UNIT TESTING WITH JUNIT

Software Engineering Class

Prof. Adriano Peron

May, 28th 2013

Valerio Maggio, Ph.D. Candidate
valerio.maggio@unina.it

BRIEF INTRO TO THE CLASS

It's about Unit Testing + It's about JUnit

It's about
Unit Testing with JUnit

You say?!

DISCLAIMER

This is not a **Tutorial Class**

- At the end of the class you (should)...

1. ..have learnt something more about unit testing;

2. ..have learnt **what** is JUnit, **how** to use it and **when**;

3. ..have realized how much important are *testing activities*!

(maybe you *already noticed* that slides are in English...)

JUNIT PRELIMINARIES

- **Q**: How many "types" of testing do you know?
  - **A**: System Testing, Integration Testing, Unit Testing....

- **Q**: How many "testing techniques" do you know?
  - **A**: Black Box and White Box Testing

    Which is the difference?

- **Q**: What type and technique do you think JUnit covers?

JUNIT WORDS CLOUD

a.k.a. *some random words (almost) related to JUnit*

Testing

xUnit

Java

Testing framework

Unit Testing

Test Suite

Black Box Testing

Testing Automation

Test Fixtures

Simple Test Program

Test Runners

JUNIT JUMPSTART

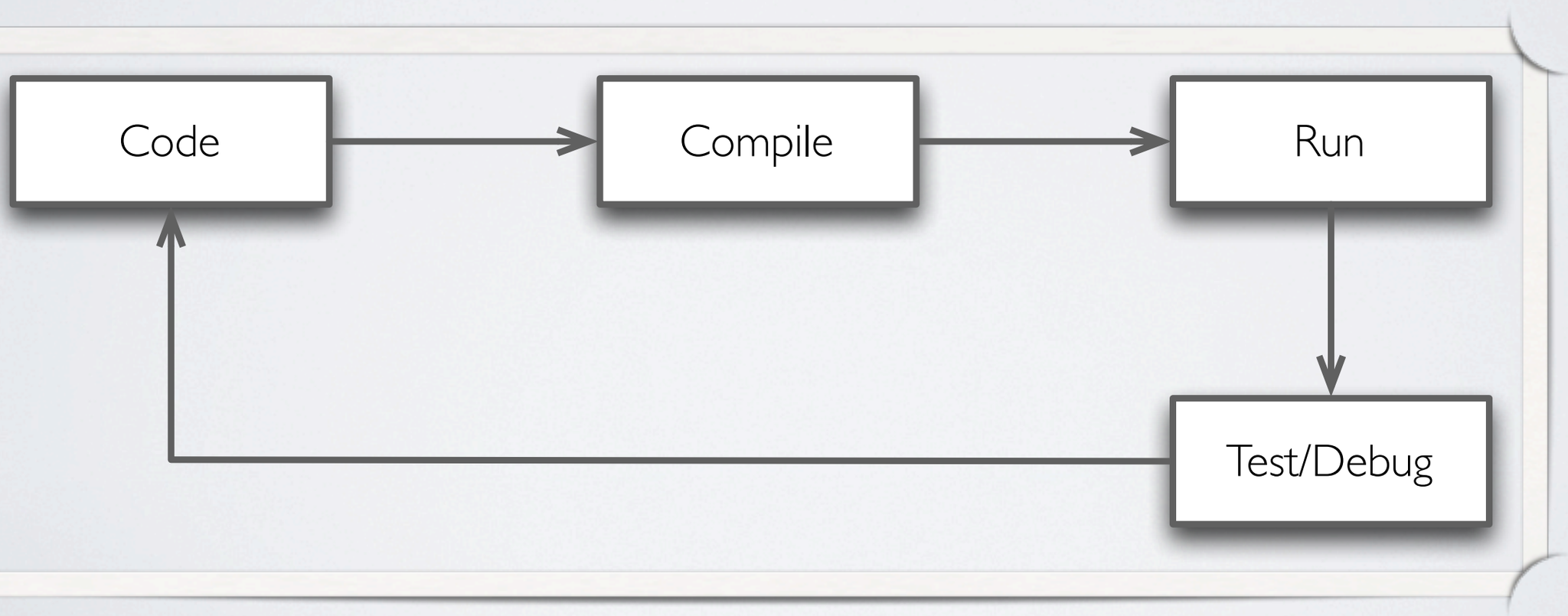
THE IMPORTANCE OF TESTING

> Never in the field of software development was so much owed by so many to so few lines of code

