Chapter 8

Mocks & Logging

Writing provably correct code is a minority occupation.

Testing distributed code

Testing code is important
Testing code improves code quality

Testing is a major way of ensuring software correctness.

Lots of books on writing software – rather fewer on testing software.

Testing distributed (multicore) code is more important.
more failure modes

More opportunities for errors and omissions.
•Network failures
•Failures of single nodes

Some programmes perform complex operations on data. Data can in principle be looked at by an individual.

Some programmes process so much data that it cannot be looked at even in principle.

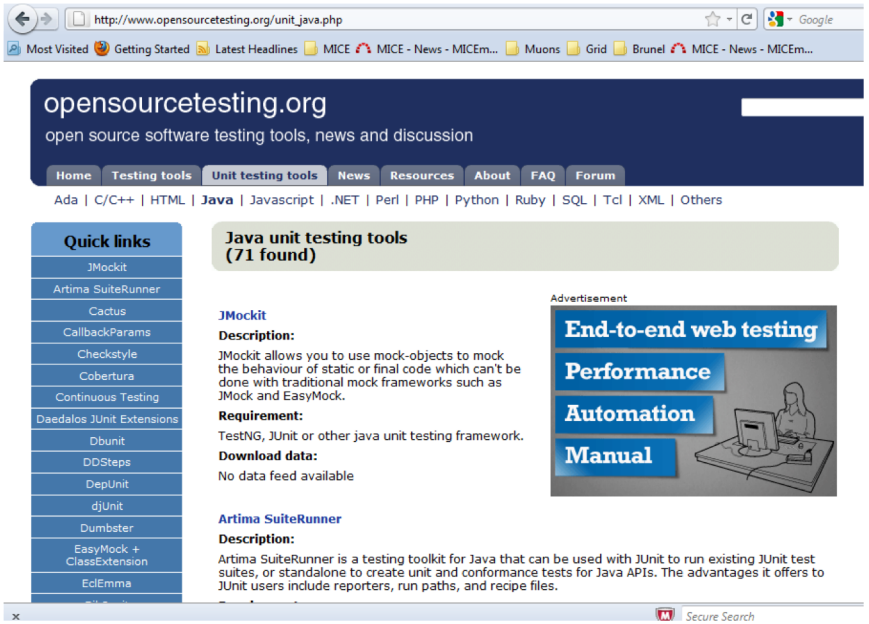
Modern Distributed systems may create systems where there are so many jobs running that you cannot look at all the jobs!

Distributed testing

Writing provably correct code is a minority occupation.

Interactions between the “jobs” create an environment where testing is extremely difficult.

There are no simple frameworks such as are available for single threaded applications.



It is **vital** to make sure that the sub-units of the application work reliably in isolation.
Unit tests

It is important to run tests on the interactions with the sub tasks with each other.
Integration tests/Mocks

Mock Objects

Endo-Testing: Unit testing with mock objects. Mackinnon, Freeman, Craig eXtreme Programming and Felxible processes in Software Engineering – XP2000

See http://www.mockobjects.com For more information

During programme development it is useful to use **mock objects**.

Introduced by Mackinnon, Freeman, Craig 2000

Normal code (non-trivial) code is difficult to test in isolation.
Object is to test **only** one feature at a time and be informed **immediately** there is problem.

"How extremely silly of one not to have thought of it before" T.H. Huxley

Replace real code with code that simulates behaviour.

Code is passed to the objects which they test from the inside.
Similar to stubs **but**

finer granularity
simpler code
Drive the development process

You may re-factor code to allow tests, both unit tests and mock tests.

Claim: testable code is not only better quality (less bugs); easier to maintain,…
But also: intrinsically better structured.

Programmers who write this way become better programmers.

Why Mock

Origin paper suggest reasons

Deferring infrastructure choice : no need to choose the database to proceed with development.

Simplifying : can “mock” a complex system with simple components.

Fault conditions: can be “mocked” when creating the fault conditions is hard/impossible.

Failure conditions local: mock objects report failure conditions at the point of failure.

Beneficial on coding style: testing is difficult without breaking the scope. Designing with mocks reduces the need to exposure the structure of the code

In addition it is possible to “mock” a complex system which might take unacceptably long time to respond.

In general if calls to a service return non-deterministic answers, it is difficult to check that the response to all response classes is accurate.

It reduces unwanted side effects … entries in a database; manipulation of file systems; switching of external devices.

Mock Uses

Simulate behavior of objects, useful when:

a) Has not yet been written.

b) Object returns non-deterministic results: time

c) States difficult to create: a network error

d) Time to create the object is time consuming: render an image.

CMS monte carlo produces a "real" output stream

Mock objects can be created to return boundary values to test operation.

Mocks have the interface of the object they mimic,

Not a stub

More than a stub – must be controllable by the test system. **mock object frameworks** available

Google has a good mocking framework

.

http://seven-mock.sourceforge.net//

Allow the test to set up a mock object so that when invoked by a real object it returns a suitable output. So complex interaction(s) between the object under test and an associated class can be exhaustively tested.

Mocks v stubs

At first sight mocks appear to be another name for a programme stub.

A programme stub also presents the same interface as the object it is “subbing”.
Used in unit/black-box testing. Operation of class under test requires other (unwritten) classes.

