Unit Testing Workshop

Dr. Peter Lappo
www.smr.co.uk

Agenda

- How do you use the XUnit testing tools?
- What is available?
- Are they useful?

- But first some context.

Testing

- Most projects carry at two distinct levels of testing.
  - Unit Testing.
  - Acceptance Testing.
- Unit tests are written and owned by developers.
- Acceptance tests are written and owned by customers.

Why Test?

- Proves something has been completed.
  - no test
  - no completion
  - no customer sign-off
- Proves software works as intended.
- Easier to evolve and refactor software.
- Use tests to reproduce defects.

Acceptance Testing

- Black box testing.
- Written by customer or business analyst.
- Detailed specification of the system.
- Different flavours e.g.
  - Functional or System
  - Integration
  - User Acceptance
  - Operational

Example Acceptance Tests

- Story: Login

- Entering a known user name and password results in the main menu appearing.
- Entering an unknown user name or password results in the login screen being re-displayed.

- Ideally should be automated.

Unit Testing

- White box testing,
  - look into code to find ways to break it.
- Test first design (another workshop!).
- Test all classes.
- Test every thing that could possibly break.
- Automated (essential).

- If there is no test it does not exist!

What to Test

- Everything that could break.
  - Use your conscience and experience.
  - If in doubt write more tests.
- Anything we want to remember.
- Anything that looks interesting.
- Not all combinations.
- Not invalid data unless it can be received.

How to Write Tests

- Look at code and write tests to break it.
- If a class depends on another class such as inner classes.
  - Need to test both.
- For loops test for first, one in the middle and the last one.

Aside

- Small functions are MUCH easier to test than big ones.
- Consequence of good unit testing is simpler and better designed code.

How do you use the XUnit testing tools?

Two Ways To Test

- Write tests which produce text to a file.
- Inspect text.
- Rerun tests when code changes and compare text output.

- But comparison may produce lots of false negatives.
- Need to re-inspect text after adding new tests.

Alternatively

- Write tests which get responses from code.
- In the test compare responses against expected outcomes.
- Send failure messages to console.
- Rerun tests when code changes and look for failure messages.

- No false negatives.
- No need to re-inspect test results.

Technical Background

- Some programming languages (e.g. Eiffel) allow post conditions to be programmed into a function.
- XUnit tests are analogous to post conditions except they are external to the code.
- Assertions are used to determine if the outcome of a test is correct.

For Example

- The following test searches for a User object with a userID and asserts an object is returned and its userID is the same as expected.

```
User user =  find(userID);
assertNotNull(user);
assertEquals(userID, user.getUserID());
```

Contrast With

- The same test using text output.

```
User user =  find(userID);
if (user != null)
   System.out.println("user is " + user.getUserID());
else
   System.out.println("Error: User not found");
```

- Run program and redirect output to a file.
- Compare text output with expected results and previous result files.

Test Fixtures

- Most of the XUnit implementations have concept of a test fixture.
- This is nothing more than the "stuff" that needs to be there for each test.

- For example the preceding test may have required a database connection to exist, or perhaps some data structures are needed.

XUnit Test Fixtures

- Most XUnit tools allow the tester to create a test fixture before each test and destroy the fixture after the test.
- Consequently tests are independent of each other and failures are not propagated.
- The naming convention is:
- setUp – to create a test fixture
- tearDown – to destroy a test fixture

More Than One Test Fixture

- Most modern languages have the concept of class.
- Tests are normally grouped into a class and each class has its own test fixture.
- This provides a great deal of flexibility and means test fixtures only need to be made for the tests in the class.

Test Case and Test Suite

- Individual test functions are called test cases.
- An application could have many test cases.
- Test cases are grouped into test suites to help organise them.

What is available?

Resources

- Loads available, mostly open source.

www.xprogramming.com/software.htm

www.junit.org/news/extension/index.htm

www.c2.com/cgi/wiki?TestingFramework

Implementations Available For

Java	JUnit
C++	CppUnit
Visual Basic	VBUnit
Active Server Pages	ASPUnit
PHP3	PHPUnit
SQL	DBUnit
XML	XMLUnit
XSL	XSLUnit
Java Swing	JFCUnit
Others....	

Roll Your Own

- If your language is not covered it is relatively easy to write your own.
- For a simple framework, see

www.c2.com/cgi/wiki?PoorMansTestingFramework

Are they useful?

Undoubtedly

- An XP project I am working has approximately 130 tests covering 63 Java classes.
- Gives a lot of confidence that the software will work as expected.
- Makes software releases less fraught.

But

- Unit testing takes time.
- Needs discipline.

Demonstration

- The Test being demonstrated is from an XP project I'm working on.
- The class being tested is ItemRange.java which represents a table in a laboratory information and test management database.
- The test retrieves 3 values from a "mock" database and compares them against expected values.

End