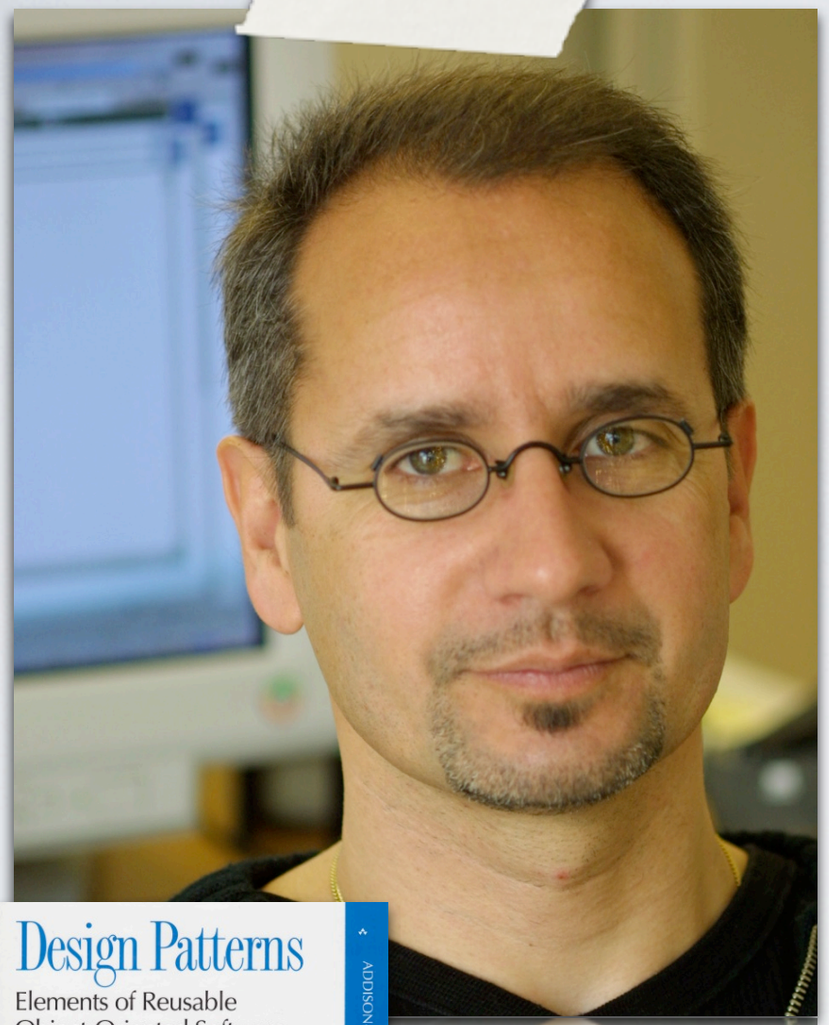
Martin Fowler

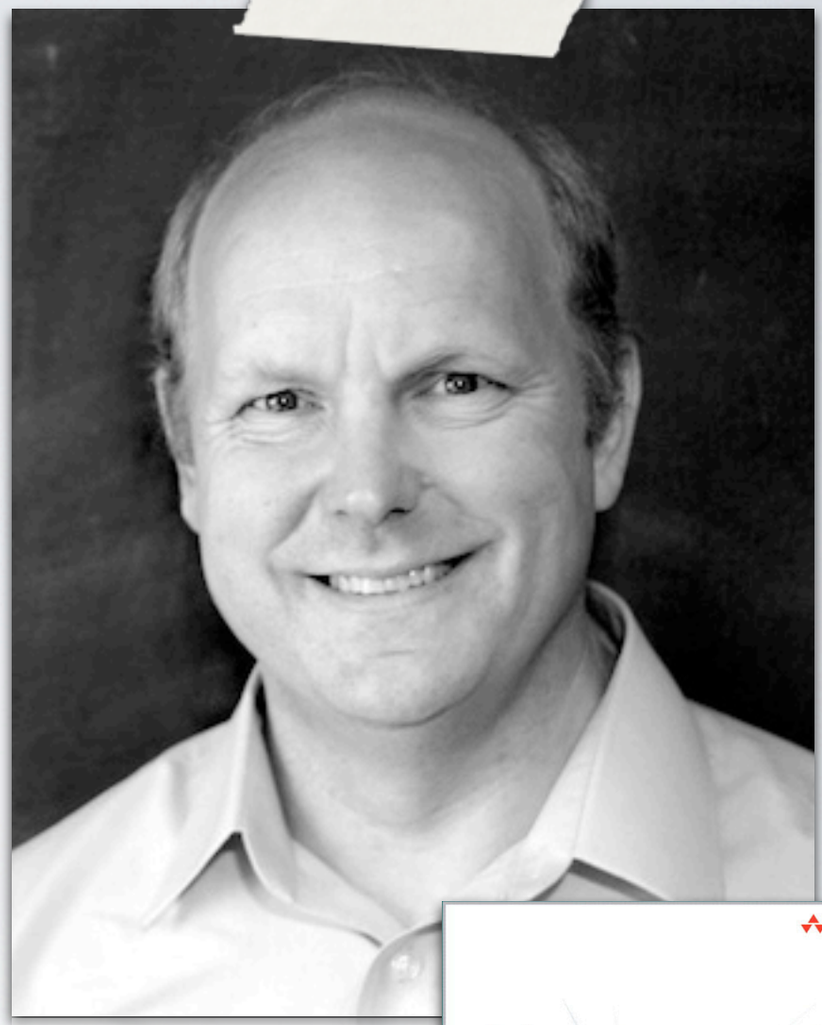
THE IMPORTANCE OF TESTING

- During development, the first thing we do is run our own program
  - This is *(sometimes)* called *Acceptance Testing*

