Testing Automation

“Design your world so that you can’t help but succeed”

Creating a no-fail environment

• Version Control
• Testing
• Automation

Testing from “Coding in Fear” to “Coding with Confidence”
Testing Tools and Techniques

1. Unit Testing with JUnit
2. Mock Techniques
3. Domain Specific JUnit Extensions
4. Code Coverage Analyzers
5. Continuous Integration Tools
6. Design for Testing
Categories of Tools (1)

• General purpose test frameworks
  – JUnit
  – NUnit

• Support libraries
  – Mock Objects

• Domain-specific test frameworks
  – HttpUnit
  – XmlUnit
  – JfcUnit

• Memory management analyzers
  – Valgrind
Categories of Tools (2)

- Code coverage analysers
  - Clover
  - Quilt
- Test generators
  - jTest
- Stress Testing Tools
  - LoadSim, JUnitPerf
- Static analysis
  - Lint
  - FindBugs
JUnit
xUnit

• xUnit Family
  – Simplicity
  – Define, run, and check

• Instance: JUnit
  – Very simple: 5 classes in junit.framework
  – Many extensions
  – http://www.junit.org

• Instance: NUnit
  – .NET Attributes
  – http://nunit.sf.net
import junit.framework.TestCase;

class TestSimple extends TestCase {

    TestSimple(String name) {
        super(name);
    }

    public void testSplitEmpty() {
        String[] result = " ".split("\s+");
        assertEquals("Should not produce a token", 0, result.length);
    }

    public void testSplit2() {
        String[] result = "a bc ".split("\s+");
        assertEquals("Should be 2 tokens", 2, result.length);
    }
}
JUnit: Grouping Tests

**TestSuite**

- `addTest(Test)`
- `addTestSuite(Class)`

How to construct a Test?

```java
new MyTest("testFoo")
```

or

```java
new MyTest("Some text") {
    public void runTest() {
        testFoo();
    }
}
```
JUnit: Grouping Tests

public class TestFoo extends TestCase {

    public TestFoo(String method) { ... }

    public void testFoo() { ... }
    public void testBar() { ... }

    public static Test suite() {
        TestSuite result = new TestSuite();
        result.addTest(new TestFoo("testFoo"));
        result.addTest(new TestFoo("testBar"));
        return result;
    }
}

or:

    return new TestSuite(TestFoo.class);
JUnit: Grouping Tests

Typical composition class:

```java
public class TestAll {
    public static Test suite() {
        TestSuite result = new TestSuite("All tests");
        result.addTestSuite(TestFoo.class);
        result.addTestSuite(TestBar.class);
        return result;
    }
}
```
JUnit: Running a Test

**TestRunner**

- invokes `suite` method
- or constructs `TestSuite`

Graphical

$ java junit.swingui.TestRunner

Textual

$ java junit.textui.TestRunner
JUnit: Assertions

“Real testing checks results”

- assertTrue, assertFalse
- assertEquals
- assertSame
- assertNotNull
- fail

Advice: Use assert* with a failure message.
JUnit: Fixture per Test

private XMLReader _reader;

protected void setUp() throws Exception {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setNamespaceAware(true);
    factory.setValidating(false);
    _reader = factory.newSAXParser().getXMLReader();
}

protected void tearDown() {
    _reader = null;
}

public void testFoo () throws Exception {
    _reader.parse(new InputSource(new StringReader("<Foo/>")));
    // I want to assert something :(
}

Fixture setUp and tearDown is performed for every test.
JUnit: Fixture per Suite

```java
TestSetup wrapper = new TestSetup(suite) {
    protected void setUp() {
        oneTimeSetUp();
    }

    protected void tearDown() {
        oneTimeTearDown();
    }
};

Don’t use static initializers.
Must be in static variables.
JUnit: (Excepted) Exceptions

Not expected: don’t catch, just throw!

```java
public void testRead () throws IOException {
    StringReader reader = new StringReader("SWE!");
    assertEquals("First char must be S", reader.read(), 'S');
}
```

Expected: try-fail-catch

```java
public void testList () {
    List<String> list = new ArrayList<String>();
    list.add("Foo");

    try {
        String result = list.get(1);
        fail("Not a valid index.");
    } catch(IndexOutOfBoundsException exc) {
        assertTrue(true);
    }
}
```
JUnit: Best Practices (1)

• Make a test fail now and then
  “Spring the Trap”

• Keep tests independent
  – No order guarantee
  – Separate instances for every test

• Custom, domain-specific asserts

• Design test code.
  – Production/Test : 50/50
  – Tests are not a list of statements.
JUnit: Best Practices (2)

- Test just one object
  - Mock up related objects

- Write test back (asserts) first.

- Don’t use the same value twice \((1 + 1)\)

- Don’t break **encapsulation** for testing.
  - Test code in same package
  - Use parallel source trees

- Testing is not an end in itself
JUnit: More Automation

Why not build in IDE?

- Poor automation, building from source, deployment

⇒ Integrate in automated build process.

