CSCE 747 Software Testing and Quality Assurance

Tools 04 – Junit4

9/9/2013

Tools Supporting Software Testing

Earlier Tools
- Eclipse/Java

Today
- Junit4

Junit4 References

- Test Infected: programmers like Writing Tests, Java Report, 3(7) 37-50, 1998.
  - Kent Beck and Eric Gamma(Famous Tutorial, but 3.8)
- Practical Unit Testing with Junit and Mockito by Tomek Kaczanowski
  - http://practicalunittesting.com/
  - Source code
- http://en.wikipedia.org/wiki/JUnit
- http://www.vogella.com/articles/JUnit/article.html

Testing Levels

- Unit Tests
  - "make sure the class that you are working on right now works correctly"
  - In isolation; no db; stubs for other classes
- Integration Tests
- System Tests

Unit Tests

- Goal of a Unit Test - "Make sure the class you are working on right now works correctly"
- Scope of a Unit Test
- Test one thing in isolation
- SUT
- Depended On Collaborator DOC

What isn't a Unit Test (Michael Feathers)

- A test is not a unit test if:
  - It talks to the database
  - It communicates across the network
  - It touches the file system
  - It can't run at the same time as any of your other unit tests
  - You have to do special things to your environment (such as editing config files) to run it.

Unit Tests with no Collaborators

- Actually Unit tests for which the SUT (system under test) has no collaborators
- Ridiculous!, of course in most cases
  - We will eventually use stubs to ensure testing insolation
  - But for now let's us focus the discussion

JUnit - Tutorial - Lars Vogel

- Table of Contents
- 1. Introduction to unit testing 1.1. Unit tests and unit testing 1.2. Unit testing with JUnit 1.3. Available JUnit annotations 1.4. Assert statements 1.5. Create a JUnit test suite 1.6. Run your test outside Eclipse
- 2. Installation of JUnit 2.1. Using JUnit integrated into Eclipse 2.2. Downloading the JUnit library
- 3. Eclipse support for JUnit 3.1. Creating JUnit tests 3.2. Running JUnit tests 3.3. JUnit static imports 3.4. Wizard for creating test suites 3.5. Testing exception
- 4. Exercise: Using JUnit 4.1. Project preparation 4.2. Create a Java class 4.3. Create a JUnit test 4.4. Run your test in Eclipse
- 5. Advanced JUnit options 5.1. Parameterized test 5.2. Rules
- 6. Mocking with EasyMock
- 7. Thank you
- 8. Questions and Discussion
- 9. Links and Literature 9.1. JUnit Resources 9.2. vogella Resources

Unit testing with JUnit4

- Creating a JUnit test method
  - via File → New → JUnit → JUnit Test case

```
@Test
public void testMultiply() {
     // MyClass is tested
     MyClass tester = new MyClass();

    // Check if multiply(10,5) returns 50
    assertEquals("10 x 5 must be 50", 50,tester.multiply(10, 5));
}
```


General approach JUnit4 within Eclipse

- JUnit assumes that all test methods can be executed in an arbitrary order.
- Therefore tests should not depend on other tests.
- To write a test with JUnit you annotate a method with the @org.junit.Test annotation and
- use assert or another method provided by JUnit to check the expected result of the code execution versus the actual result
- run the test, via right-click on the test class and selecting Run → Run As → JUnit Test.

JUnit annotations

Annotation	Description
@Test public void method()	The annotation @Test identifies that a method is a test method.
@Before public void method()	This method is executed before each test. This method can prepare the test environment (e.g. read input data, initialize the class).
@After public void method()	This method is executed after each test. This method can cleanup the test environment (e.g. delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.

Annotation	Description
@BeforeClass public static void method()	method executed once, before the start of all tests.
@AfterClass public static void method()	This method is executed once, after all tests have been finished. --clean-up activities
@Ignore	Ignores the test method. out-of-date, too expensive
@Test (expected = Exception.class)	Fails, if the method does not throw the named exception.
@Test(timeout=100)	Fails, if the method takes longer than 100 milliseconds.

Assert statements

Statement	Description
fail(String)	**Let the method fail. Might be used to check that a certain part of the code is not reached. Or to have a failing test before the test code is implemented.**
assertTrue([message], boolean condition)	**Checks that the boolean condition is true.**
assertsEquals([String message], expected, actual)	**Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.**
assertsEquals([String message], expected, actual, tolerance)	**Test that float or double values match. The tolerance is the number of decimals which must be the same.**

Assert statements

Statement	Description
assertsEquals([String message], expected, actual, tolerance)	**Test that float or double values match. The tolerance is the number of decimals which must be the same.**
assertNull([message], object)	Checks that the object is null.
assertNotNull([message], object)	Checks that the object is not null.
assertSame([String], expected, actual)	Checks that both variables refer to the same object.
assertNotSame([String], expected, actual)	Checks that both variables refer to different objects.

Creating a JUnit test suite

```
package com.vogella.junit.first;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
@RunWith(Suite.class)
@SuiteClasses({ MyClassTest.class,
MySecondClassTest.class })
public class AllTests {
}
```


Run your test outside Eclipse

- org.junit.runner.JUnitCore class provides the runClasses() method which allows you to run one or several tests classes

In your test folder create a new class MyTestRunner

```
package de.vogella.junit.first;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
public class MyTestRunner {
   public static void main(String[] args) {
         Result result = JUnitCore.runClasses(MyClassTest.class);
         for (Failure failure : result.getFailures()) {
                  System.out.println(failure.toString());
         }
   }
}
```


Eclipse support for JUnit

- 3.1 Creating JUnit tests
- 3.2. Running JUnit tests
- 3.3. JUnit static imports
- 3.4. Wizard for creating test suites
  - **select New → Other... → JUnit → JUnit Test Suite**
- 3.5. Testing exception
- 4. Exercise: Using JUnit