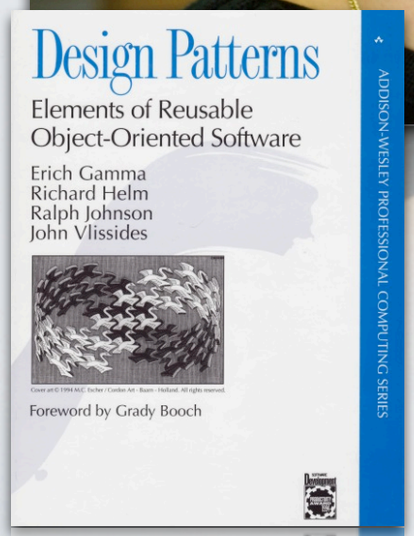


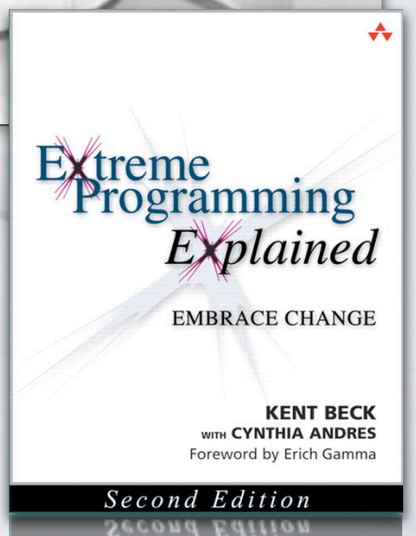
WHO ARE THESE TWO GUYS?





Creators of the **xUnit** Framework





Erich Gamma

Kent Beck

XUNIT FRAMEWORK

- *A framework is a semi-complete application that provides a reusable, common structure that can be shared between applications.*

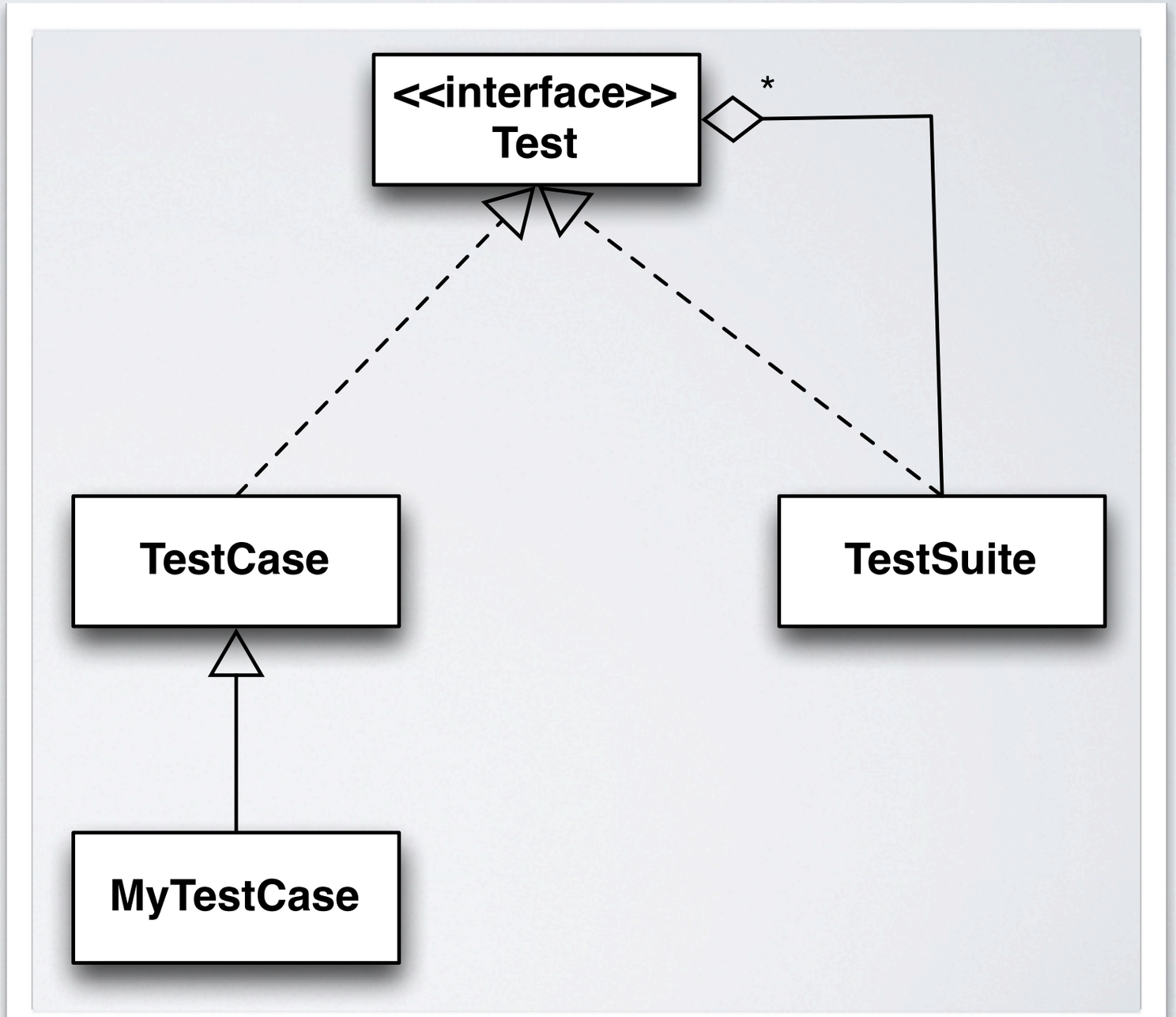
- *Developers incorporate the framework in their own application an extend it to meet their specific needs.*