They are commonly used in network programming, so development does not rely on two machines (or the existence of a network connection) and local tools can make build decisions without contacting the remote machine.

They are static. They are written to return a single answer (or certainly only a limited range of answers). They are written by hand and in order to pass a test the correct stub must be called.

Mocks are inherently dynamic, integrated into the test framework and framework driven.

My definition – a stub is a piece of code written by the programmer to allow tests.
A mock is an intrinsic part of the testing environment.

Use of Mocks

Mocks are dynamic, part of the testing process;
Not an adjunct

A mock in a testing environment is created dynamically by the test environment in response from instructions to the test environment.
So for a particular test the mock response is defined in that test.

Implies the mock can be used in places where a class has yet to be written
But a new test means no change to the existing mock. Both tests can run.

It can be used like a stub to replace a response from a network resource.

It can be used to provide a response from a class whose response cannot be simulated by the class itself. Anything dependant on outside circumstances.

It can be used to return error conditions or situations which provoke error conditions

Output from a sensor.

Mocks are part of tests.

TDD – Test Driven Design

If the class exists but takes too long to run (or is otherwise too expensive

A programme which picks up the results from say 500 jobs (each of which take 12 hours to run) and performs some sort of amalgamation.
If it takes > 12 hours to run a test it won't be tested often.
TDD encourages regular tests (even overnight releases not possible in the above example).

Distributed jobs typically require large scale resources (not local) and long periods. In this environment use of *mocks* is almost mandatory.

Stubs can always be hand written for each case, but that increases work (*and possibilities for error*).
Unless they are written to document their behaviour it depends on the record keeping of the programmer.

Mocks (*in a reasonable framework*) are self documenting.
Provide audit trail. Ensure that it is unambiguous what a test was and where the failure occurs.

Did I make a mistake in stubbing

Example Mock

Uses EasyMock

```
public class collision {
  public int doSomething(target tObj) {
     … stuff here
    int inform = tObj.getInfo();
    double version = tObj.version();
    boolean success = tObj.setMode("Lund");
     … more stuff here
}
```

to test collision we create a unit test which creates an instance and passes it an instance of the **target** class. But we don't want to use it ….

So using EasyMock we create a test that looks like the following

....

Uses EasyMock

```
public class collisionTest extends TestCase {

 public void testWithMocks() throws Exception {
  target mockCollab = createStrickMock(target.class);

  expect (mockCollab.getinfo().andReturn(121));
  expect (mockCollab.version().andReturn(2.43));
  expect (mockCollab.setmode("Lund").andReturn(true));

  replay(mockCollab)
  new collision.doSomething(mockCollab)
  verify(mockCollab);
  }
}
```

Create a mock object.

Define expected calling sequence and required response.

Run things

```
public class collision {
  public int doSomething(target tObj) {
     … stuff here
    int inform = tObj.getInfo();
    double version = tObj.version();
    boolean success = tObj.setMode("Lund");
     … more stuff here
}
```

If the routine creates an object using the constructor then it cannot be mocked.

Return it from a factory method and get the factory to return a mock object.

Refactor to test

Tendency to test by putting in print out.
Shipped code has most of the printout suppressed
Leave in for diagnosis in case of failure.
Alternative?

Add logging to the code
Java Logging API -- Part of Java 2 Standard Edition Version 1.4.

Log4j -- An open source logging framework from the Apache Jakarta project.

Logging Toolkit for Java -- A logging framework from the IBM

Protomatter Syslog -- A logging framework that is part of Protomatter, an open source collection of Java utility classes.

Java Logging Framework -- A simple logging framework from The Object Guy.

What to look for
Configuration: flexible, dynamic
Loggers: support for multiple loggers
Levels: DEBUG, WARN, ERROR?
Filters: which messages go to where
Output Devices: multiple output devices? files, but output to the console, sockets, JMS,email.
Speed: Logging adds overhead to an application.

When to log

testing
log4j can write to remote log4j server. Testing of distributed code, can be all written to a central place. Unit testing has trouble with this.

production
remote writing allows central collation of correlated error messages which may occur in various places.

unexpected conditions (i)
encounter unexpected error conditions can write a record of system state which can be used for subsequent analysis.

unexpected conditions (ii)
a system which has some recurrent problem can be run with more logging information enabled for subsequent analysis

...

understanding use
logging leaves an audit trail – it can be useful to see how the code is being used in practice.
Unused code can be *removed*
The performance of slow code can be addressed.
The development of heavily used parts of the code can be emphasised.
It may allow the identification of performance problems not connected to the code which is underperforming. Task A using so many resources that task B is adversely affected – task A may appear to be running correctly.

Logging is not debugging.
it may complement testing
it may help in the development

any output statement which is informational or a warning and likely to stay in the code when it ships should probably be a logging statement.

Native java logging API is now available – log4j is generally thought to be better.

Lessons

Testing concurrent systems is harder than single threaded ones.
The group at Malaga university (neo.lcc.uma.es) is looking at various other approaches to check the correctness of the finished system including;
Ant Colony Optimisation
Particle Swarm Optimisation
Simulated annealing

Especially for safety critical system **being careful** is not enough.

Mocks help to test out parts of a system in terms of interactions with other systems which cannot be easily tested with the systems themselves.

Logging is part of programme design and development. A suitable logging solution should be identified during the design phase.
Logging like error handling is part of the design and not an optional add-on.