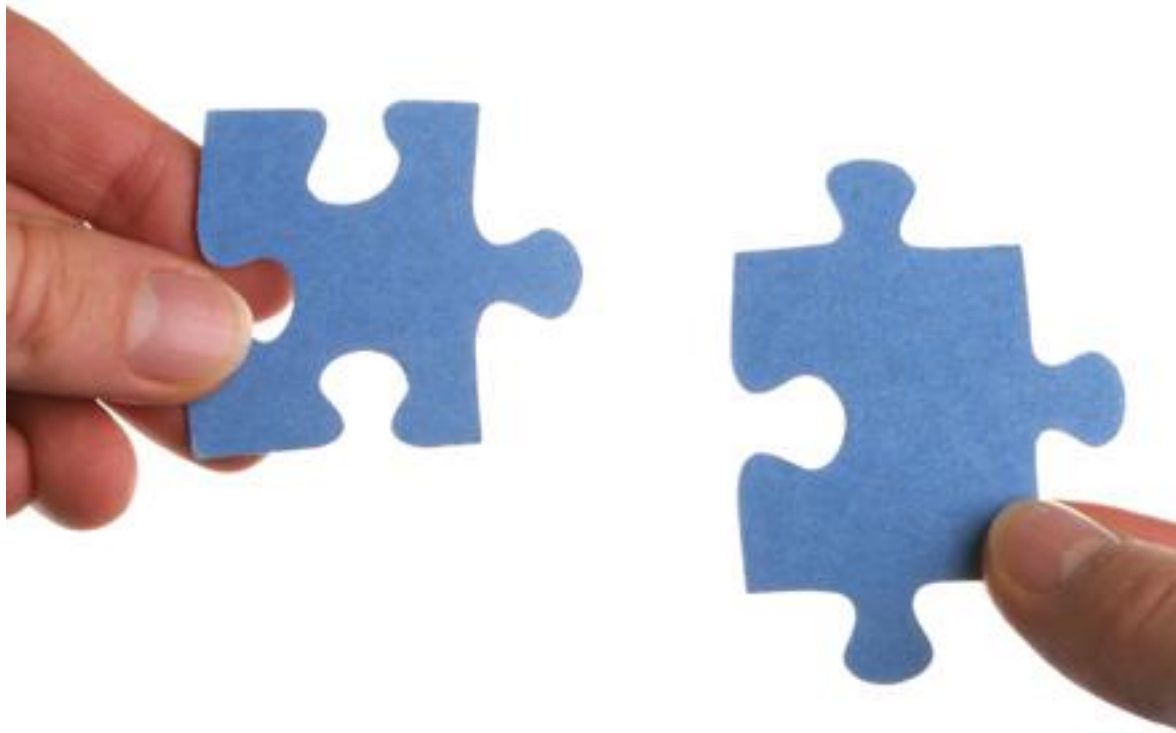
Finally, an agile test strategy (that works)!

VI REALISERER
DIN
KONKURRANSEKRAFT



Who is Erik Drolshammer?

- Consultant Objectware AS – erik.drolshammer@objectware.no

- Master in Computer Science from NTNU
  - *Master thesis*

    Improved Backward Compatibility and API Stability with Advanced Continuous Integration

Who is Bård Lind?

- Senior consultant Objectware AS – bard.lind@objectware.no

- Finance industry
- Operations
- Java since 1997

- Contributor to Objectware's SOA/EA R&D team

Agenda

1. Background and problem description
2. OW Test Model
3. Implementation
4. Summary

Background

Complex project;

- Many remote services within the project
- Integration with multiple external systems
- JMS
- OSWorkflow
- Webstart
- C, C++, Java 1.5, Java 6
- Solaris, Linux, OSX, Windows
- Performance critical data processing

Problem description

What is an integration test?
What is a system test?
How to decide which definition of a term to use?
How to synchronize the terminology in the company? Across projects?
Communication with the customer? With project manager?
Agile and TDD versus the V-model
When to run the different types of integration tests?
Before check in to the version control system?
Run by CI server?
Manual test? Automatic test?
What must be tested in an expensive environment? Can it be done in a cheaper way?
File structure for tests – how to decide?

