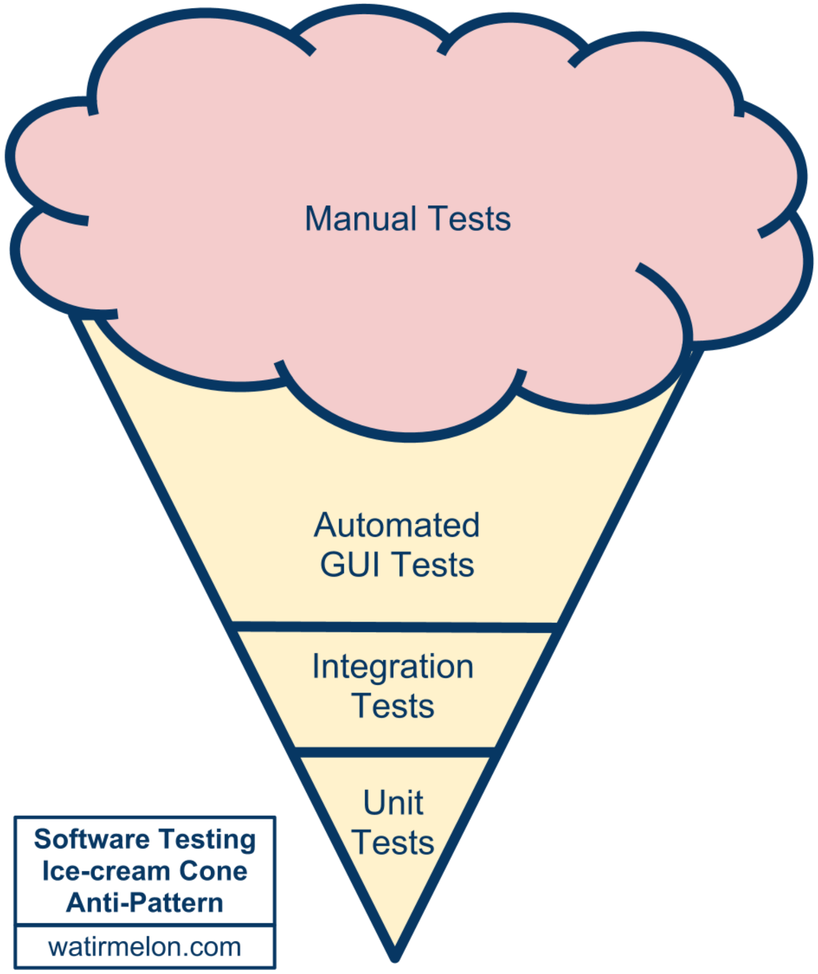
http://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

Software testing pyramid



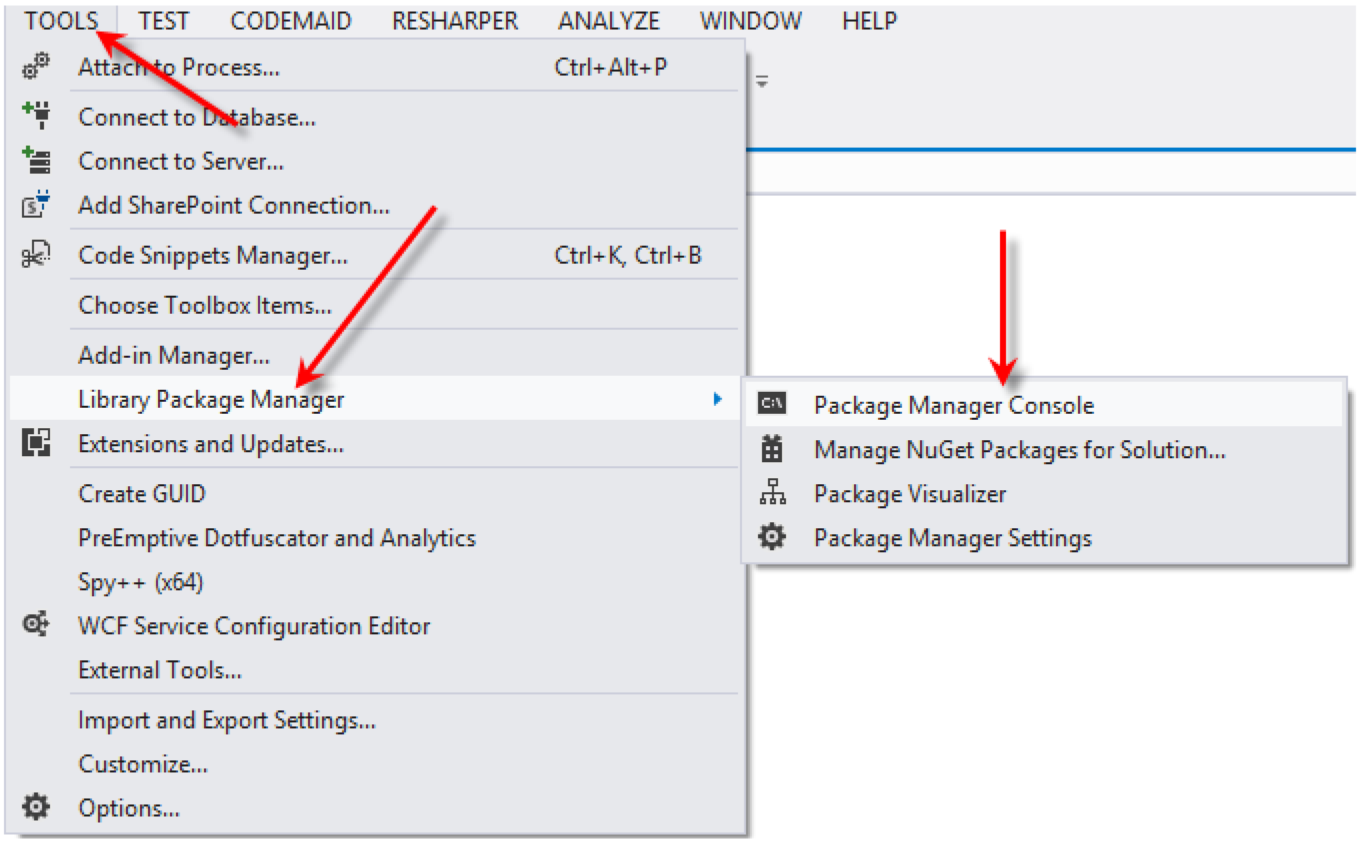
http://watirmelon.com/tag/software-testing-pyramid/

Software testing ice-cream cone **anti-pattern**

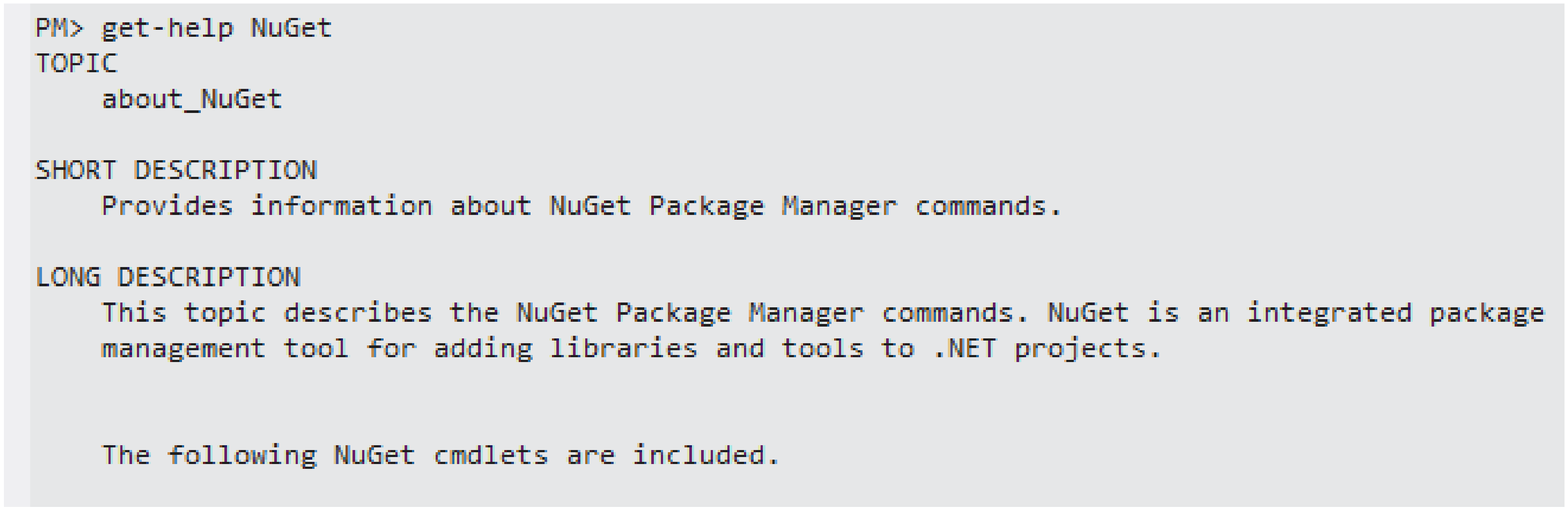


http://watirmelon.com/tag/software-testing-pyramid/

Package Manager Console - NuGet

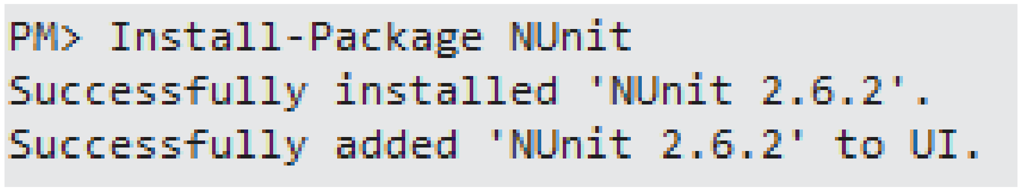


Package Manager Console - NuGet

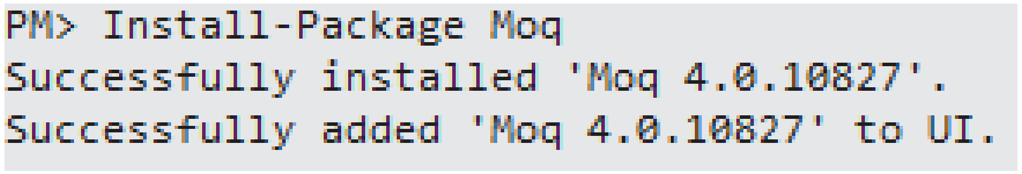


```
PM> get-help NuGet
TOPIC
    about_NuGet

SHORT DESCRIPTION
    Provides information about NuGet Package Manager commands.

LONG DESCRIPTION
    This topic describes the NuGet Package Manager commands. NuGet is an integrated package
    management tool for adding libraries and tools to .NET projects.


    The following NuGet cmdlets are included.
```

Package Manager Console - NuGet



```
PM> Install-Package NUnit
Successfully installed 'NUnit 2.6.2'.
Successfully added 'NUnit 2.6.2' to UI.
```

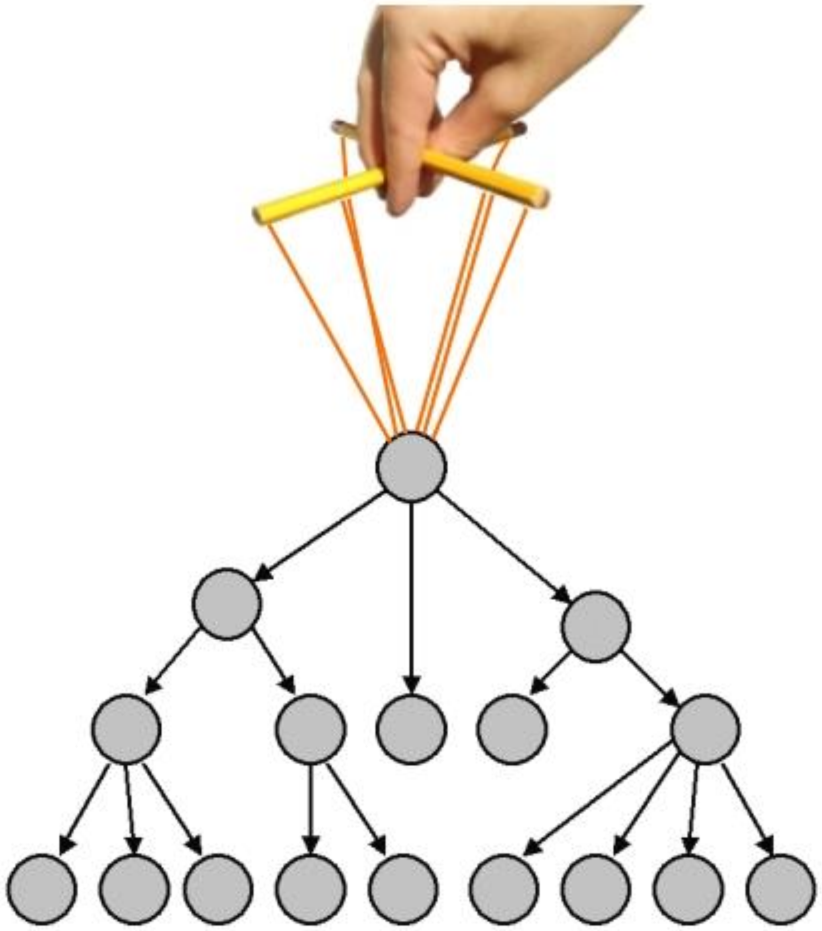


```
PM> Install-Package Moq
Successfully installed 'Moq 4.0.10827'.
Successfully added 'Moq 4.0.10827' to UI.
```

