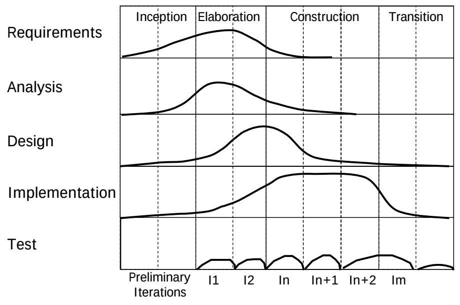
Unit Testing Tools

Wolfgang Emmerich
Professor of Distributed Computing
University College London
http://sse.cs.ucl.ac.uk

Context



2

Learning Objectives

- To be aware of the spectrum of functionality provided by unit testing tools
- To be able to define unit tests
- To be able to measure the quality of unit tests using coverage analysis
- To be able to execute unit tests in a fully automated fashion both inside and outside an IDE

3

Reminder: What is unit testing?

- Modern software production uses modular languages
- Modules may take different forms, e.g.
  - Java / C# / C++ classes
  - Servlets and Server Pages,
  - OSGi Bundles or
  - Components / Beans / Enterprise Beans
- Integration is considerably simplified if quality of modules is established beforehand
- This is done by unit testing
- Involves mundane tasks that should be automated

4

Requirements for Unit Testing Tools

- Definition and Execution of Unit Tests, even if
  - Unit code not yet available (agile test-driven development)
  - Units it depends on are not yet available
- Execution of unit tests
  - Single tests
  - Suites of a number of unit tests
  - Interactively
  - In an automated manner
- Summary and visualization of unit test results
- Analysis of quality of unit tests - how well does a test suite exercise the unit under test?

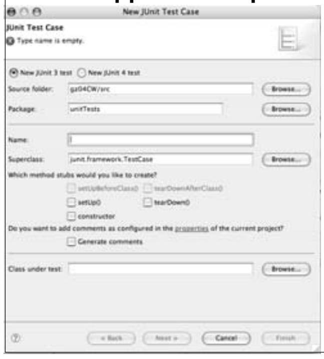
5

De-facto standard: JUnit

- JUnit was developed to unit test Eclipse
- Emerged from Sunit for unit testing Smalltalk classes
- Large number of derivatives:
  - Nunit (for .NET development)
  - DBUnit (for testing DB applications)
  - Httpunit (for testing web applications)
  - …
- Principle idea:
  - Define tests as methods in a test class
  - Define suites of tests in packages
  - Provide assertion framework to specify expected results
  - Provide run-time infrastructure to automate the tests

6

JUnit Support in Eclipse: Test Definition



New JUnit Test Case

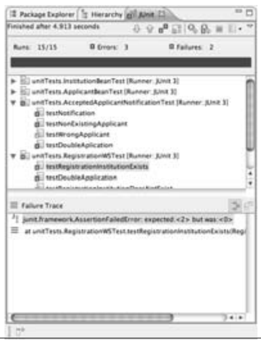
JUnit Test Case
Type name is empty.

(●) New JUnit 3 test ( ) New JUnit 4 test
Source folder: gz04CW/src
Package: unitTests
Name:
Superclass: junit.framework.TestCase
Which method stubs would you like to create?
[ ] setUpBeforeClass() [ ] tearDownAfterClass()
[ ] setUp() [ ] tearDown()
[ ] constructor
Do you want to add comments as configured in the properties of the current project?
[ ] Generate comments
Class under test:

- Wizards for creating test cases of both JUnit3 and JUnit4
- JUnit test cases are methods in Java
- Use JUnit assertion framework which is yet another class.
- To define the test case just use the JDT program editor

7

JUnit support in Eclipse: Test Execution



Finished after 4.913 seconds

Runs: 15/15 Errors: 3 Failures: 2

```
▶ unitTests.InstitutionBeanTest [Runner: JUnit 3]
▶ unitTests.ApplicantBeanTest [Runner: JUnit 3]
▼ unitTests.AcceptedApplicantNotificationTest [Runner: JUnit 3]
    testNotification
    testNonExistingApplicant
    testWrongApplicant
    testDoubleAplication
▼ unitTests.RegistrationWSTest [Runner: JUnit 3]
    testRegistrationInstitutionExists
    testDoubleApplication
```

Failure Trace

```
junit.framework.AssertionFailedError: expected:<2> but was:<0>
at unitTests.RegistrationWSTest.testRegistrationInstitutionExists(Regi
```

- Eclipse provides Junit execution environment for
  - Classes
  - Packages
- Visualizes test case execution results
- Drill-down to obtain assertion failures and exception details
- Supports navigation to failed test cases

8

Using JUnit with ant

- Might want to automate unit test suites for execution outside IDE (because they might take too long)
- Ant build.xml file:

```
<property name="junit.output.dir" value="junit"/>
<target name="junit">
      <mkdir dir="${junit.output.dir}"/>
      <junit fork="yes" printsummary="withOutAndErr">
          <formatter type="xml"/>
          <test name="uk.ac.ucl.cs.sse.test.Stack.StackTest”
                todir="${junit.output.dir}"/>
          <classpath refid="StackTest.classpath"/>
      </junit>
  </target>
```