Problem description

What is an integration test?
What is a system test?
How to decide which definition of a term to use?
How to synchronize the terminology in the company? Across projects?
Communication with the customer? With project manager?
Agile and TDD versus the classic?
When to run the different types of integration tests?
Before check in to the version control system?
Run by CI server?
Manual test? Automatic test?
What must be tested in an expensive environment? Can it be done in a cheaper way?
File structure for tests – how to decide?

CHAOS

Symptoms

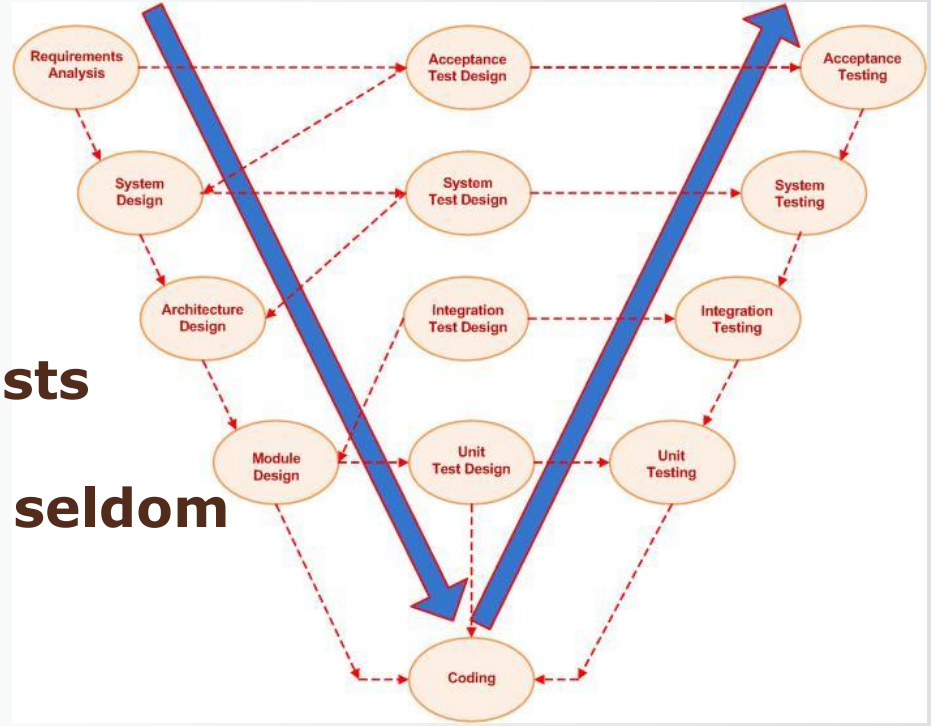
Complex project =>
- Hard to *discuss* tests
- Hard to *write* tests
- Hard to *maintain* tests
- Hard to *change code*
- Hard to run tests

Hard to test => hard to do employ TDD properly

The V-model is not adequate

What we often see:

- **enormous unit tests**
- **too few integration tests**
- **many broken integration tests**
- **integration tests are run to seldom**
- **hard to write system tests**
- **System tests are too fragile**



Source: http://en.wikipedia.org/wiki/Image:V-model.JPG

OW Test Model

OW Test Model

Goal
make it easier to test complex projects

How
pragmatic approach, support agile work patterns

Authors
Bård Lind and Erik Drolshammer

Test categorization

Terminology

+ Unit tests (simple and well-defined)

- Integration test <- too inprecise

- Functional test <- too inprecise

+ Service test (it is a service oriented world)

+ System test

Tackle complexity #1

Single-responsibility principle:

”A ~~class~~**test** should have one,

and only one, reason to change.” SRP

Tackle complexity #2

Follow general OO practices also for test code, especially

- **Separation of concerns**
- **High cohesion, low coupling**

Tackle complexity #3

It *is* relevant whether a broken build is caused by bad input data or the latest change to the business logic.

-> with or without data

Tests that depend on external systems can be complex and expensive.