The first task

http://internationalbudget.org/blog/2009/06/11/cash-or-work-what-do-people-want/

Test hierarchies and organization



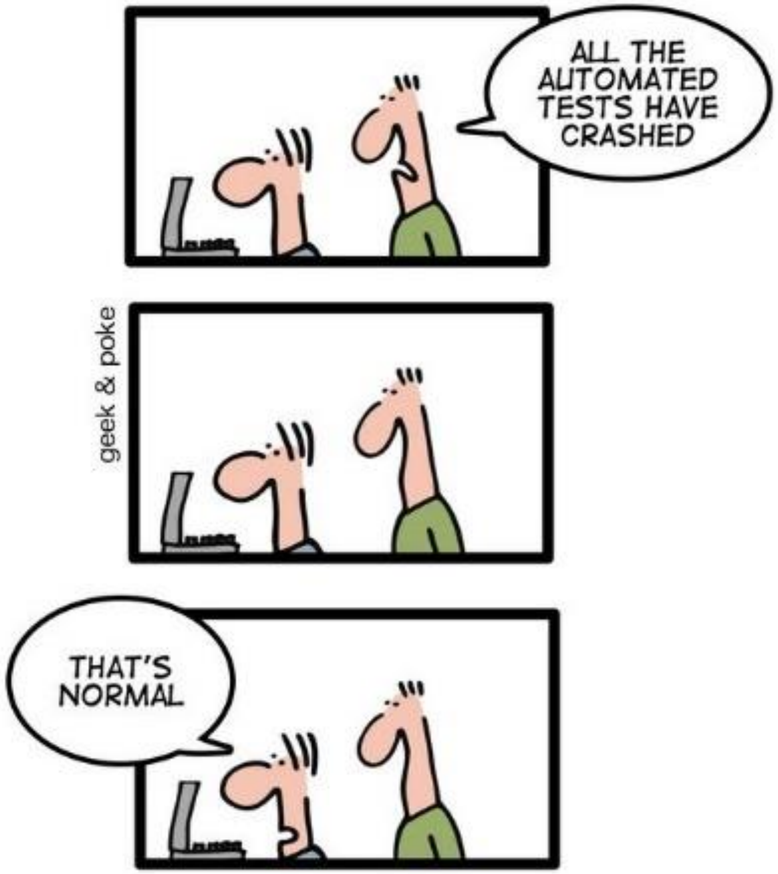
Mapping out tests based on speed and type

- The human factor of separating unit from integration tests.
- The **safe green zone**.

*Separate your **integration** and **unit tests** into separate places. If some tests in the safe green zone don't pass, there's a **real problem**, not a (false positive) configuration problem in the test.*

GEEK & POKE'S LIST OF BEST PRACTICES

TODAY: CONTINUOUS INTEGRATION GIVES YOU THE COMFORTING FEELING TO KNOW THAT EVERYTHING IS NORMAL



http://geekandpoke.typepad.com/geekandpoke/2010/10/geekpokes-list-of-best-practices-today-continuous-integration.html

Ensuring tests are part of source control

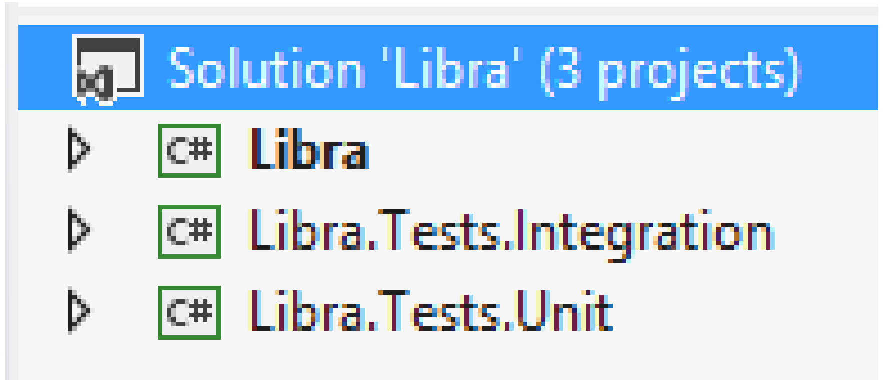
Tests ***must*** *be* part of **source control**. The test code that you write needs to reside in a source control repository, just like your **real production code**.

Mapping test classes to code under test

- Mapping tests to projects.
- Mapping tests to classes.
  - One test class per class under test.
  - One test class per feature.
- Mapping tests to specific methods.

Mapping tests to projects

Create a project to contain the tests, and give it the same name as the project under test, adding [**.Tests**] to the end of the name.



Mapping tests to classes

- One test class per class under test

  LoginManagerTests

- One test class per feature

  LoginManagerTestsChangePassword

Basic rules for placing and naming tests

Object to be tested	Object to create on the testing side
Project	Create a test project named [*ProjectUnderTest*].Tests.
Class	For each class, create at least one class with the name [*ClassName*]Tests.
Method	For each method, create at least one test method with the following name: [*MethodName*]_[*StateUnderTest*]_[*ExpectedBehavior*].

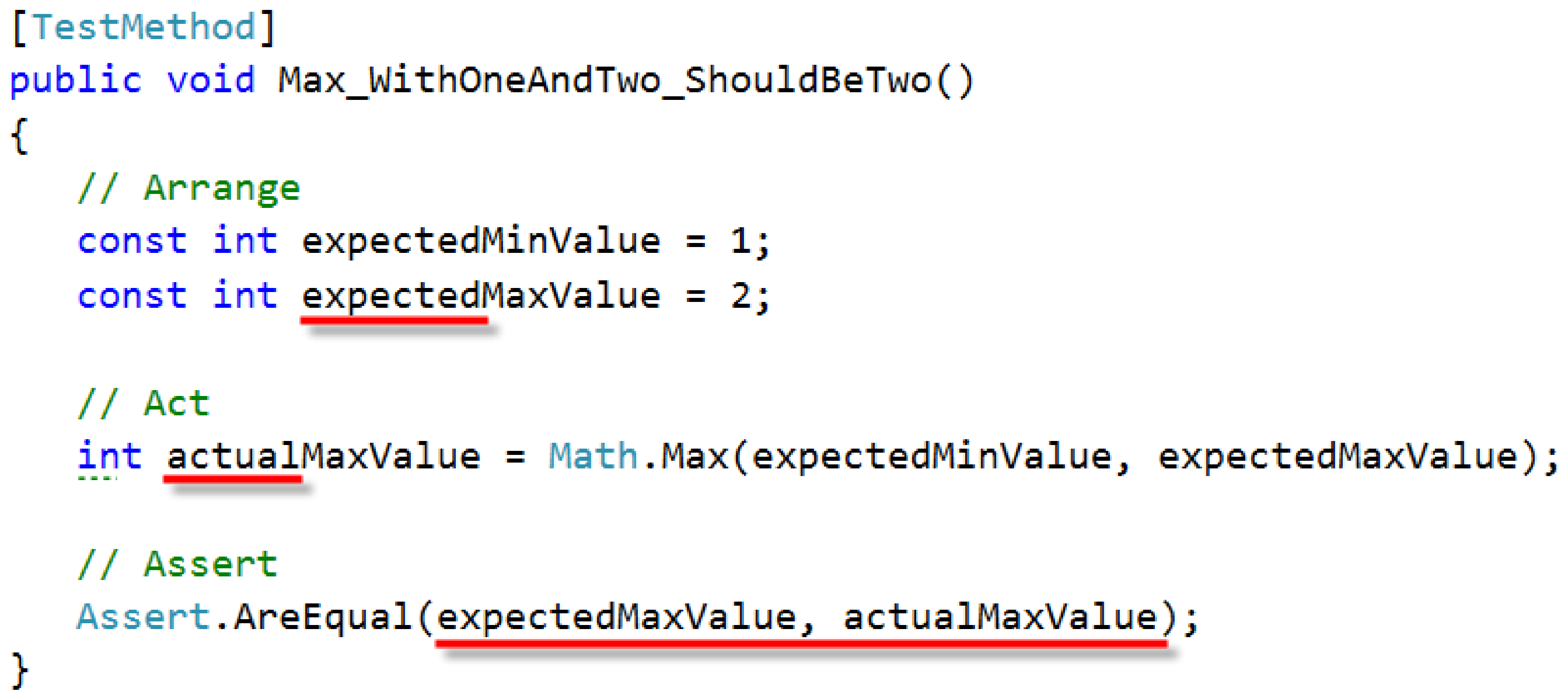
The Art of Unit Testing: With Examples in .Net by *Roy Osherove*.

The AAA pattern for **unit tests**

Bill Wake coined the term 3A (in 2003) for this:

- ***Arrange*** objects, creating and setting them up as necessary.
- ***Act*** on an object.
- ***Assert*** that something is as expected.

The AAA pattern for **unit tests** *(continued)*



```
[TestMethod]
public void Max_WithOneAndTwo_ShouldBeTwo()
{
    // Arrange
    const int expectedMinValue = 1;
    const int expectedMaxValue = 2;

    // Act
    int actualMaxValue = Math.Max(expectedMinValue, expectedMaxValue);

    // Assert
    Assert.AreEqual(expectedMaxValue, actualMaxValue);
}
```

The second task

Roy Osherove

http://osherove.com/

The Art of Unit Testing:

With Examples in .Net

http://www.enggbooks.com/books/detail/THE+ART+OF+UNIT+TESTING+WITH+EXAMPLES+IN+.NET/5922/