```xml
<junit printsummary="yes" haltonfailure="yes">
  <classpath>
    <path refid="project.cp"/>
    <pathelement location="${build.tests}"/>
  </classpath>
  <formatter type="plain"/>
  <test name="org.foo.TestAll" />
</junit>
```

batchtest: compose tests in Ant
JUnit: Fancy Reports

Formatter

```xml
<junit ...>
...
<formatter type="xml"/>
<test todir="$builddir/reports" ... />
</junit>
```

Report Generation

```xml
<junitreport>
<fileset dir="$builddir/reports">
<include name="TEST-*.xml"/>
</fileset>

<report todir="$builddir/reports/html"/>
</junitreport>
```
JUnit: Further Reading

- “Pragmatic Unit Testing in Java and JUnit”
- “JUnit in Action”
- “Java Open Source Programming with XDoclet, JUnit, WebWork, Hibernate”
- http://www.junit.org
Mock Techniques
Mock Objects: Rationale

- Test just one thing
- Tests should focus. Be Independent.
- Faulty code ⇔ Test

But ... What if code is not just ‘one thing’?  
⇒ **Fake** the other ‘things’!  
⇒ **Mock objects**
Use Mock Objects If . . .

- Behaviour that is beyond your control.
- Exceptional situations (difficult to reproduce)
- External resources (server, database)
- Lot of code to setup.
- Expensive, poor performance
- Reflection and interaction: How was code used?
- Non-existing code.
Solution: Test the Interactions

- More difficult with state
  - Is the content type set?
  - Is the 'temp directory' method not invoked twice in a session?
  - Is a topic scheduled for indexing?
  - Is a stream closed?

- Almost impossible with state
  - Is the request’s pathinfo requested when ShowFile is invoked?
  - Are the access flags checked when a topic is requested?
  - Is the log function invoked with an error?
Mock Object Generators

At compile-time

• MockMaker – http://mockmaker.sf.net
• MockCreator – http://mockcreator.sf.net

At runtime

• DynaMock – http://www.mockobjects.com
• jMock – http://www.jmock.org
• EasyMock – http://www.easymock.org

Features

• Define Expectations
• Mock-ready implementations of standard libraries
Mock Objects: EasyMock

Expectations as actual method calls!

Mock Control

- Record and replay mode.
- Control, StrictControl, NiceControl

MockControl methods

- getMock()
- replay(), verify()
- setReturnValue, setThrowable(…)

http://www.easymock.org
private XMLReader _reader;
private MockControl _control;
private ContentHandler _handler;

protected void setUp() throws Exception {
    ...

    _control = MockControl.createNiceControl(ContentHandler.class);
    _handler = (ContentHandler) _control.getMock();
    _reader.setContentHandler(_handler);
}

public void testFoo () throws Exception {
    _handler.endElement("", "Foo", "Foo");
    _control.replay();

    _reader.parse(new InputSource(new StringReader("<Foo/>")));  
    _control.verify();
}
protected void setUp() throws Exception {
    ...
    _control = MockControl.createStrictControl(ErrorHandler.class);
    _handler = (ErrorHandler) _control.getMock();
    _reader.setErrorHandler(_handler);
}

public void testIllegalFoo() throws Exception {
    _handler.fatalError(null);
    _control.setMatcher(MockControl.ALWAYS_MATCHER);
    _control.replay();

    try {
        _reader.parse(new InputSource(new StringReader("<Bar></Bar>")));
    } catch (SAXParseException exc) {
        assertTrue(true);
    }
    _control.verify();
}
Mock Objects: Limitations

- Requires an interface: implementation must be substitutable.
  - Design for Testing
  - Example: test if a `strong` element is created.  
    Use `JDOMFactory`

- Requires ‘mock-ready’ libraries. Many libraries do not allow this kind of configuration.

- DynaMock: no type checking by compiler.

- DynaMock: verbose constraints and return values.

- Easy to break the real code: what does it use?
Mock Objects: Further Reading

• “Endo Testing: Unit Testing with Mock Objects” (XP2000)

• “Mock Objects” in “Pragmatic Unit Testing”
  Sample chapter available online

• www.mockobjects.com
Domain specific extensions
Some Popular Extensions

• Web Applications
e.g. HttpUnit

• Performance
e.g. JUnitPerf

• Integration
e.g. Cactus (J2EE)

• GUI Applications
e.g. JfcUnit

• XML Processing
e.g. XML Unit
JUnitPerf: Test Decorators

**TimedTest**

Test testCase = ...;
Test timedTest = new TimedTest(testCase, 2000);

**LoadTest**

Test testCase = ...;
Test loadTest = new LoadTest(testCase, 25);

**RepeatedTest**

Test testCase = ...;
Test repeatedTest = new RepeatedTest(testCase, 10);
Timer timer = new ConstantTimer(1000)
Test loadTest = new LoadTest(repeatedTest, 25, timer);

http://www.clarkware.com/software/JUnitPerf.html
XML Processing: XMLUnit

Handy utils for XML, XPath, XSLT.

**XMLTestCase: assertXMLEqual**

- Identical / similar
- Meaningful diff messages

**Transform**

- Easy to use in XSLT tests

**Validator**

- Tools for DTD and Schema validation

http://xmlunit.sf.net
Testing Web Applications
Some Issues

• Security
  – Resources protected?
  – Unexpected input?
  – ... not generated by a browser?

• Validation of XHTML, CSS, JavaScript

• Check broken links

• Are files served correctly? (type)

• Does my caching mechanism work?

• Can you handle loads of users?
Unit Testing of Java Web Applications

- **Servlet Unit Testing**
  - Mock objects for servlets container
  - Servlet container: difficult to mock
  - Mock Objects Framework implements servlet mocks

- **In-Container Integration Testing**
  - Cactus (J2EE integration tests)
  - Rather heavy-weight

- **Functional Testing**
  - Simulate browser
  - HttpUnit
Servlet Testing with Mock Objects

Approach

• Mock the objects on which a servlet depends.
• Mock the servlet container.

Implementation (mockobjects.com)

• MockHttpServletRequest
  – Supports expectations
e.g. request.setExpectedError(SC_NOT_FOUND)
  – Verify: request.verify();

• MockServletConfig

• MockServletInputStream

• ...

Servlet Container Mock: (Dis)advantages

- Possible to mock the dependencies of servlets
- Tends to be close to functional testing
- Why not test at the client-side?

In general: Leave servlet specific environment as soon as possible.
HttpUnit: Functional Testing

• Automated functional tests ⇒ No need for checking the site by hand.

• Targets programmers

• Simplicity. Just code.

• Well-designed web applications:
  – Request/response is unit
  – Possible to test

Supports frames, forms, JavaScript, basic HTTP authentication, cookies and redirection.

http://httpunit.sf.net
HttpUnit: Hello World

- WebClient / WebConversation
- WebRequest
-WebResponse