- **Unit Test**: A unit test examines the behavior of a distinct *unit of work*.
  - The "distinct unit of work" is often (but not always) a single method.

XUNIT DESIGN

JUnit 3.x design was compliant with xUnit framework guidelines

- JUnit
- CppUnit
- PyUnit
- NUnit
- XMLUnit
- PHPUnit
- RUnit
- SUnit
- .....



WHY A FRAMEWORK IS NEEDED?

Let's do a very dummy example...

```
public class Calculator {

    public double add(double number1, double number2) {
        return number1 + number2;
    }
}
```

Q: How would you test this method?

VERY SIMPLE TESTING STRATEGY

```
public class TestCalculator {

    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        double result = calculator.add(10,50);
        if (result != 60){
            System.out.println("Bad result: " + result);
        }// end if
    }//end main
}
```

Q: How would you improve it?

IMPROVED (NAIVE) SOLUTION

```
public class TestCalculator {
    private int nbErrors = 0;

    public void testAdd() {
        Calculator calculator = new Calculator();
        b double result = calculator.add(10, 50);
        if (result != 60) {
            throw new RuntimeException("Bad result: " + result);
        }
    }

    public static void main(String[] args) {

        TestCalculator test = new TestCalculator();
        try {
            test.testAdd();
        } catch (Throwable e) {
            test.nbErrors++;
            e.printStackTrace();
        }

        if (test.nbErrors > 0) {
            throw new RuntimeException("There were " + test.nbErrors +
                                       " error(s)");
        }
    }// end main
}
```

LESSON LEARNED

Objective Test **+** Repeatable Test **=** Simple Test Program

Disclaimer:
The previous example showed a *naive* way to test (a.k.a. the **wrong** one)

That was **not** JUnit!!

JAVA UNIT TESTING FRAMEWORK

- **JUnit** is a simple, open source framework to write and run repeatable tests.
  - It is an instance of the xUnit architecture for unit testing frameworks.
  - (*source*: http://junit.org)

- **JUnit** features include:
  - Assertions for testing expected results
  - Test fixtures for sharing common test data
  - Test runners for running tests

JUNIT 3.X DESIGN RULES

- All the Test classes must extend TestCase
  - Functionalities by inheritance
- All the test method's names **must** start with the "keyword" **test** in order to be executed by the framework
  - `testSomething(...)`
  - `testSomethingElse()`

JUNIT TEST EXAMPLE

```
import junit.framework.TestCase;

public class TestCalculator extends TestCase {

    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals(60, result, 0);
    }
}
```

JUNIT 4.X DESIGN

- Main features inspired from other Java Unit Testing Frameworks
  - **TestNG**
- Test Method Annotations
  - Requires Java5+ instead of Java 1.2+
- Main Method Annotations
  - `@Before, @After`
  - `@Test, @Ignore`
  - `@SuiteClasses, @RunWith`

JUNIT TEST ANNOTATIONS

- `@Test public void method()`
  - Annotation `@Test` identifies that this method is a test method.
- `@Before public void method()`
  - Will perform the method() before each test.
  - This method can prepare the test environment
  - E.g. read input data, initialize the class, ...
- `@After public void method()`

JAVA ANNOTATIONS AT GLANCE

- Meta Data Tagging
  - `java.lang.annotation`
  - `java.lang.annotation.ElementType`
    - `FIELD`
    - `METHOD`
    - `CLASS`
- **Target**
  - Specify to which ElementType is applied
- **Retention**
  - Specify how long annotation should be available