The pillars of good tests

- Trustworthiness
- Maintainability
- Readability

The pillars of good tests

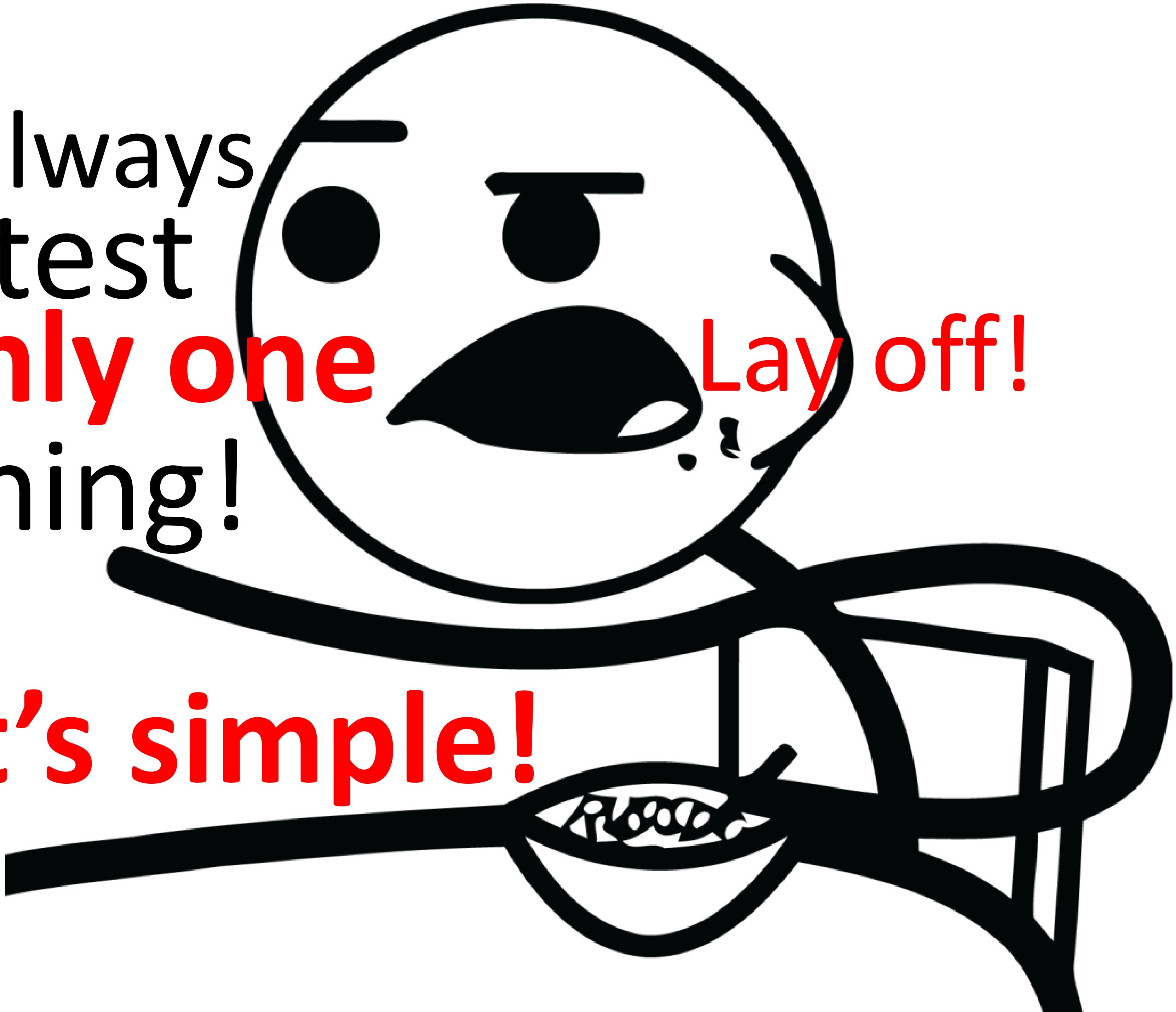
Trustworthiness

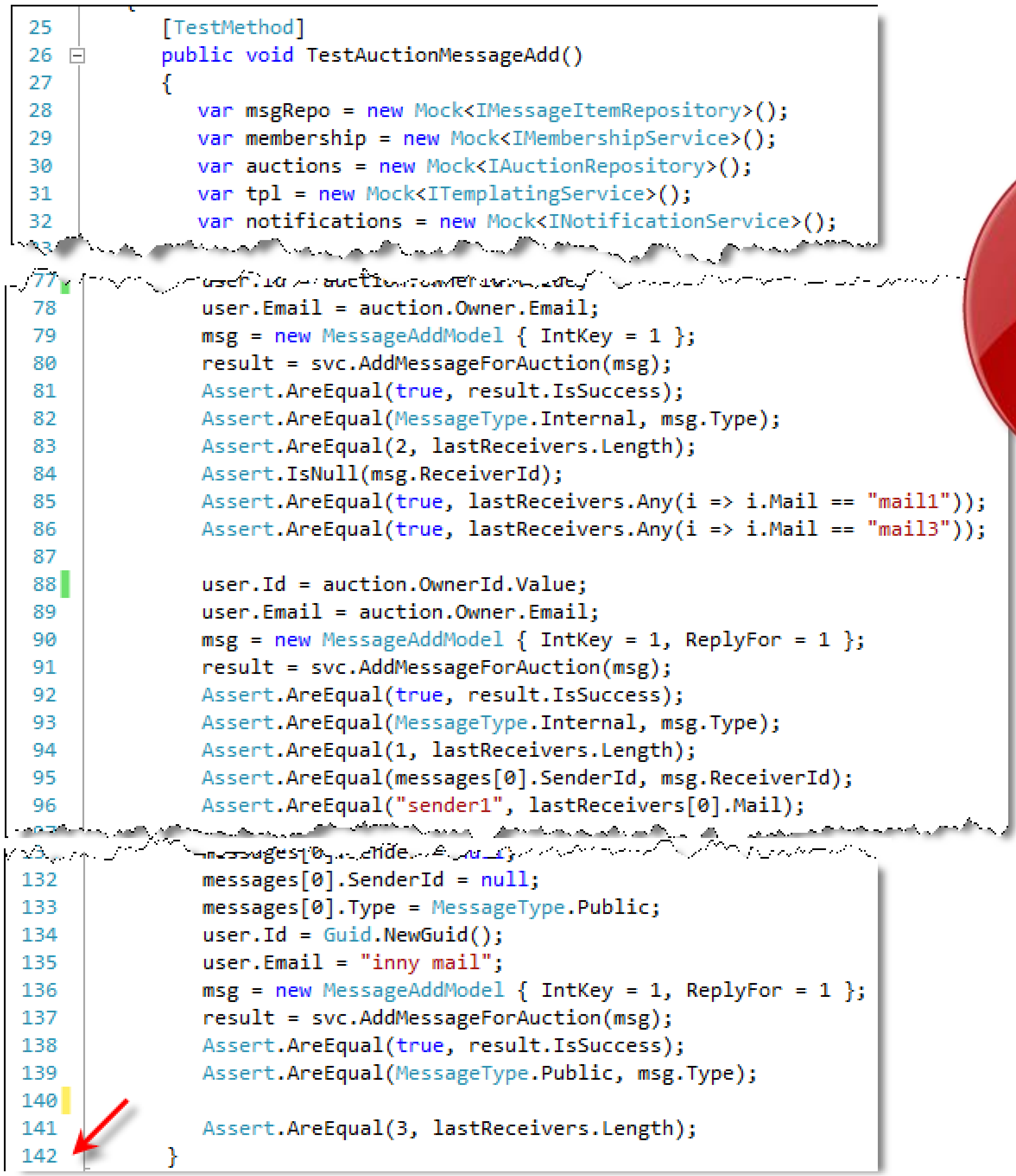
Trustworthy tests don’t have bugs, and they test the right things.

Trustworthiness

- Decide when to remove or change tests.
- Avoid test logic.
- Test only one thing.
- Make tests easy to run.
- Assure code coverage.

Test only one thing!

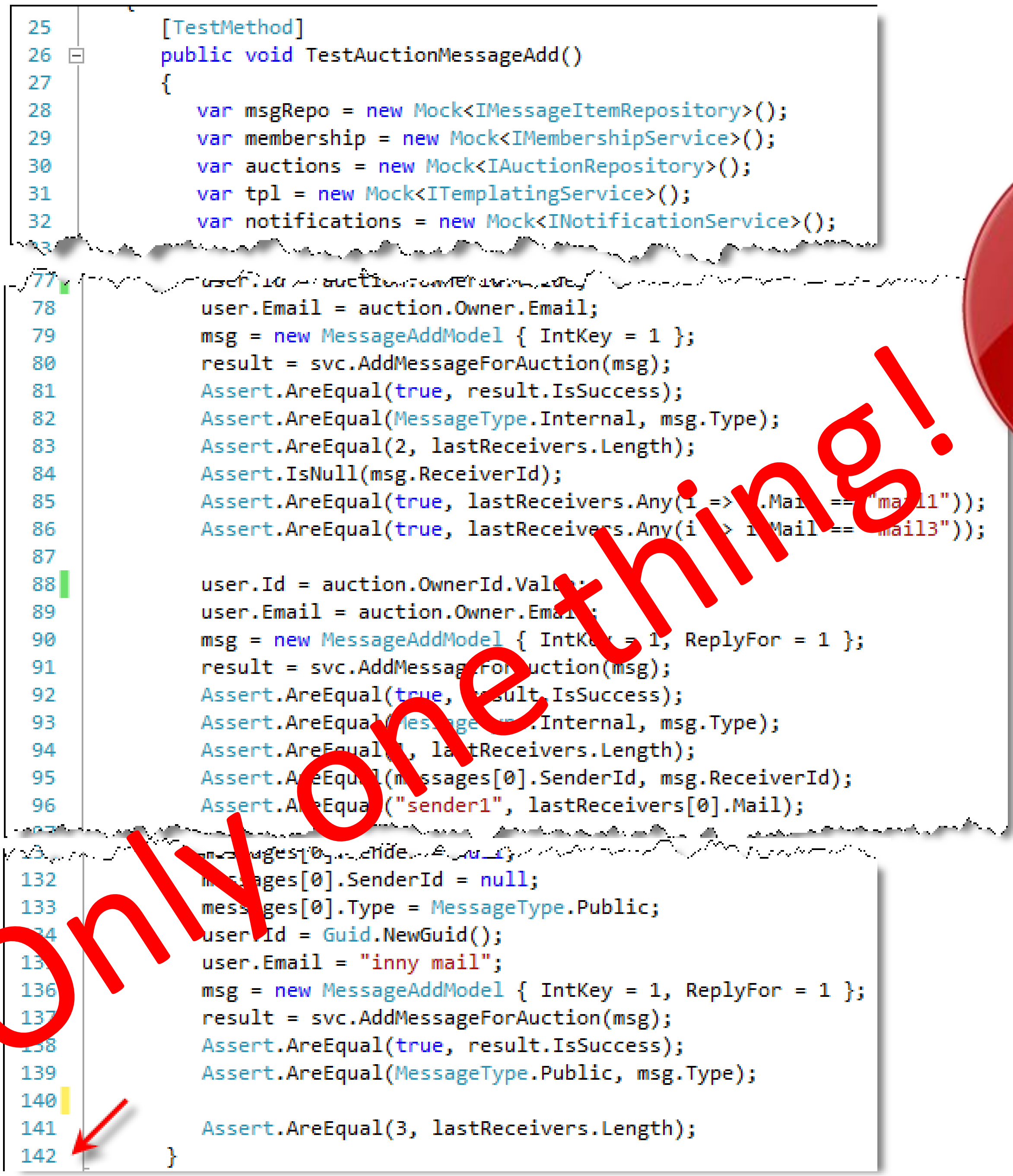

I always test **only one** thing!
Lay off!
It’s simple!



```
[TestMethod]
public void TestAuctionMessageAdd()
{
    var msgRepo = new Mock<IMessageItemRepository>();
    var membership = new Mock<IMembershipService>();
    var auctions = new Mock<IAuctionRepository>();
    var tpl = new Mock<ITemplatingService>();
    var notifications = new Mock<INotificationService>();
```

```
    user.Email = auction.Owner.Email;
    msg = new MessageAddModel { IntKey = 1 };
    result = svc.AddMessageForAuction(msg);
    Assert.AreEqual(true, result.IsSuccess);
    Assert.AreEqual(MessageType.Internal, msg.Type);
    Assert.AreEqual(2, lastReceivers.Length);
    Assert.IsNull(msg.ReceiverId);
    Assert.AreEqual(true, lastReceivers.Any(i => i.Mail == "mail1"));
    Assert.AreEqual(true, lastReceivers.Any(i => i.Mail == "mail3"));

    user.Id = auction.OwnerId.Value;
    user.Email = auction.Owner.Email;
    msg = new MessageAddModel { IntKey = 1, ReplyFor = 1 };
    result = svc.AddMessageForAuction(msg);
    Assert.AreEqual(true, result.IsSuccess);
    Assert.AreEqual(MessageType.Internal, msg.Type);
    Assert.AreEqual(1, lastReceivers.Length);
    Assert.AreEqual(messages[0].SenderId, msg.ReceiverId);
    Assert.AreEqual("sender1", lastReceivers[0].Mail);
```

```
    messages[0].SenderId = null;
    messages[0].Type = MessageType.Public;
    user.Id = Guid.NewGuid();
    user.Email = "inny mail";
    msg = new MessageAddModel { IntKey = 1, ReplyFor = 1 };
    result = svc.AddMessageForAuction(msg);
    Assert.AreEqual(true, result.IsSuccess);
    Assert.AreEqual(MessageType.Public, msg.Type);

    Assert.AreEqual(3, lastReceivers.Length);
}
```