-> without or without environment

Categorization matrix

	No data	Data	No environment	Environment
Unit test				
Service test				
System test				

Other drivers

- **Speed**
- Cost
  - Setup cost
  - Test development cost
  - Cost of running tests
- Minimize overlap between tests

Test categorization
+
Continuous Integration

CI Phases
- when to run the different tests

1 - Before check-in to VCS

2 - Multiple times a day*
3 - Nightly
4 – Weekly
5 – System Release

*(e.g. hourly or every second hour)

CI Phases
- when to run the different tests

1 - Before check-in to VCS } **developer**

2 - Multiple times a day*
3 - Nightly
4 – Weekly
5 – System Release
} **CI server**

* (e.g. hourly or every second hour)

Categorization matrix with phases

	No data	Data	No environment	Environment
Unit test	Fast (1-2)	n/a	Fast (1-2)	n/a
Service test	Fast (1-4)	Slow (2-4)	Fast (1-4)	Slow (2-4)
System test	Slow (3-5)	Slow (3-5)	Slow (3-5)	Slow (3-5)

NOTE! This is only an example.

Test categorization

+

Continuous Integration

+

Groups

Groups and CI phases

- Groups are used to filter which tests to run in each phase.
  - One group for each **external dependency**
  - One group for each **data source** or provider
  - One group for very **slow tests**
- One error should (ideally) result in only *one* test failure.

Group example

```
public class MyDatabaseTest {

  public void testSomeSimpleBusinessLogic() {

  }

  @Test(groups = { "database-oracle"})

  public void testDatabaseTransaction() {

  }

}
```

Test Code Location

Driver: Keep code and tests close.

	Location
Unit test	Same module as code
Service test	Separate module (possibly same module)
System test	Separate project (possible only a separate module)

Test categorization
+
Continuous Integration
+
Groups

= OW Test Model

Test model extension points

- Test <-> requirement mapping
- Catalogue of test tactics/patterns
- Code coverage for tests in separate modules and projects

Test <-> requirement mapping

- One user story <-> one *group of tests*
- Difficult to set up mapping => low quality user stories
- TestNG groups can be used to display the mapping.
- BDoc?

Implementation

Prerequisites

Continuous Integration (CI) server

Version Control System

Maven Artifact Repository

Prerequisites

Continuous Integration (CI) server

Version Control System

Maven Artifact Repository

Enterprise Maven Infrastructure

Implementation requirements

Groups

Dependencies between tests and groups of tests

setUp and **tearDown** at test and group level

Dependency injection

Embedded, in-memory alternatives for heavyweight technologies

Implementation requirements

Groups

Dependencies between tests and groups of tests

setUp and **tearDown** at test and group level

TestNG

Dependency injection

Spring

Embedded, in-memory alternatives for heavyweight technologies

JMS: **ActiveMQ**

JTA**: Atomikos, JOTM**

JPA: **Spring**

EJB3: **Spring Pitchfork**

JAX-WS: **XFire**

Servlet container: **Jetty**

Database: **HSQLDB, Apache Derby**

Source: http://wiki.community.objectware.no/display/smidigtonull/Lightweight+alternatives+to+heavyweight+technologies

Software stack example

Maven (release and surefire plugins are central)

TestNG, EasyMock

Spring 2.x, ActiveMQ, HSQLDB

Continuum

Artifactory

Subversion

Benefits and summary

Observations

Improved communication

More tests, but fewer lines of test code. (Minimal overlap between tests.)

Less responsibility per test

Less time spent on writing tests

Developer Benefits

- **High complexity becomes manageable**
- **Agile testing**
  - Iterative
  - Test Driven Development
  - Extensive use of Continuous Integration
- **Open Source Software**

Management Benefits

- **Risk reduction**
  - Release at any time
  - High complexity becomes manageable
  - Iterative costs
- **Cost reduction**
  - easier to find cause of bugs
  - early detection of bugs and integration problems
- **Improved quality**
  - More and better testing

Summary

- OW Test Model is **AGILE!**
- **It works!**
- **Read more in community wiki**

http://wiki.community.objectware.no/display/smidigtonull/OW+Test+Model

Q & A

Resources

Enterprise Maven Infrastructure

http://wiki.community.objectware.no/display/smidigtonull/Enterprise+Maven+Infrastructure

Single-Responsibility Principle:

http://www.objectmentor.com/resources/articles/srp.pdf

Next Generation Java Testing: TestNG and Advanced Concepts:

http://www.amazon.com/Next-Generation-Java-Testing-Advanced/dp/0321503104

Manifesto for Agile Software Development: http://agilemanifesto.org/