@TEST ANNOTATION

```
package org.junit;

@java.lang.annotation.Retention(java.lang.annotation.RetentionPolicy.RUNTIME)
@java.lang.annotation.Target({java.lang.annotation.ElementType.METHOD})
public @interface Test {
    java.lang.Class<? extends java.lang.Throwable> expected() default org.junit.Test.None.class;

    long timeout() default 0L;

    static class None extends java.lang.Throwable {
        private static final long serialVersionUID = 1L;

        private None() { /* compiled code */ }
    }
}
```

JUNIT ANNOTATIONS (2)

- `@Ignore`
  - Will ignore the test method
  - E.g. Useful if the underlying code has been changed and the test has not yet been adapted.
- `@Test(expected=Exception.class)`
  - Tests if the method throws the named exception.
- `@Test(timeout=100)`
  - Fails if the method takes longer than 100 milliseconds.

JUNIT ASSERT STATEMENTS

- `assertNotNull([message], object)`
  - Test passes if Object is not null.
- `assertNull([message], object)`
  - Test passes if Object is null.
- `assertEquals([message],expected, actual)`
  - Asserts equality of two values
- `assertTrue(true|false)`
  - Test passes if condition is True
- `assertNotSame([message], expected, actual)`
  - Test passes if the two Objects are not the same Object
- `assertSame([message], expected, actual)`
  - Test passes if the two Objects are the same Object

TESTING EXCEPTION HANDLING

try-catch trick!

```
import org.junit.TestCase;

public class TestCalculator extends TestCase {

    public void testThatSummationRaisesAnExceptionOnNegativeInputNumbers(){

        try {
            Calculator calculator = new Calculator();
            calculator.add(-1, -3);
            this.fail(); // Fail the test if no exception has been thrown!
        } catch (RuntimeException) {
            this.assertTrue(True); // Pass the test! ;)
        }
    }
}
```

TESTING THE EXCEPTION HANDLING THE NEW WAY!

Use the **`expected`** parameter of **`@Test`** annotation

```
import org.junit.Test;

public class TestCalculator {

    @Test(expected=RuntimeException.class)
    public void testThatSummationRaisesAnExceptionOnNegativeInputNumbers(){
        Calculator calculator = new Calculator();
        calculator.add(-1, -3);
    } // This is very short, isn't it?!
}
```

TESTCALCULATOR JUNIT 4

```java
public class Calculator {

    public double add(double number1, double number2) {
        return number1 + number2;
    }
}
```

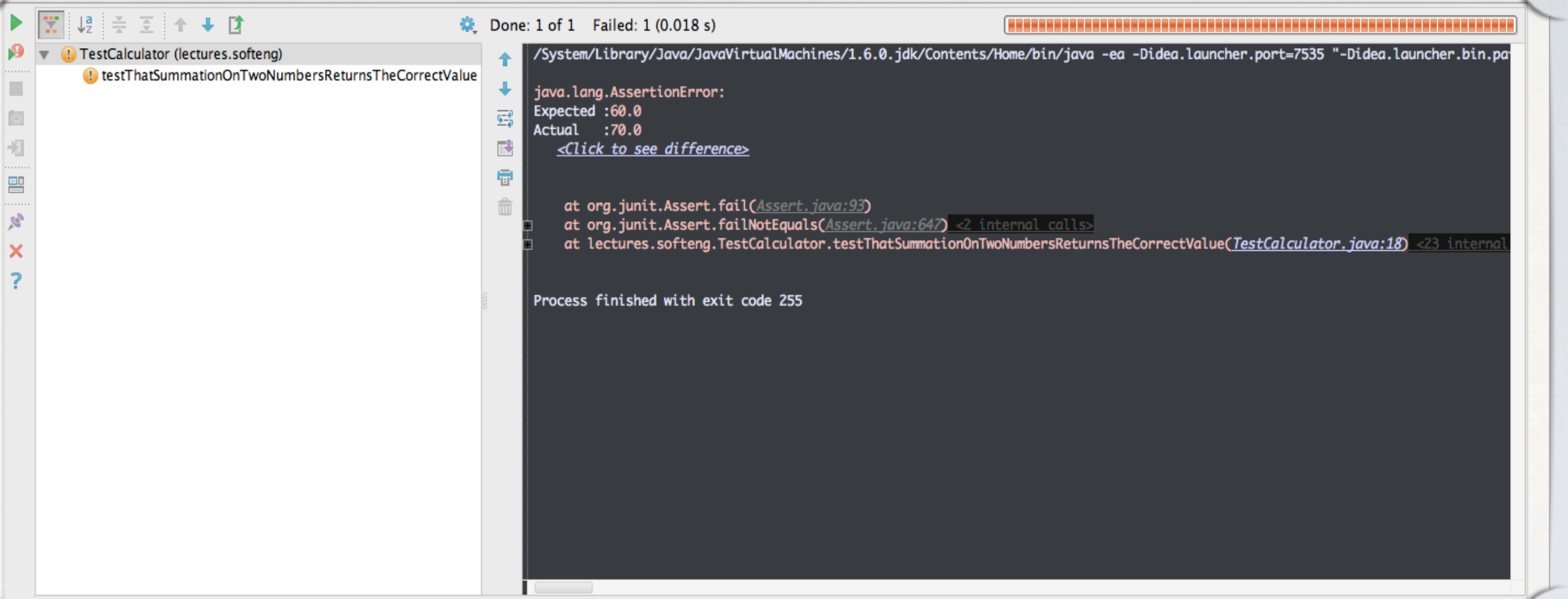
```java
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class TestCalculator {

    @Test
    public void testThatSummationOnTwoNumbersReturnsTheCorrectValue(){
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 60);
        assertEquals(60, result, 0);
    }
}
```