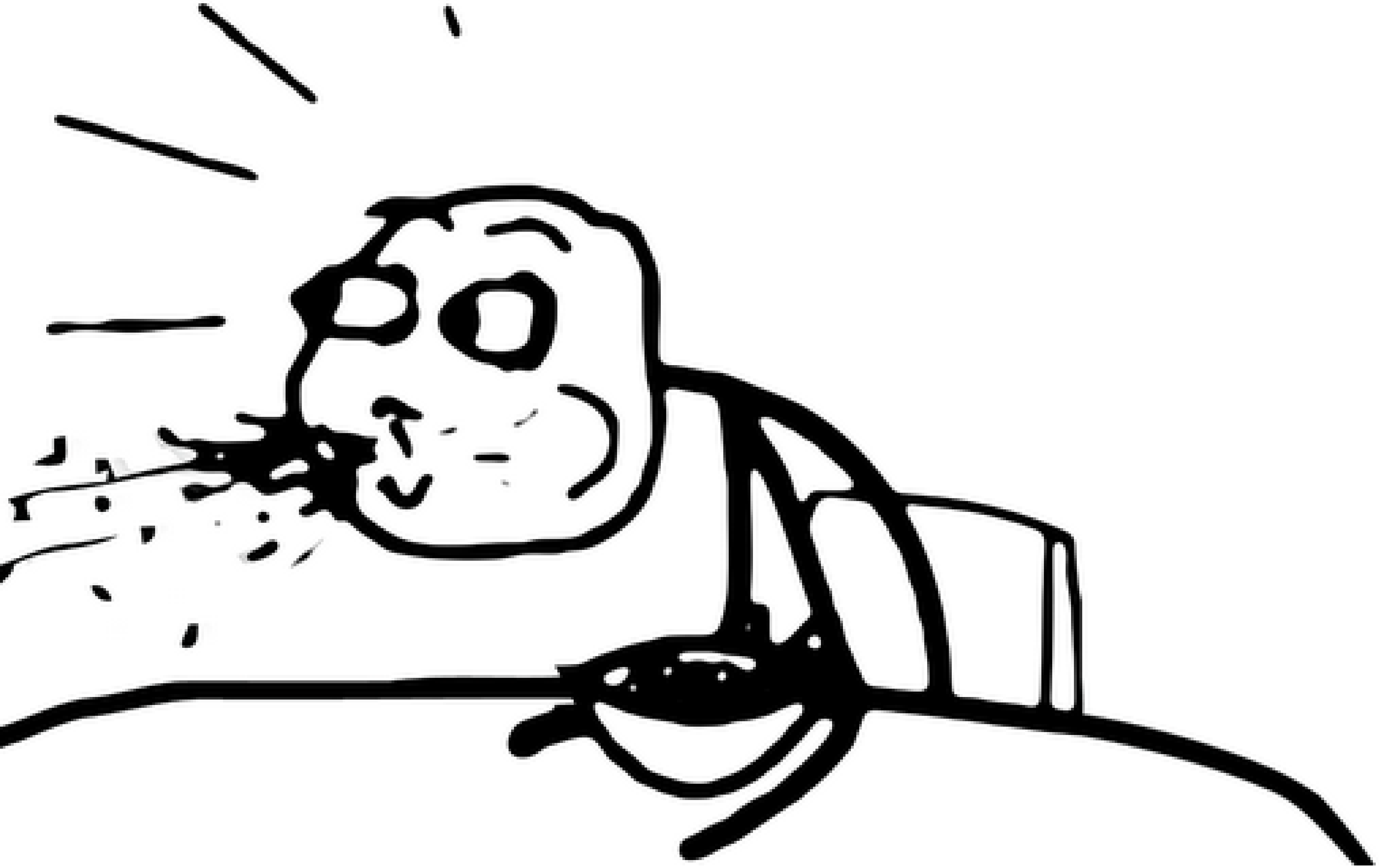


```
        [TestMethod]
        public void TestAuctionMessageAdd()
        {
            var msgRepo = new Mock<IMessageItemRepository>();
            var membership = new Mock<IMembershipService>();
            var auctions = new Mock<IAuctionRepository>();
            var tpl = new Mock<ITemplatingService>();
            var notifications = new Mock<INotificationService>();
```

```
            user.Id = auction.OwnerId.Value;
            user.Email = auction.Owner.Email;
            msg = new MessageAddModel { IntKey = 1 };
            result = svc.AddMessageForAuction(msg);
            Assert.AreEqual(true, result.IsSuccess);
            Assert.AreEqual(MessageType.Internal, msg.Type);
            Assert.AreEqual(2, lastReceivers.Length);
            Assert.IsNull(msg.ReceiverId);
            Assert.AreEqual(true, lastReceivers.Any(i => i.Mail == "mail1"));
            Assert.AreEqual(true, lastReceivers.Any(i => i.Mail == "mail3"));

            user.Id = auction.OwnerId.Value;
            user.Email = auction.Owner.Email;
            msg = new MessageAddModel { IntKey = 1, ReplyFor = 1 };
            result = svc.AddMessageForAuction(msg);
            Assert.AreEqual(true, result.IsSuccess);
            Assert.AreEqual(MessageType.Internal, msg.Type);
            Assert.AreEqual(1, lastReceivers.Length);
            Assert.AreEqual(messages[0].SenderId, msg.ReceiverId);
            Assert.AreEqual("sender1", lastReceivers[0].Mail);
```

```
            messages[0].SenderId = null;
            messages[0].Type = MessageType.Public;
            user.Id = Guid.NewGuid();
            user.Email = "inny mail";
            msg = new MessageAddModel { IntKey = 1, ReplyFor = 1 };
            result = svc.AddMessageForAuction(msg);
            Assert.AreEqual(true, result.IsSuccess);
            Assert.AreEqual(MessageType.Public, msg.Type);

            Assert.AreEqual(3, lastReceivers.Length);
        }
```

Only one thing!



Trustworthiness

Decide when to remove or change tests

- Test bugs.
- Semantics or API changes.
- Conflicting or invalid tests.
- Renaming or refactoring tests.
- Eliminating duplicate tests.

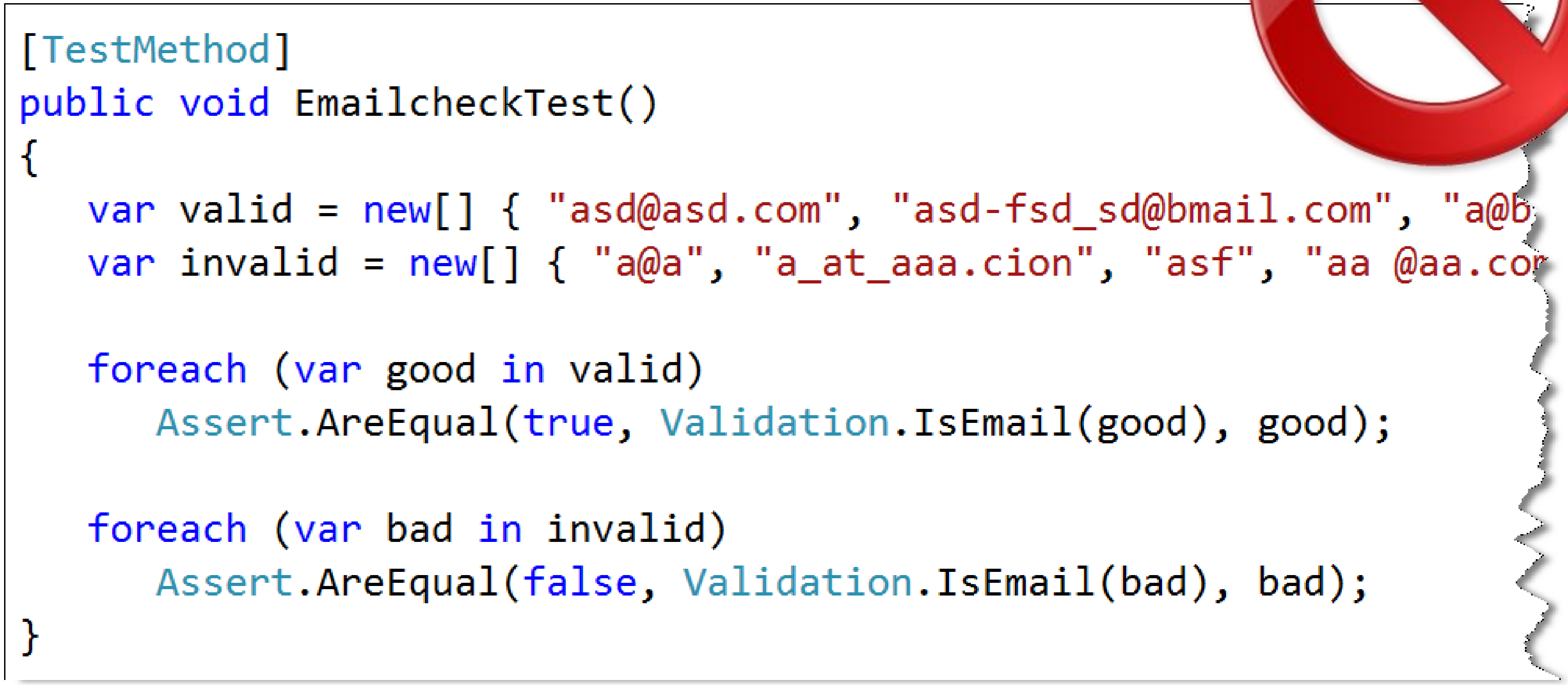
Trustworthiness

Avoid test logic

Your test should not contain:

- **switch**, **if**, or **else** statements,
- **foreach**, **for**, or **while** loops.

Avoiding foreach, switch…



```
[TestMethod]
public void EmailcheckTest()
{
   var valid = new[] { "asd@asd.com", "asd-fsd_sd@bmail.com", "a@b
   var invalid = new[] { "a@a", "a_at_aaa.cion", "asf", "aa @aa.co

   foreach (var good in valid)
      Assert.AreEqual(true, Validation.IsEmail(good), good);

   foreach (var bad in invalid)
      Assert.AreEqual(false, Validation.IsEmail(bad), bad);
}
```

How to do well?

http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

The solution

```
[TestCase("a@a")]
[TestCase("a_at_aaa.cion")]
[TestCase("asf")]
[TestCase("aa @aa.com")]
[TestCase("$aa@df.com")]
[TestCase("ąśżół@wp.pl")]
[TestCase(null)]
[TestCase("")]
public void IsEmail_WithInvalidEmails_ShouldBeFalse(string email)
{
    var isValid = Validation.IsEmail(email);

    Assert.That(isValid, Is.False);
}
```

The solution
(continued)

```
[TestCase("asd@asd.com")]
[TestCase("asd-fsd_sd@bmail.com")]
[TestCase("a@b.com")]
[TestCase("34534@23bv.pl")]
[TestCase("aa.aaa.aaa@aa.aaa.aa.pl")]
public void IsEmail_WithValidEmails_ShouldBeTrue(string email)
{
    var isValid = Validation.IsEmail(email);

    Assert.That(isValid, Is.True);
}
```

Trustworthiness

Testing only one thing

- If your test contains more than a single assert, it may be testing more than one thing ☺ …
- You should run additional asserts in separate, self-contained unit tests so that you can see what really fails.

The pillars of good tests

Maintainability

Nonmaintainable tests are nightmares.

Maintainability

- Testing private or protected methods.
- Removing duplication.
- Using setup methods in a maintainable manner.
- Enforcing test isolation.
- Avoiding overspecification in tests.

Maintainability

Testing private or protected methods

- Making methods public.
- Extracting methods to new classes.
- Making methods static.
- Making methods internal (C#).
  [InternalsVisibleTo("TestAssembly")].

Maintainability

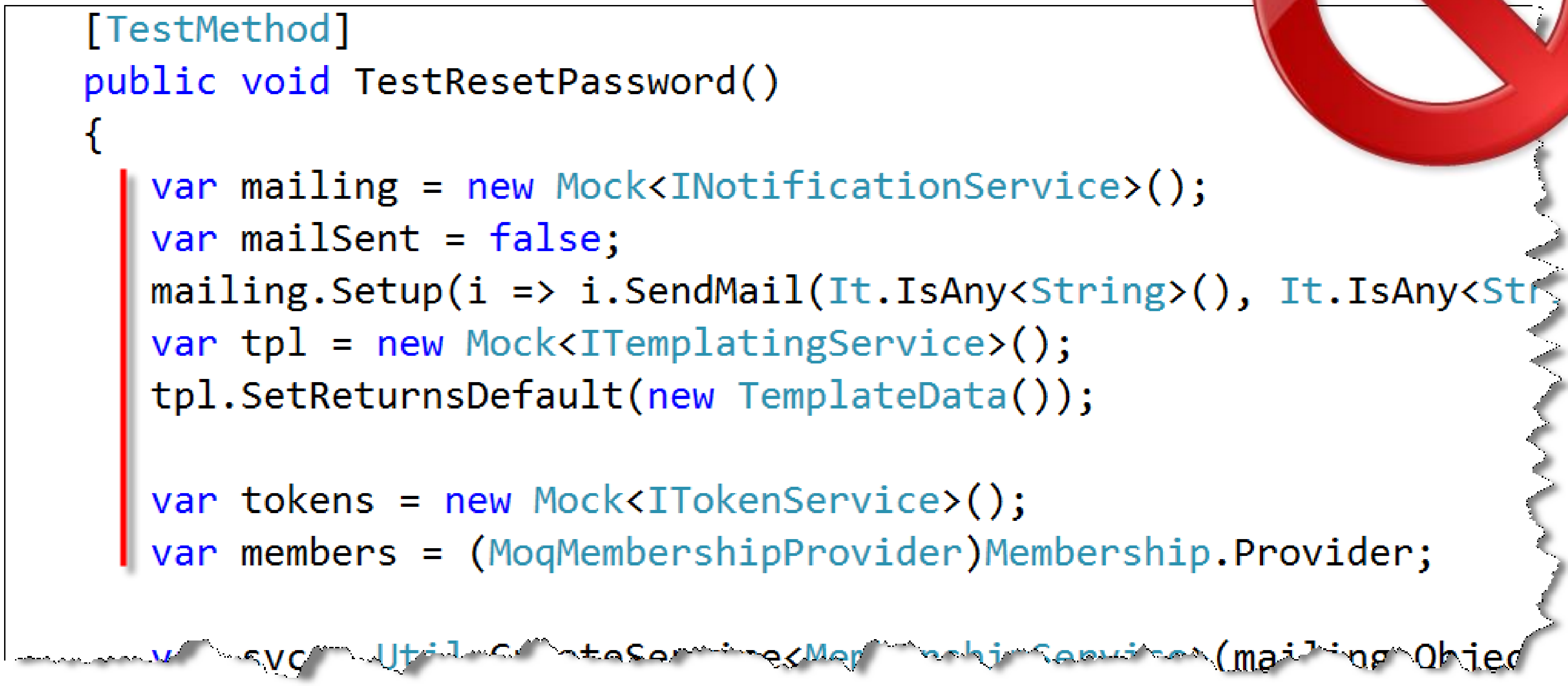
Removing duplication

- Removing duplication using a helper method.
- Removing duplication using [**SetUp**].