```java
WebClient client = new WebConversation();
WebResponse response = client.getResponse("http://www.cs.uu.nl/groups/ST/Gw");
WebLink link = response.getLinkWith("Vision");
WebRequest request = link.getRequest();
```

Links:
- WebLink[] getLinks
- WebLink getLinkWith
- WebLink getLinkWithID
- WebLink getLinkWithImageText
HttpUnit: Working with Forms

Retrieving values from a form:

```java
WebForm form = response.getForms()[0];
assertEquals("Martin Bravenboer", form.getParameterValue("name"));
assertEquals("", form.getParameterValue("password"));
assertEquals("", form.getParameterValue("password2"));
```

Submitting a form:

```java
form.setParameter("name", "Martin Bravenboer");
form.setParameter("password", "foobar");
form.setParameter("password2", "foobar");
form.submit();
```
Testing Web Applications: Further Reading

- “Testing a Servlet” in “Mock Objects” in “Pragmatic Unit Testing”
- “Unit Testing Servlets and Filters” in “JUnit in Action”
- “In-container Testing with Cactus” in “JUnit in Action”
- “Functional Testing with HttpUnit” in “Java Tools for Extreme Programming”
Code Coverage Analyzers
Code Coverage Analysis

- How well is the code exercised?
- Improving Coverage
  - Write more tests
  - Refactoring
    * Eliminate duplication
    * More reuse

Implementations

- Clover – Closed Source. Free for non commercial use
- JCoverage – GPL/Closed Source  http://www.jcoverage.com
- Quilt – Open Source  http://quilt.sf.net
- NoUnit – Open Source  http://nounit.sf.net
### Coverage Report

**Dashboard**
- **Coverage timestamp:** Thu Aug 06 2004 12:35:30 EST
- **Package stats:**
  - LOC: 4,649
  - Methods: 213
  - NCLOC: 2,974
  - Classes: 34

#### Package Overview

<table>
<thead>
<tr>
<th>Package</th>
<th>Conditionals</th>
<th>Statements</th>
<th>Methods</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>com.puppycrawl.tools.checkstyle</code></td>
<td>67.6%</td>
<td>79.7%</td>
<td>87.8%</td>
<td>78.8%</td>
</tr>
</tbody>
</table>

#### Classes Coverage

<table>
<thead>
<tr>
<th>Classes</th>
<th>Conditionals</th>
<th>Statements</th>
<th>Methods</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AllTests</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Main</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PropertyCacheFile</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>TreeWalker.SilentJava14Recognizer</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DefaultLogger</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PropertiesExpander</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Checker</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Checker.ErrorCounter</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>TreeWalker</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DebugAuditAdapter</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PackageNameLoader</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DefaultConfiguration</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PackageNameLoaderTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ConfigurationLoader</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ConfigurationLoaderTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ConfigurationLoader.InternalLoader</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PackageObjectFactoryTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>UtilTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>XMLLogger</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>StringReaderTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>CheckerTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>BaseCheckTestCase</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>BaseCheckTestCase.BriefLogger</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DebugChecker</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DebugFilter</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>DefaultContext</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>OptionTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>PackageObjectFactory</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>StringReader</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>XMLLoggerTest</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>XMLLogger.Test.ThrowAll</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public void addException(AuditEvent aEvt, Throwable aThrowable)