9

Formatting JUnit reports with ant

- Junit produces text or XML output
- XML can be translated using an XSL stylesheet
- Use the following ant target in your build.xml file

```
<target name="junitreport" depends="junit">
      <junitreport todir="${junit.output.dir}">
          <fileset dir="${junit.output.dir}">
              <include name="TEST-*.xml"/>
          </fileset>
          <report format="frames”
                  todir="${junit.output.dir}"/>
      </junitreport>
   </target>
```

10

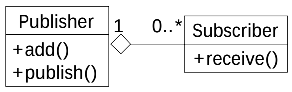
Mock Components

- Unit tests should test just the unit under test and not other units it depends on
- Requires replacing those units
- Can be mundane if classes have large number of dependencies
- Mock frameworks support the systematic replacement of dependencies without writing any code through use of reflection

11

Top-Down white-box testing

- Consider the following design:



- How to test Publisher without also building Subscriber?
  - Assertions need to be formulated on Subscriber
  - Subscriber code needs to exist

12

Using Reflection and Mock Objects

- Basic Idea:
  - Create mock objects for all classes that a class is dependent on
  - Use reflection to avoid having to code it
  - Express assertions in temporal logic based on features exhibited at the interface.
- Example:
  - JMock (http://www.jmock.org)

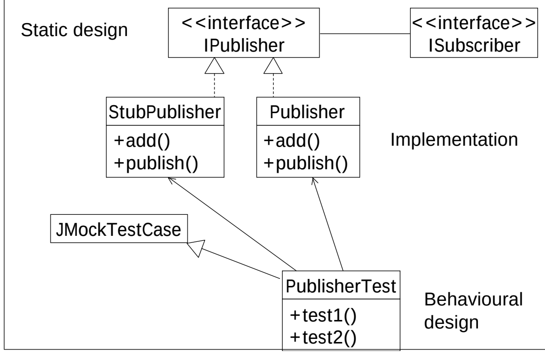
13

JMock example

```
public void testNoSubscriberReceivesMessage(){
  Mockery context = new Mockery();
  final ISubscriber subscriber=context.mock(ISubscriber.class);

  // set up expectations
  context.checking(new Expectations(){{
    never (subscriber).receive("message");
  }});

  // execute
  publisher.publish("message");

  // check expectations are met

  context.assertIsSatisfied();
}
```

14

Test Driven Development with JUnit and JMock



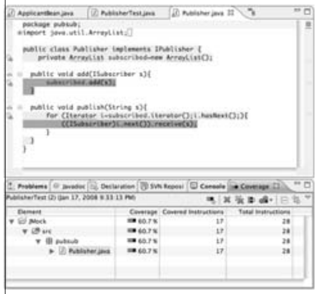
15

Reminder: Coverage Analysis

- White box analysis technique to validate quality of unit tests
- Complementary to Cyclomatic complexity analysis (which determines the maximum number of tests required)
- Different forms
  - Statement
  - Branch
  - def/use
  - Method
  - type coverage

16

Coverage Analysis with Emma



```
package pubsub;
import java.util.ArrayList;

public class Publisher implements IPublisher {
    private ArrayList subscribed=new ArrayList();

    public void add(ISubscriber s){
        subscribed.add(s);
    }

    public void publish(String s){
        for (Iterator i=subscribed.iterator();i.hasNext();){
            ((ISubscriber)i.next()).receive(s);
        }
    }
}
```

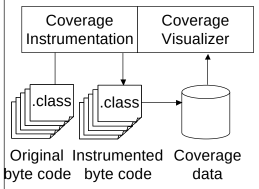
PublisherTest (2) (Jan 17, 2008 9:33:13 PM)

Element	Coverage	Covered Instructions	Total Instructions
JMock	60.7 %	17	28
src	60.7 %	17	28
pubsub	60.7 %	17	28
Publisher.java	60.7 %	17	28

- Supports analysis of coverage
- Visualizes which instructions have been covered (green) and which have not (red)
- Provides statistics
- Supports navigation

17

How Tools Perform Coverage Analysis in Java



- Dynamic analysis technique
- Instrument byte code
- To write details of executed
  - Instructions
  - Methods
  - Classes etc

  to file
- After execution analyze file
- Visualize results

18

Key Points

- Unit testing needs to be automated
- Unit tests are written using programming languages
- Execution within or outside IDE
- Mocking supports isolation of units under test
- Coverage analyzers provide feedback on quality of unit tests

19

References

- Kent Beck. JUnit Pocket Guide. O'Reilly, 2004. ISBN 0-596-00743-4.
- A. Watson and T. McCabe: Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. NIST Special Publication 500-235. http://www.mccabe.com/pdf/nist235r.pdf
- S. Freeman et al.: Mock Roles, not Objects. Proc. OOPSLA 2004. DOI: 10.1145/1028664.1028765
- Emma. http://emma.sourceforge.net/
- EclEmma http://www.eclemma.org/

20