Duplicated Code

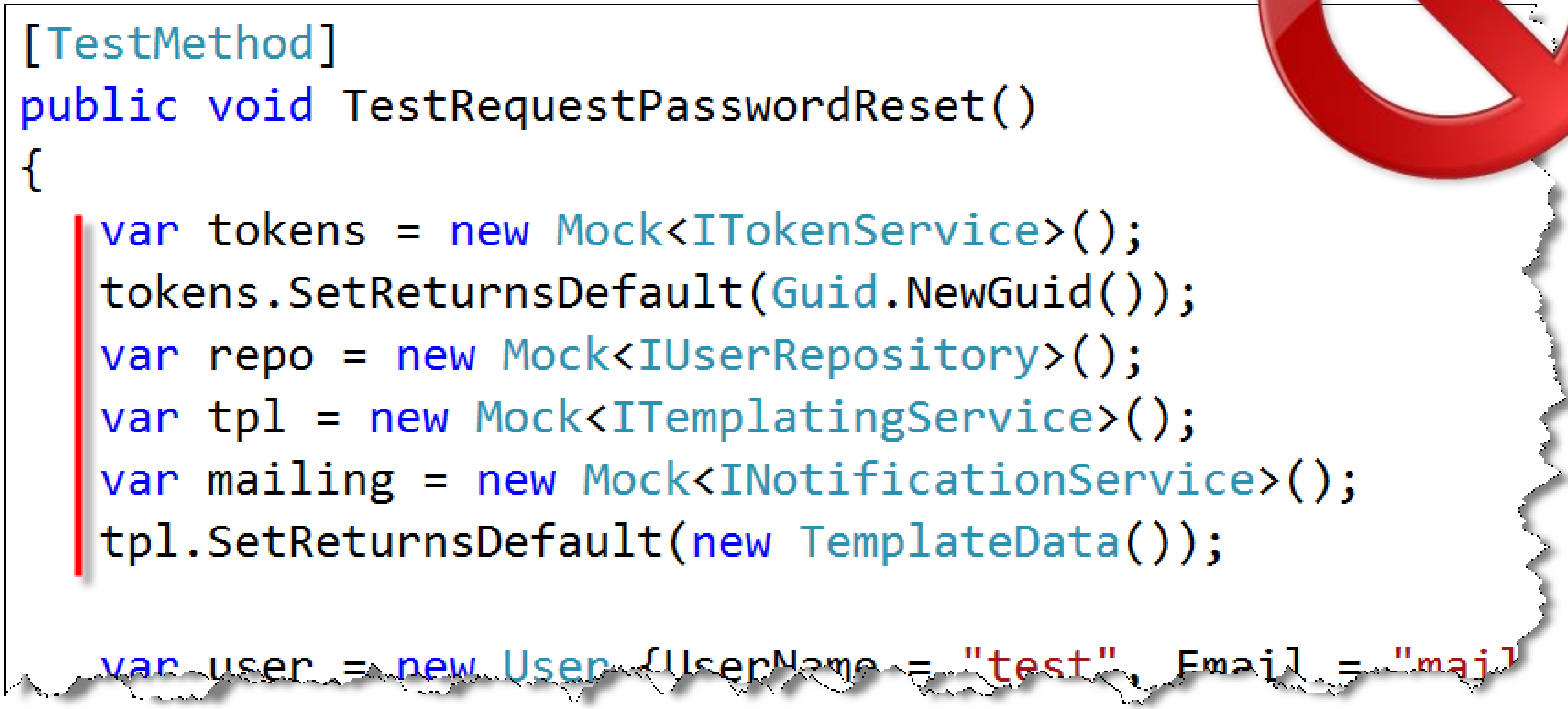
```
[TestClass]
public class MembershipServiceTests
{
    [TestMethod]
    public void TestResetPassword()...

    [TestMethod]
    public void TestRequestPasswordReset()...
}
```

Duplicated Code



```
[TestMethod]
public void TestResetPassword()
{
    var mailing = new Mock<INotificationService>();
    var mailSent = false;
    mailing.Setup(i => i.SendMail(It.IsAny<String>(), It.IsAny<Str
    var tpl = new Mock<ITemplatingService>();
    tpl.SetReturnsDefault(new TemplateData());

    var tokens = new Mock<ITokenService>();
    var members = (MoqMembershipProvider)Membership.Provider;

    var svc = Util.CreateService<MembershipService>(mailing.Objec
```

Duplicated Code



```
[TestMethod]
public void TestRequestPasswordReset()
{
    var tokens = new Mock<ITokenService>();
    tokens.SetReturnsDefault(Guid.NewGuid());
    var repo = new Mock<IUserRepository>();
    var tpl = new Mock<ITemplatingService>();
    var mailing = new Mock<INotificationService>();
    tpl.SetReturnsDefault(new TemplateData());

    var user = new User {UserName = "test", Email = "mail
```

How to do well?

http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

Removing duplication using **SetUp**

```
[TestInitialize]
public void SetUp()
{
   _mailing = new Mock<INotificationService>();
   var mailSent = false;
   _mailing.Setup(i => i.SendMail(
                         It.IsAny<String>(),
                         It.IsAny<String>(),
                         It.IsAny<MailReceiver[]>()))
            .Callback(() => mailSent = true);
   _template = new Mock<ITemplatingService>();
   _template.SetReturnsDefault(new TemplateData());
}
```

Be careful!

If you have **more than one mock** (not stub) probably your

test smell.

http://www.intellect-dt.ru/news/detail/283

Maintainability

Using setup methods in a maintainable manner

- Setup methods can only help when you need to initialize things.
- Setup methods aren’t always the best candidate for duplication removal.
- Setup methods can’t have parameters or return values.

Maintainability

Using setup methods in a maintainable manner

- Setup methods can’t be used as factory methods that return values.
- Setup methods should only contain code that applies to all the tests in the current test class, or the method will be harder to read and understand.

Maintainability

Using setup methods in a maintainable manner

- Initializing objects that are only used by some of the tests.
- Having setup code that’s long and hard to understand.
- Setting up mocks and fakes in the setup method.

Maintainability

Enforcing test isolation

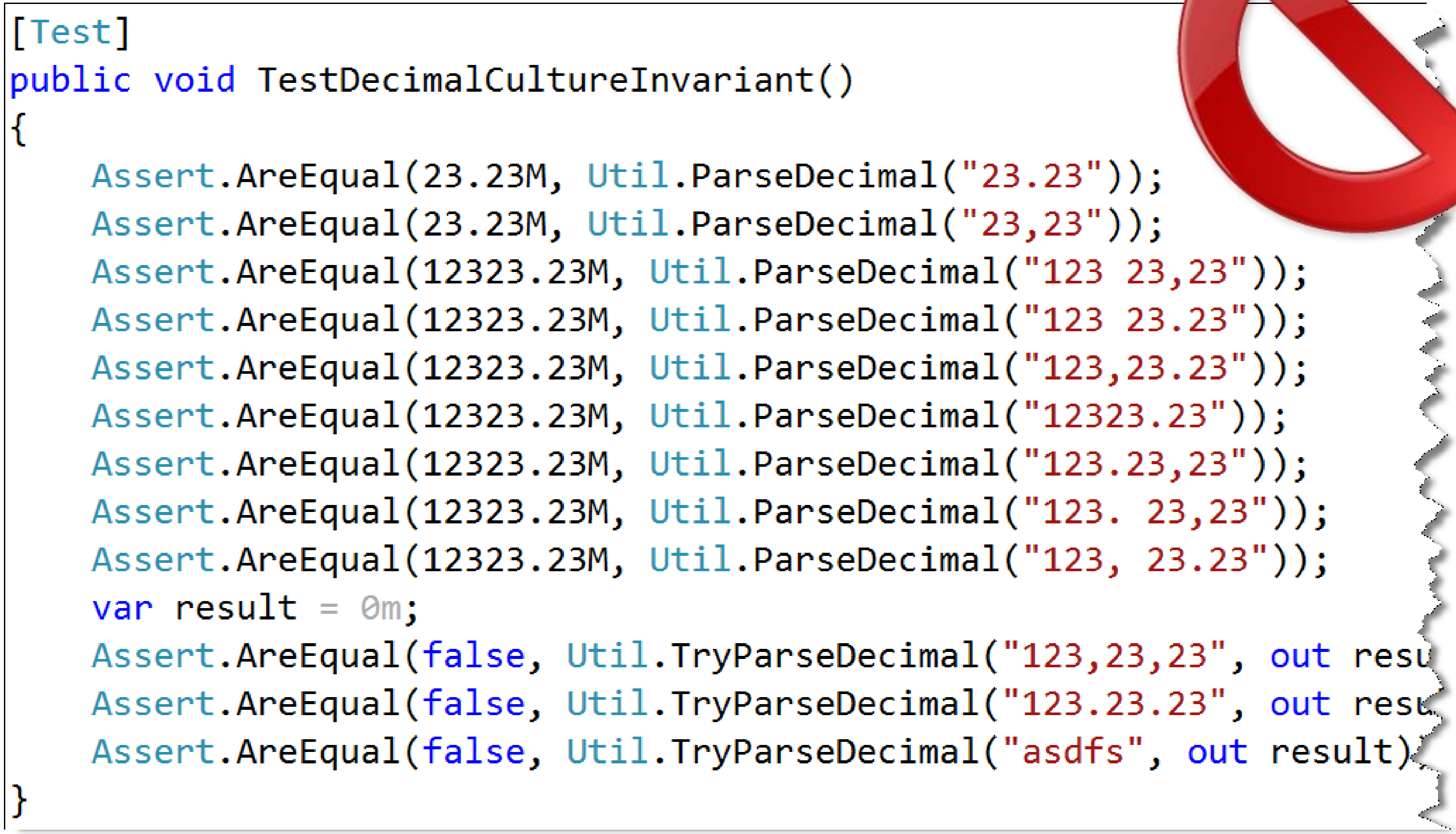
- Anti-pattern: constrained test order.
- Anti-pattern: hidden test call.
- Anti-pattern: shared-state corruption.
- Anti-pattern: external-shared-state corruption.

Maintainability

Avoiding multiple asserts

- Refactoring into multiple tests.
- Using parameterized tests.
- Wrapping with **try**-**catch**.

Avoiding multiple asserts



```
[Test]
public void TestDecimalCultureInvariant()
{
    Assert.AreEqual(23.23M, Util.ParseDecimal("23.23"));
    Assert.AreEqual(23.23M, Util.ParseDecimal("23,23"));
    Assert.AreEqual(12323.23M, Util.ParseDecimal("123 23,23"));
    Assert.AreEqual(12323.23M, Util.ParseDecimal("123 23.23"));
    Assert.AreEqual(12323.23M, Util.ParseDecimal("123,23.23"));
    Assert.AreEqual(12323.23M, Util.ParseDecimal("12323.23"));
    Assert.AreEqual(12323.23M, Util.ParseDecimal("123.23,23"));
    Assert.AreEqual(12323.23M, Util.ParseDecimal("123. 23,23"));
    Assert.AreEqual(12323.23M, Util.ParseDecimal("123, 23.23"));
    var result = 0m;
    Assert.AreEqual(false, Util.TryParseDecimal("123,23,23", out resu
    Assert.AreEqual(false, Util.TryParseDecimal("123.23.23", out resu
    Assert.AreEqual(false, Util.TryParseDecimal("asdfs", out result)
}
```

How to do well?

http://siriusbuzz.com/what-happens-with-that-sirius-lifetime-subscription.php/big-question-mark

The solution

```
[TestCase(23.23M, "23.23")]
[TestCase(23.23M, "23,23")]
[TestCase(12323.23M, "123 23,23")]
[TestCase(12323.23M, "123 23.23")]
[TestCase(12323.23M, "123,23.23")]
[TestCase(12323.23M, "12323.23")]
[TestCase(12323.23M, "123. 23,23")]
[TestCase(12323.23M, "123, 23.23")]
public void ParseDecimal_WithGoodValues_ShouldBeEqualToExpected(
    decimal expectedValue, string value)
{
    var actualValue = Util.ParseDecimal(value);

    Assert.That(expectedValue, Is.EqualTo(actualValue));
}
```

The solution

(continued)

```
[TestCase(23.23M, "23.23")]
[TestCase(23.23M, "23,23")]
[TestCase(12323.23M, "123 23,23")]
[TestCase(12323.23M, "123 23.23")]
[TestCase(12323.23M, "123,23.23")]
[TestCase(12323.23M, "12323.23")]
[TestCase(12323.23M, "123. 23,23")]
[TestCase(12323.23M, "123, 23.23")]
public void ParseDecimal_WithGoodValues_ShouldBeEqualToExpected(
    decimal expectedValue, string value)
{
    var actualValue = Util.ParseDecimal(value);

    Assert.That(expectedValue, Is.EqualTo(actualValue));
}
```

The solution
(continued)

```
[TestCase("123,23,23")]
[TestCase("123.23.23")]
[TestCase("asdfs")]
public void TryParseDecimal_WithBadValues_ShouldBeFalsea(string value)
{
    var result = 0m;

    var actualValue = Util.TryParseDecimal(value, out result);

    Assert.That(actualValue, Is.False);
}
```

The pillars of good tests

Readability

Readability is so important that, without it, the tests we write are almost meaningless.

What makes a clean test?

Three things:

- Readability.
- Readability.
- Readability.

Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin.*

Readability

- Naming unit tests.
- Naming variables.
- Creating good assert messages.
- Separating asserts from actions.