{
    synchronized (mErrorWriter) {
        mErrorWriter.println("Error auditing " + aEvt.getFileName());
        aThrowable.printStackTrace(mErrorWriter);
    }
}

protected void closeStreams()
{
    mInfoWriter.flush();
    if (mCloseInfo) {
        mInfoWriter.close();
    }
}

mErrorWriter.flush();
if (mCloseError) {
    mErrorWriter.close();
}
Continuous Integration Tools

• Types of Automation
  – Commanded Automation
  – Scheduled Automation
  – Triggered Automation

• Integrate components as early and often as possible.

• Server monitors repository

• Avoid big bang integration

• Requires
  – Automated build process
  – Automated distribution process
  – Automated deploy process
Continuous Integration: Implementations

- CruiseControl (for Java)
- AntHill
  - Handle multiple projects and their dependencies
  - Separate checkouts
  - Time-based build scheduler (instead of commit push)
  - Bad experience with time-based scheduling.
- Daily Build System
  - continuous stresses that a daily build is not enough
  - StrategoXT: Daily builds - Nix Buildfarm
    - http://www.program-transformation.org/Tools/DailyBuildSystem
- Nix Buildfarm
Continuous Integration: CruiseControl

• Repeated builds
• Builds jobs implemented in Ant
• Trigger: modification checks
  CVS, Subversion, VSS, file system, HTTP and more.
• Publishers
  Status, Email, SCP, FTP

Limitations
• SCM logic in build files
• No managing of dependencies (Maven is supported)

http://cruisecontrol.sf.net
Nix Continuous Integration System

- Nix: dependencies and their configuration
- Nix: caching for free
- Functional abstraction: variants
- Distributed builds
- Building RPMs in User-Mode Linux

Prototype limitations
- Supports Subversion only
- Lack of status interface
- Abstractions targeted at GNU packages.

Eelco Dolstra will tell more about this.

http://www.cs.uu.nl/groups/ST/Buildfarm
Continuous Integration: Further Reading

- “Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Applications”
  Excerpts available

- “Continuous Integration” (Martin Fowler)
  http://www.martinfowler.com/articles/continuousIntegration.html
Design for Testing
Design for Testing

“Your code sucks if it isn’t testable”

Dave Astels, an Artima Blogger

● Separation of concerns
  ⇒ Separate, independent tests
  ⇒ Faulty code easier to identify

● Design for flexibility.
  – Testing requires different implementations

● Unit testing improves the design of code
  – Small methods
  – Reduction of side effects
  – More explicit arguments
Design for Testing: ‘Inversion of Control’

- aka ‘Dependency Injection’, ‘Tell, Don’t Ask’
- Connecting stuff
- Don’t let objects collect their own stuff
- Push versus pull
- Container/Context: the container sets everything an object needs
- Allows replacement with a Mock.
Design for Testing: ‘Reduce Coupling’

- No static singletons
  - Cannot be replaced
  - Might not be cleared while running several tests
- Use factories instead of explicit instantiation
  For example “Registry”
  - Scope: process, thread, session
  - Static method can hide a ’thread-scoped’ data

- **Law of Demeter**
  Method target restricted to ‘local’ or ‘global’ objects
Design for Testing: ‘Gateway’

“Encapsulate access to external systems and resources”

- Libraries
- Services
- Databases

**Implementation**
- Create an API for your usage
- Translate into external resource invocation

**Use** “Separated Interface”
- Real Implementation
- “Service Stub” implementation (Mock)
Design for Testing: Further Reading

- “Patterns of Enterprise Application Architecture”

- “Tell, Don’t Ask”
  http://c2.com/cgi/wiki?TellDontAsk

- “Law of Demeter”
  http://c2.com/cgi/wiki?LawOfDemeter

- “Design Patterns: Elements of Reusable Object-Oriented Software”
  http://c2.com/cgi/wiki?DesignPatternsBook