TestAnnotation

JUnit Assert

THE TEST FAILS!

(as expected)

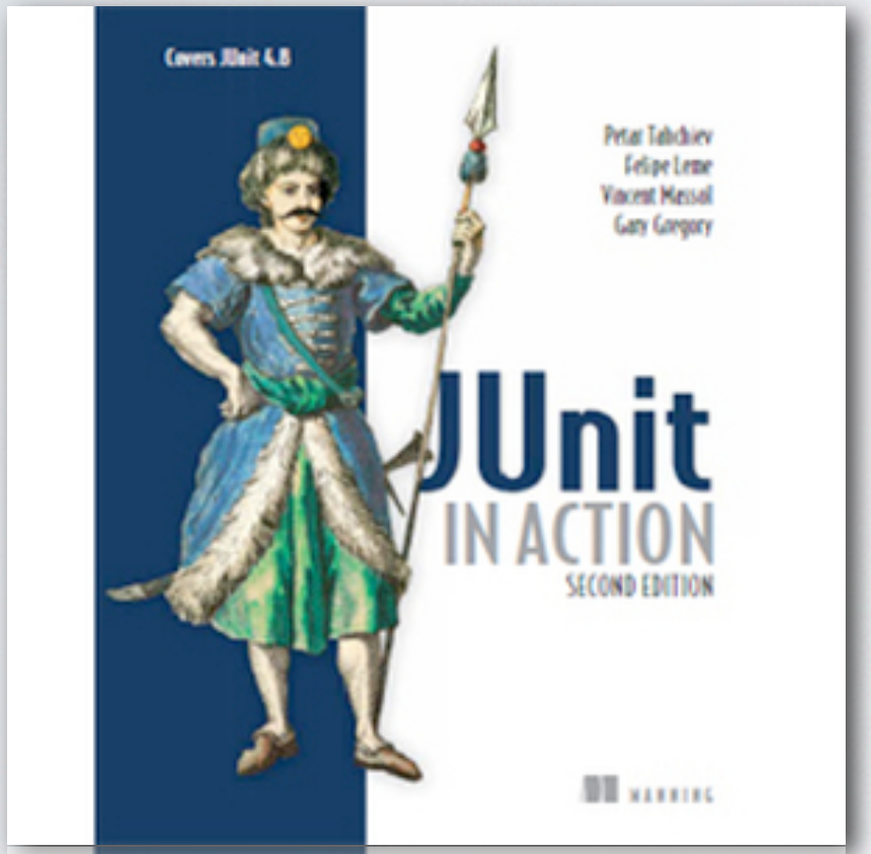


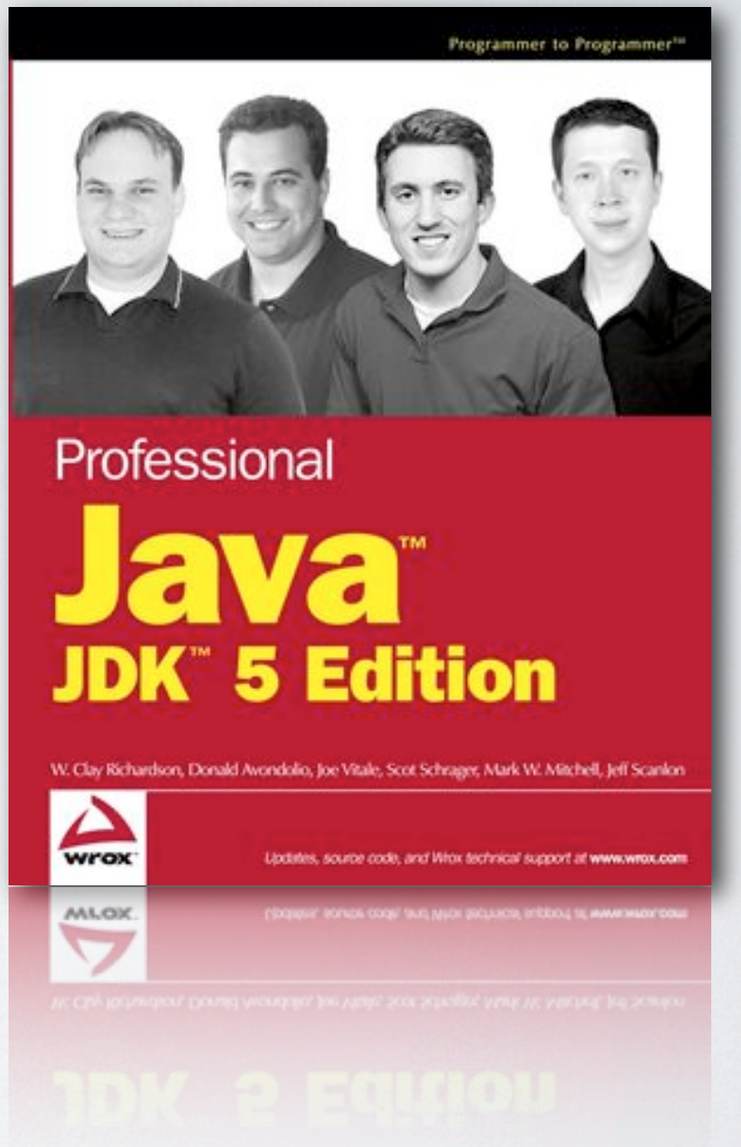
IDE: IntelliJ IDEA 12 CE

http://www.jetbrains.com/idea/

REFERENCES 1/2

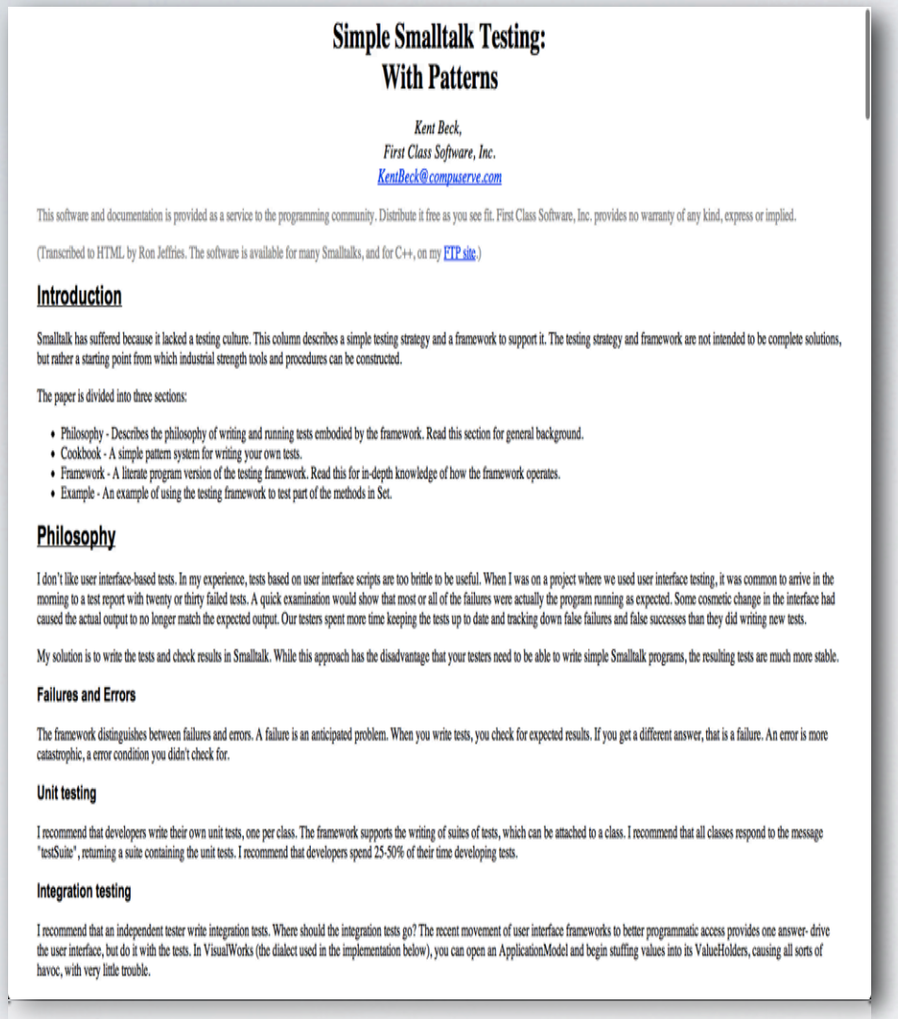
Professional Java JDK 5 Edition
Richardson et. al., Wrox Publications 2006





JUnit in Action, 2nd Ed.
Massol et al. , Manning Pubs 2009

REFERENCES 2/2



Simple Smalltalk Testing:
With Patterns

Kent Beck,
First Class Software, Inc.
KentBeck@compuserve.com

This software and documentation is provided as a service to the programming community. Distribute it free as you see fit. First Class Software, Inc. provides no warranty of any kind, express or implied.