Readability

Naming unit tests

- The name of the method being tested.
- The scenario under which it’s being tested.

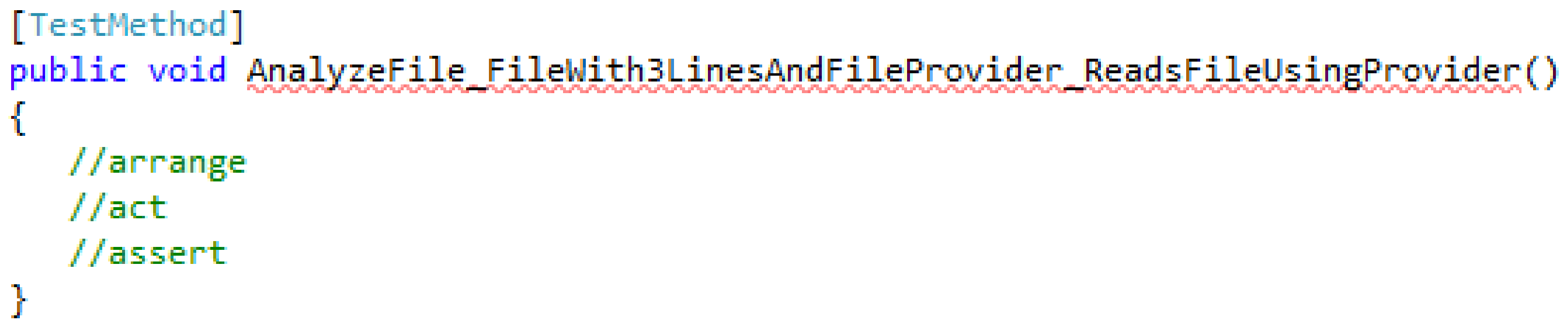
  When I call method X **with a null value**, then it should do Y.
- The expected behavior when the scenario is invoked.

  When I call method X with a null value, then it **should do Y**.

  MethodUnderTest_Scenario_Behavior()

Readability

Naming unit tests

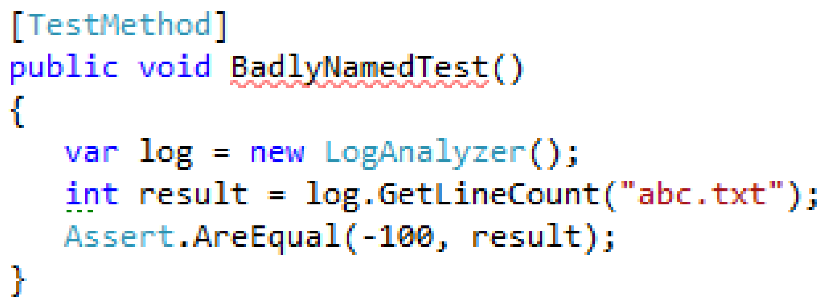


```
[TestMethod]
public void AnalyzeFile_FileWith3LinesAndFileProvider_ReadsFileUsingProvider()
{
   //arrange
   //act
   //assert
}
```

Readability

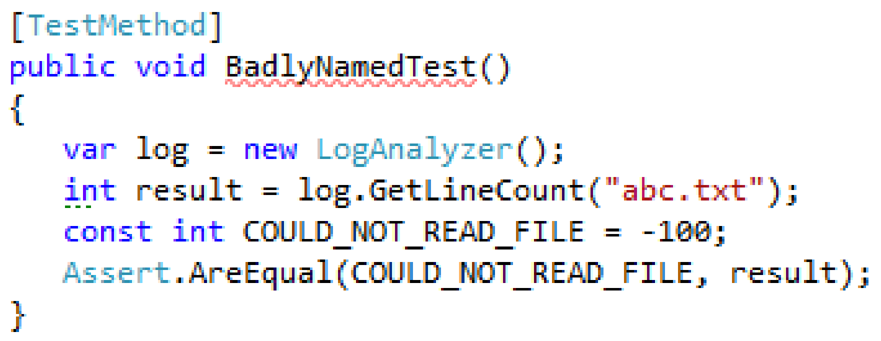
Naming variables

Bad



```
[TestMethod]
public void BadlyNamedTest()
{
    var log = new LogAnalyzer();
    int result = log.GetLineCount("abc.txt");
    Assert.AreEqual(-100, result);
}
```

Better



```
[TestMethod]
public void BadlyNamedTest()
{
    var log = new LogAnalyzer();
    int result = log.GetLineCount("abc.txt");
    const int COULD_NOT_READ_FILE = -100;
    Assert.AreEqual(COULD_NOT_READ_FILE, result);
}
```

The Art of Unit Testing: With Examples in .Net by *Roy Osherove*.

Readability

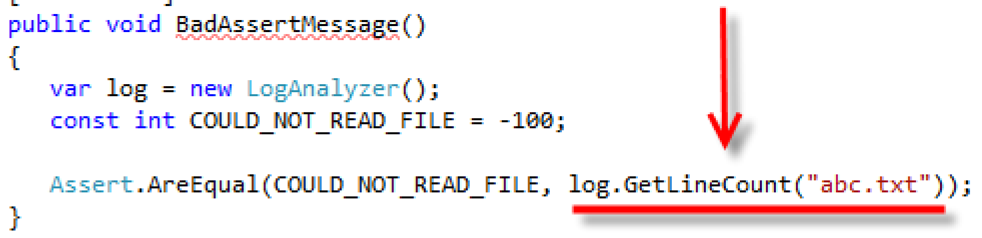
Asserting yourself with meaning

- Don’t repeat what the built-in test framework outputs to the console.
- Don’t repeat what the test name explains.
- If you don’t have anything good to say, don’t say anything.
- Write what should have happened or what failed to happen, and possibly mention when it should have happened.

Readability

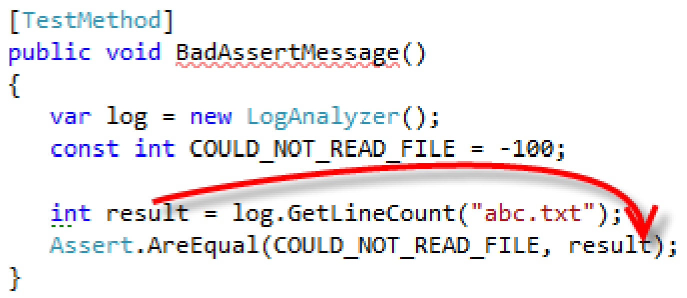
Separating asserts from actions

Bad



```
[TestMethod]
public void BadAssertMessage()
{
    var log = new LogAnalyzer();
    const int COULD_NOT_READ_FILE = -100;

    Assert.AreEqual(COULD_NOT_READ_FILE, log.GetLineCount("abc.txt"));
}
```

Better



```
[TestMethod]
public void BadAssertMessage()
{
    var log = new LogAnalyzer();
    const int COULD_NOT_READ_FILE = -100;

    int result = log.GetLineCount("abc.txt");
    Assert.AreEqual(COULD_NOT_READ_FILE, result);
}
```

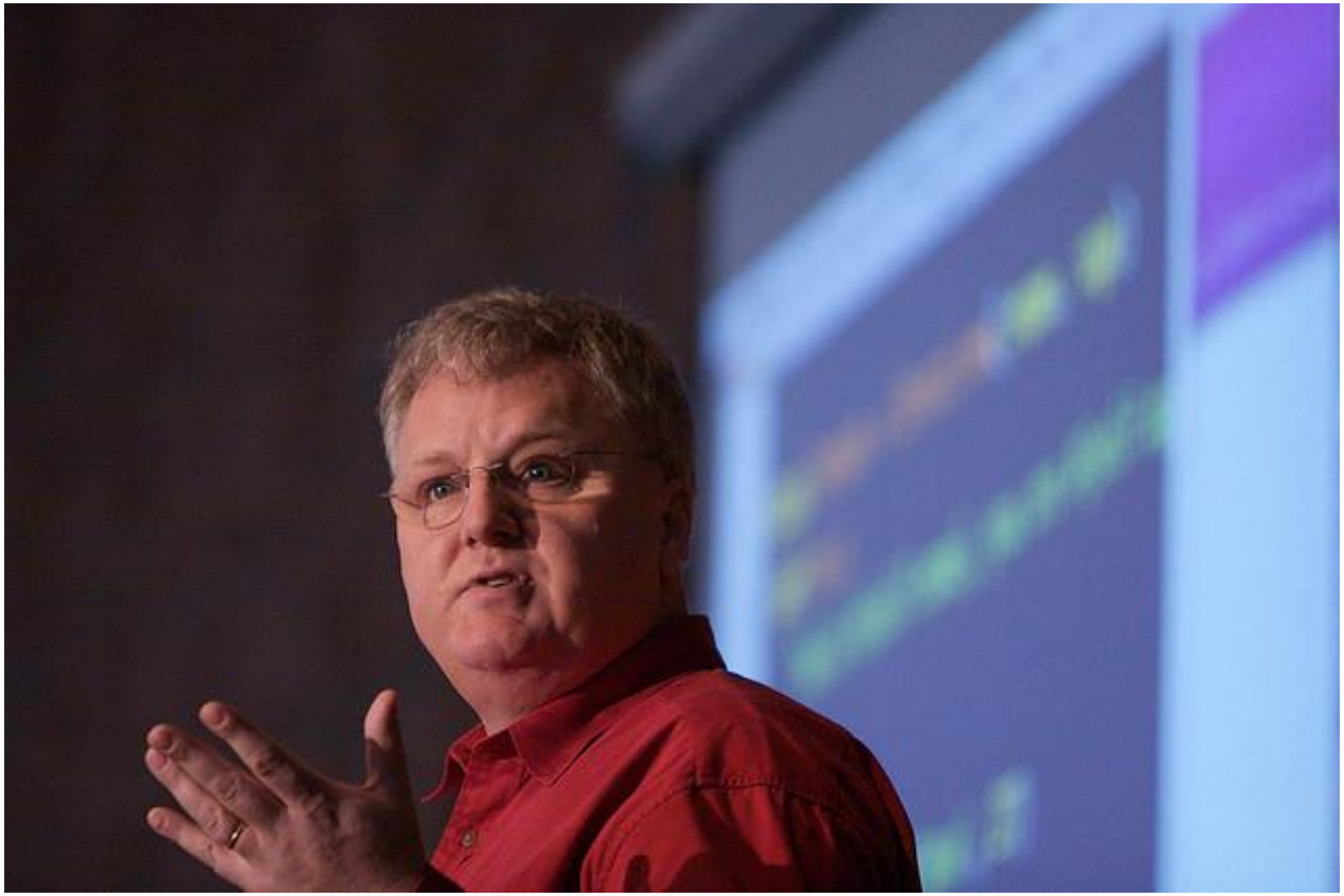
The third task

http://internationalbudget.org/blog/2009/06/11/cash-or-work-what-do-people-want/

Andy Hunt

http://www.toolshed.com/

Dave Thomas



http://www.flickr.com/photos/fraserspeirs/3391852097/

What to test: The Right-BICEP

- Right
- Boundary
- Inverse
- Cross-check
- Error conditions
- Performance characteristics

The Right-BICEP

Are the results **right**?

The Right-BICEP

Are all the **boundary** conditions **CORRECT?**

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

The Right-BICEP
CORRECT

- **C**onformance
- **O**rdering
- **R**ange
- **R**eference
- **E**xistence
- **C**ardinality
- **T**ime (absolute and relative)

The Right-BICEP (boundary conditions) CORRECT

Conformance

Does the value conform to an expected format?

The Right-BICEP

CORRECT

Ordering

Is the set of values ordered or unordered as appropriate?

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

The Right-BICEP
CORRECT

Range

Is the value within reasonable minimum and maximum values?

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

The Right-BICEP
CORRECT

Reference

Does the code reference anything external that isn't under direct control of the code itself?

The Right-BICEP CORRECT

Existence

Does the value exist (e.g., is non-null, nonzero, present in a set, etc.)?

The Right-BICEP
CORRECT

Cardinality

Are there exactly enough values?

The Right-BICEP CORRECT

Time (absolute and relative)

Is everything happening in order? At the right time? In time?

The Right-BICEP

Can you check **inverse** relationships?

The Right-BICEP

Can you **cross-check** results using other means?

The Right-BICEP

Can you force **error conditions** to happen?

The Right-BICEP

Are **performance** characteristics within bounds?

The fourth task

http://internationalbudget.org/blog/2009/06/11/cash-or-work-what-do-people-want/

Double objects

MockObjects

Tim Mackinnon, Steve Freeman presented a paper at XP2000 on the topic of MockObjects

Tim Mackinnon

Steve Freeman

A mock object

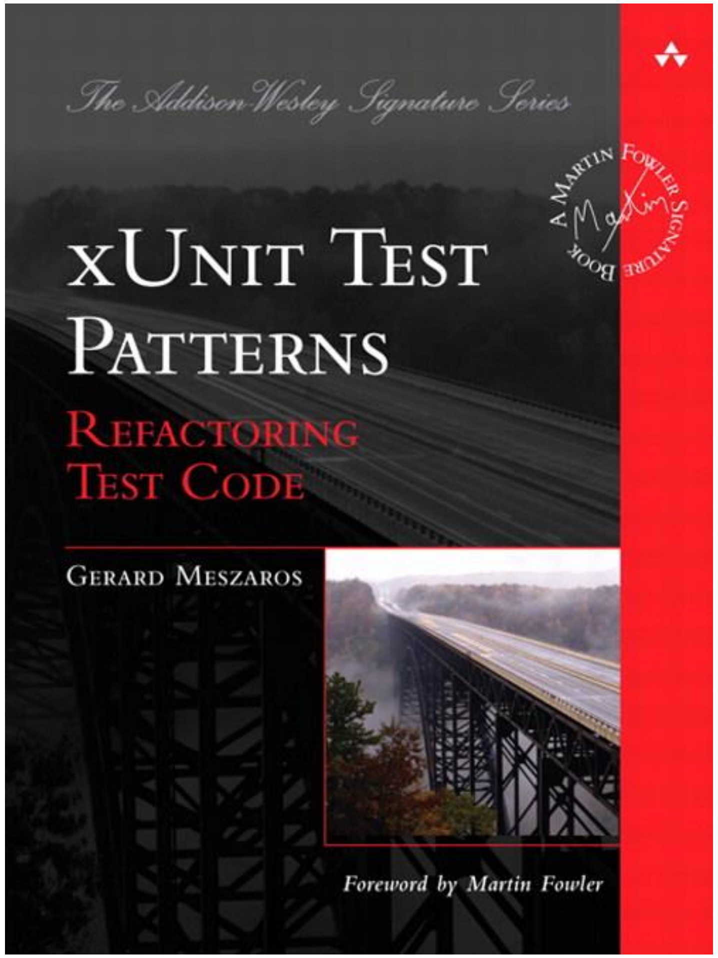
- is easily created
- is easily set up
- is quick
- is deterministic
- has easily caused behaviour
- has no direct user interface
- is directly queriable

The Art of Unit Testing: With Examples in .Net by *Roy Osherove*.

Gerard Meszaros

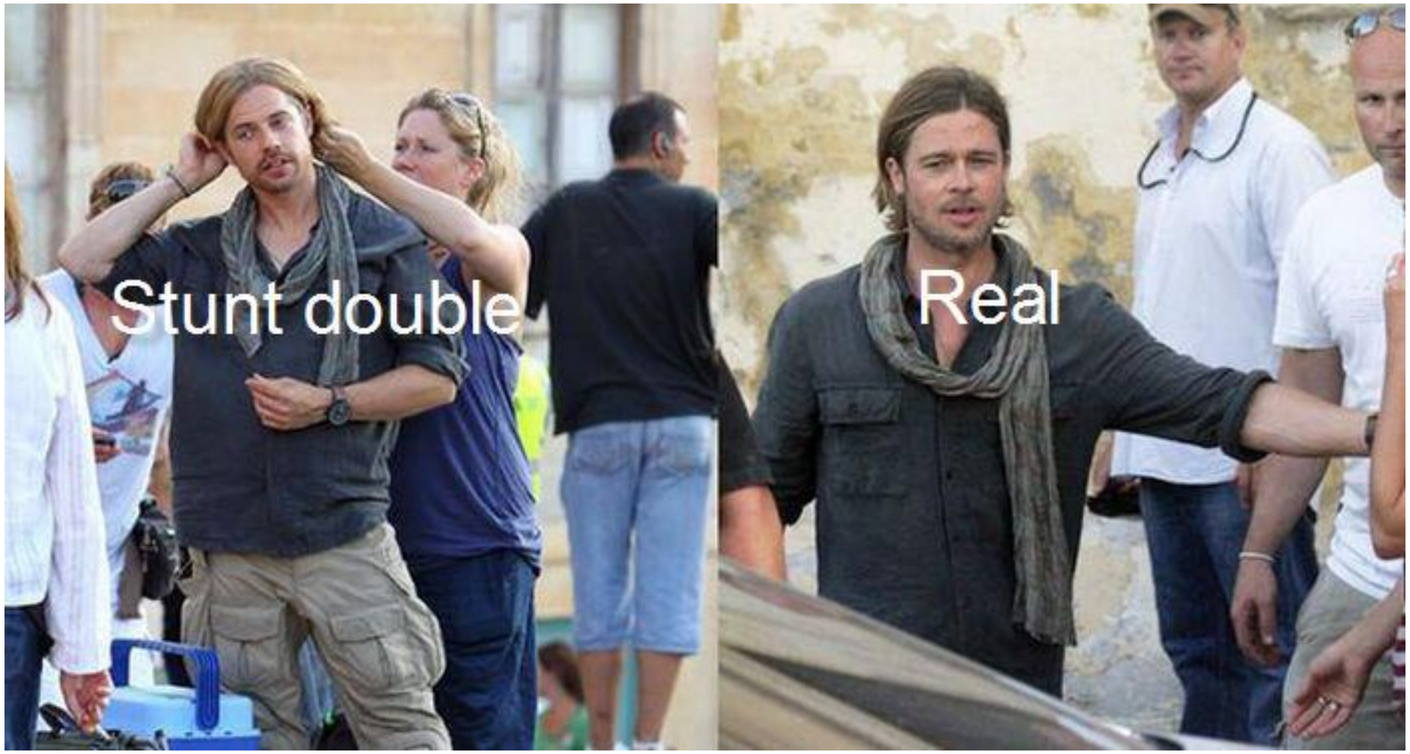
http://tech.qq.com/a/20121025/000197.htm

xUnit Test Patterns: Refactoring Test Code


The Addison Wesley Signature Series
A Martin Fowler Signature Book
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler

http://martinfowler.com/books/meszaros.html

Stunt double

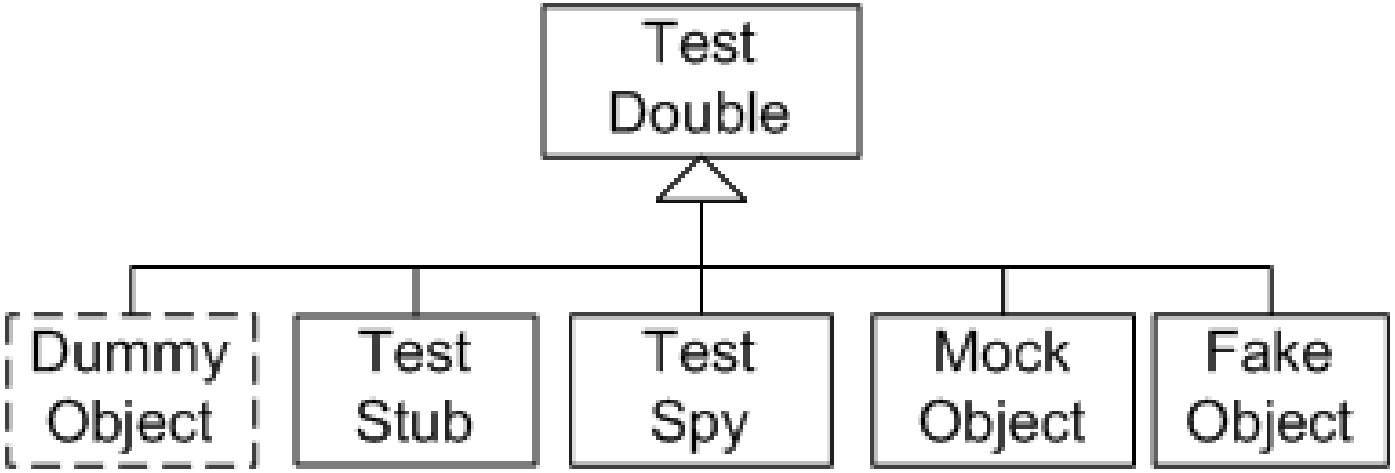


http://zombiecreep.blogspot.com/2012/05/celebrities-and-their-stunt-doubles.html

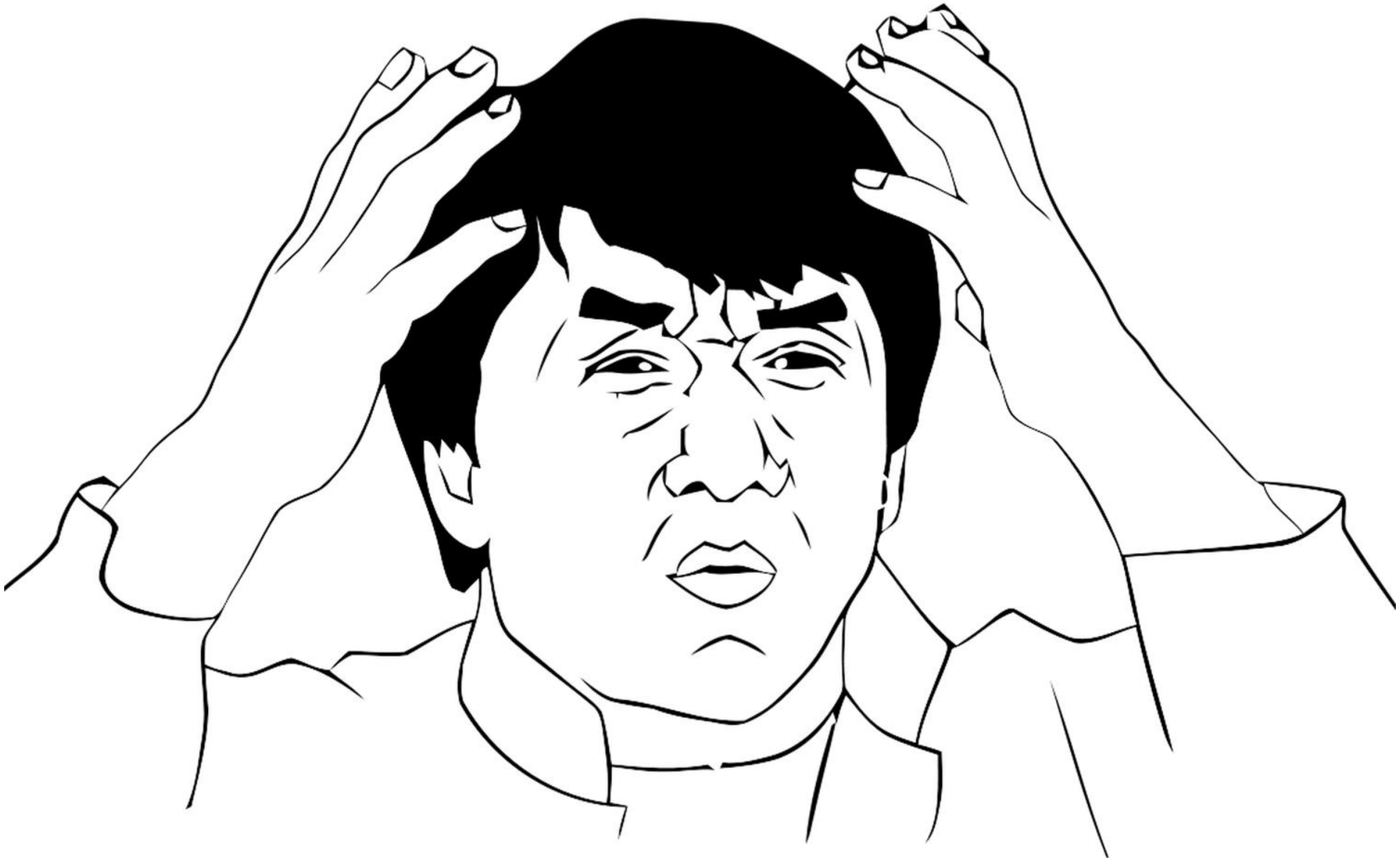
xUnit Test Patterns: Refactoring Test Code