(Transcribed to HTML by Ron Jeffries. The software is available for many Smalltalks, and for C++, on my FTP site.)

Introduction

Smalltalk has suffered because it lacked a testing culture. This column describes a simple testing strategy and a framework to support it. The testing strategy and framework are not intended to be complete solutions, but rather a starting point from which industrial strength tools and procedures can be constructed.

The paper is divided into three sections:

- Philosophy - Describes the philosophy of writing and running tests embodied by the framework. Read this section for general background.
- Cookbook - A simple pattern system for writing your own tests.
- Framework - A literate program version of the testing framework. Read this for in-depth knowledge of how the framework operates.
- Example - An example of using the testing framework to test part of the methods in Set.

Philosophy

I don't like user interface-based tests. In my experience, tests based on user interface scripts are too brittle to be useful. When I was on a project where we used user interface testing, it was common to arrive in the morning to a test report with twenty or thirty failed tests. A quick examination would show that most or all of the failures were actually the program running as expected. Some cosmetic change in the interface had caused the actual output to no longer match the expected output. Our testers spent more time keeping the tests up to date and tracking down false failures and false successes than they did writing new tests.

My solution is to write the tests and check results in Smalltalk. While this approach has the disadvantage that your testers need to be able to write simple Smalltalk programs, the resulting tests are much more stable.

Failures and Errors

The framework distinguishes between failures and errors. A failure is an anticipated problem. When you write tests, you check for expected results. If you get a different answer, that is a failure. An error is more catastrophic, a error condition you didn't check for.

Unit testing

I recommend that developers write their own unit tests, one per class. The framework supports the writing of suites of tests, which can be attached to a class. I recommend that all classes respond to the message "testSuite", returning a suite containing the unit tests. I recommend that developers spend 25-50% of their time developing tests.

Integration testing

I recommend that an independent tester write integration tests. Where should the integration tests go? The recent movement of user interface frameworks to better programmatic access provides one answer- drive the user interface, but do it with the tests. In VisualWorks (the dialect used in the implementation below), you can open an ApplicationModel and begin stuffing values into its ValueHolders, causing all sorts of havoc, with very little trouble.

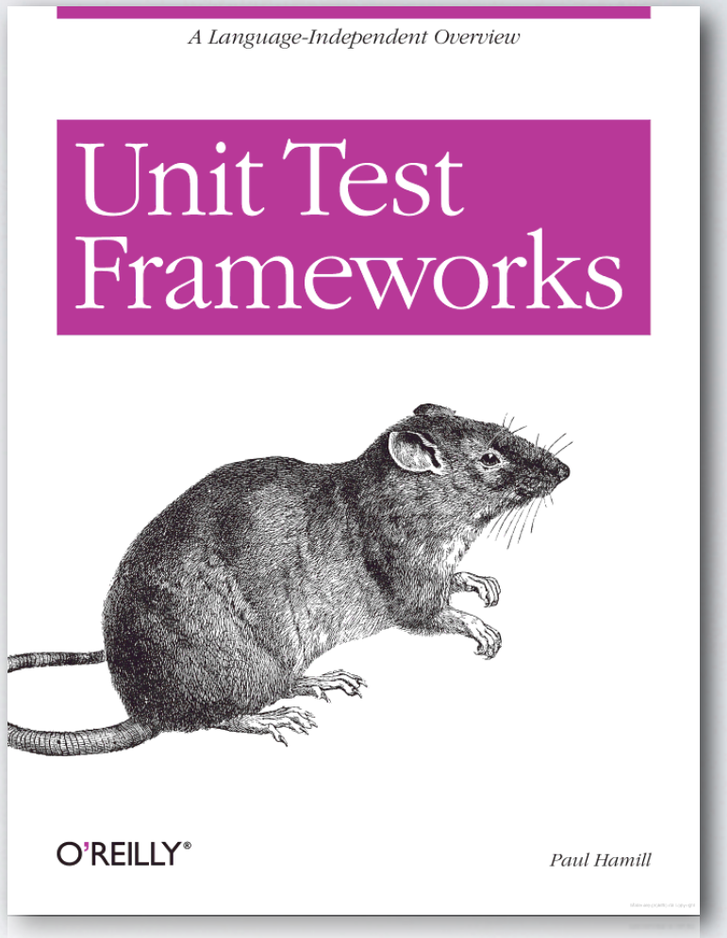
Kent Beck's Original Testing Framework Paper

http://www.xprogramming.com/testfram.htm

Unit Test Frameworks

Tools for High-Quality Software Development

Paul Hamill, O'Reilly Media 2004



A Language-Independent Overview

Unit Test Frameworks

O'REILLY®

Paul Hamill