- Dummy object
- Test stub
- Test spy
- Mock object
- Fake object

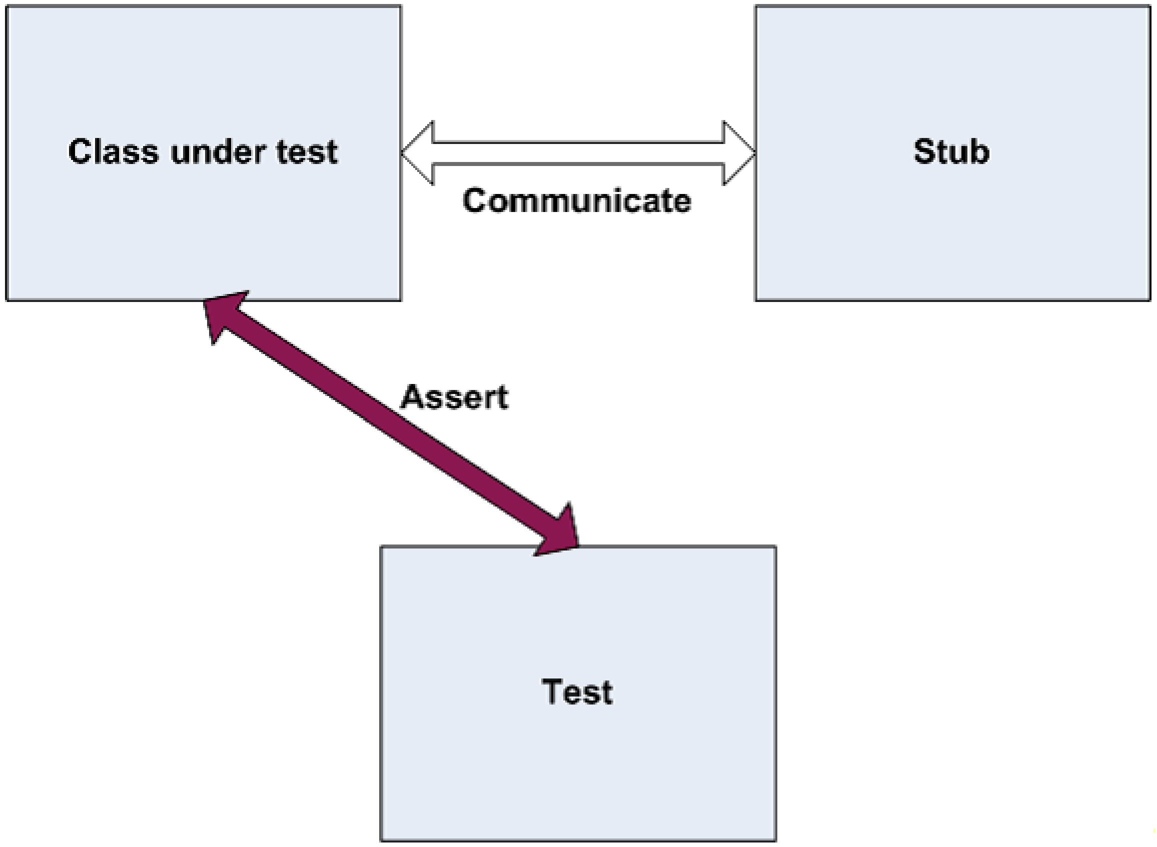
Test double



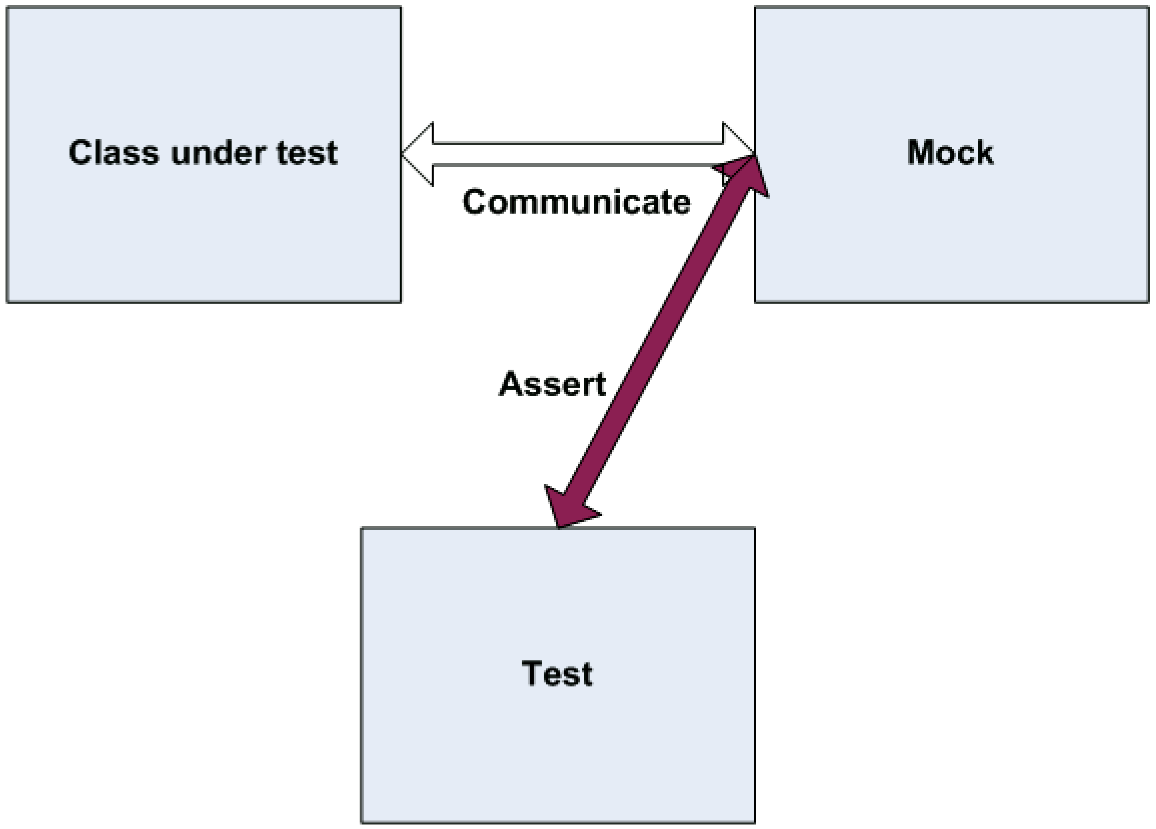
xUnit Test Patterns by *Gerard Meszaros*.



Stub



Mock



Moq

moq

test first. mock me later.

https://github.com/Moq

In Moq library all double objects are Moq’s.

QuickStart - moq

```
// WOW! No record/replay weirdness?! :)
mock.Setup(framework => framework.DownloadExists("2.0.0.0"))
    .Returns(true)
    .AtMostOnce();

// Hand mock.Object as a collaborator and exercise it,
// like calling methods on it...
ILoveThisFramework lovable = mock.Object;
bool download = lovable.DownloadExists("2.0.0.0");

// Verify that the given method was indeed called with the expected value
mock.Verify(framework => framework.DownloadExists("2.0.0.0"));
```

http://internationalbudget.org/blog/2009/06/11/cash-or-work-what-do-people-want/

The fifth task

http://internationalbudget.org/blog/2009/06/11/cash-or-work-what-do-people-want/

Properties of **good** unit tests

Good tests have the following properties, which makes them **A-TRIP**.

A-TRIP

- **A**utomatic
- **T**horough
- **R**epeatable
- **I**ndependent
- **P**rofessional

A-TRIP

Unit tests need to be run **automatically.**

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

A-TRIP

Good unit tests are **thorough**.

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

A-T**R**IP

Good unit tests are **repeatable.**

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

A-TRIP

Tests need to be **independent** from the environment and each other.

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

A-TRIP

Professional.

The code you write for a unit test is real.

Pragmatic Unit Testing in (C#/Java) by *Andrew Hunt, David Thomas*

Robert Cecil Martin aka "Uncle Bob"

http://www.butunclebob.com/CleanCoder.UncleBob

F.I.R.S.T.

- **F**ast
- **I**ndependent
- **R**epeatable
- **S**elf-Validating
- **T**imely

Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin.*

F.I.R.S.T.

Fast

Tests should be fast.

Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin.*

F.I.R.S.T.

Independent

Tests should not depend on each other.

Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin.*

F.I.R.S.T.

Repeatable

Tests should be repeatable in any environment.

Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin.*

F.I.R.S.T.

Self-Validating

The tests should have a boolean output.

F.I.R.S.T.

Timely

The tests need to be written in a timely fashion.

Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin.*

The sixth task

http://internationalbudget.org/blog/2009/06/11/cash-or-work-what-do-people-want/

Anti-patterns

Unit tests - anti-patterns

- The Liar
- Excessive Setup
- Giant
- The Mockery
- The Inspector
- Generous Leftovers
- The Local Hero

http://blog.james-carr.org/2006/11/03/tdd-anti-patterns

Unit tests - anti-patterns *(continued)*

- The Nitpicker
- The Secret Catcher
- The Dodger
- The Loudmouth
- The Greedy Catcher
- The Sequencer
- Hidden Dependency

http://blog.james-carr.org/2006/11/03/tdd-anti-patterns

Unit tests - anti-patterns *(continued)*

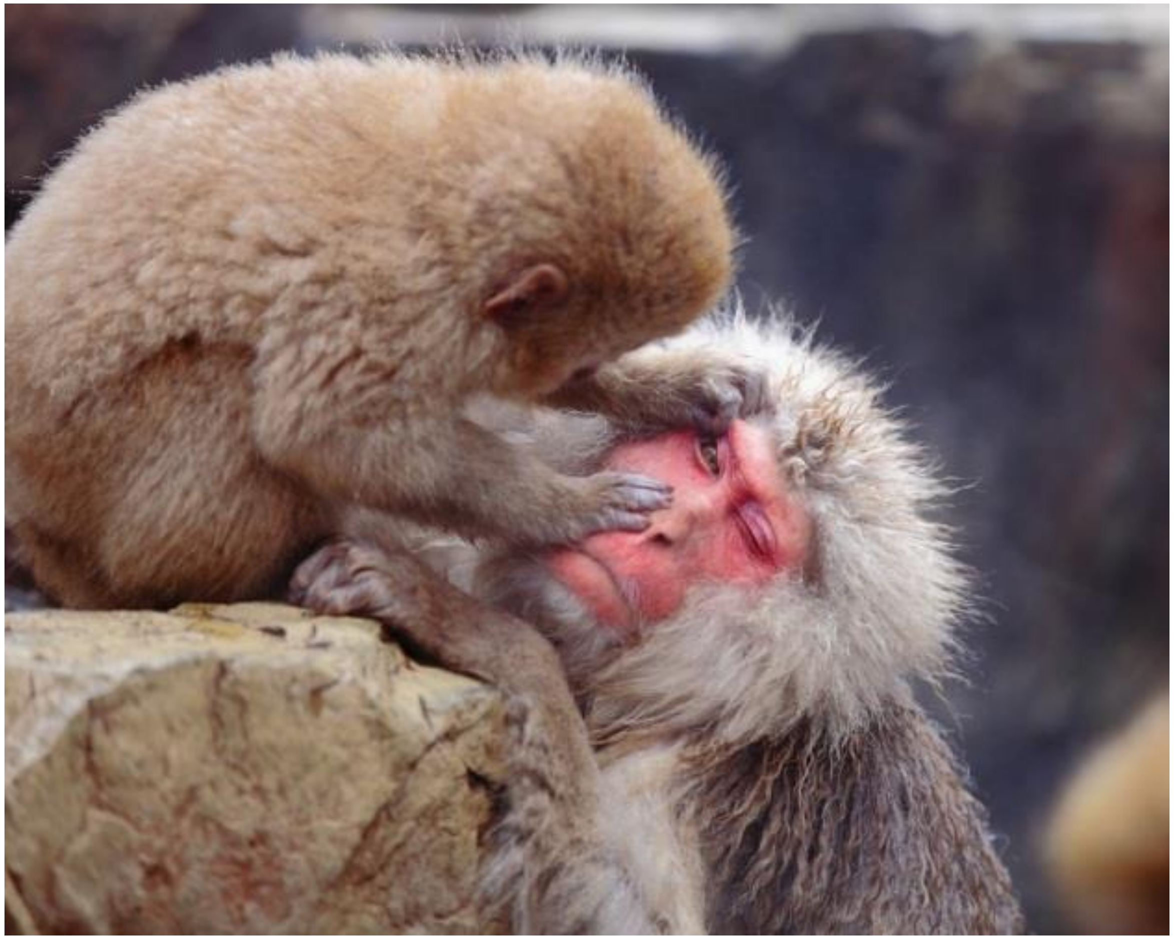
- The Enumerator
- The Stranger
- The Operating System Evangelist
- Success Against All Odds
- The Free Ride
- The One
- The Peeping Tom
- The Slow Poke

http://blog.james-carr.org/2006/11/03/tdd-anti-patterns

The seventh task

http://internationalbudget.org/blog/2009/06/11/cash-or-work-what-do-people-want/

It's not over yet



Testing the **tests**

There are two things you can do to help ensure that the test code is correct:

- improve tests when fixing bugs,
- prove tests by introducing bugs.

General principles

- Test anything that might break.
- Test everything that does break.
- New code is guilty until proven Innocent.
- Write at least as much test code as production code.
- Run local tests with each compile.
- Run all tests before check-in to repository.

Questions to ask

- If the code ran correctly, how would I know?
- How am I going to test this?
- What else can go wrong?
- Could this same kind of problem happen anywhere else?

In summary

- Unit tests make our **code robust**.
- Unit tests give us an enormous amount of **confidence** in our code.
- Unit tests serve as solid and reliable **documentation** and illustration as to how our code can be used.
- Writing good test code is **hard**, and maintaining obtuse test code is even harder!

Books

- **The Art of Unit Testing**: **With Examples in .Net** by *Roy Osherove*.
- **Pragmatic Unit Testing in C# with NUnit, 2nd Edition** by *Andy Hunt, Dave Thomas, Matt Hargett*.
- **Pragmatic Unit Testing in Java with JUnit** by *Andy Hunt, Dave Thomas*.
- **xUnit Test Patterns** by *Gerard Meszaros*.
- **Working Effectively with Legacy Code** by *Michael Feathers.*
- **Clean Code**: A Handbook of Agile Software Craftsmanship *Robert C. Martin.*
- **Unit Test Frameworks** by *Paul Hamill*.

So... enough for today