Model – Based Testing
Using UML 2.0

Per Blysa
Telelogic AB
Model based testing using UML 2.0

Per Blysa
Telelogic
Telelogic At A Glance

- Founded: 1983
- HQ: Malmö, Sweden
- US HQ: Irvine, California
- Public Company: Listed in 1999
- Revenue 2004: $142 million / €118 million
- Development Sites: USA, Sweden, UK, India
"Doing the Right Things"
Plan, Prioritize, Visualize, Decide

"Doing Things Right"
Define, Design, Develop, Test, Deliver

Establish and Maintain Focus
Decision Support & Portfolio Management
Configuration & Change Management
Requirement Management

Modeling Business and Systems
Business Process Modeling & Enterprise Architecture
Systems and Software Modeling, Implementation, Test
“Doing the Right Things”
Plan, Prioritize, Visualize, Decide

“Doing Things Right”
Define, Design, Develop, Test, Deliver

Establish and Maintain Focus

Modeling Business and Systems

Focal Point

Systems Architect

SYNERGY

DOORS

TAU

Products

© Telelogic AB
Objectives

• Improving overall consistency and quality of the end product, as well as improving project communication, understanding and visibility
  – visualize, verify, validate and automate

• Introduce testing at a much earlier stage
  – not only when it comes to defining tests, but also to execute them

• Pull requirements, systems development and testing more tightly together
  – all expressed in the same language
UML 2.0

The most used diagrams

Class diagram
Composite structure diagram
Package diagram
Use case diagram
Object diagram
Component diagram
Deployment diagram
Sequence diagram
Interaction overview diagram
Timing diagram
Communication diagram
State machine diagram
Activity diagram
Model Driven Architecture (MDA)

- Requirements analysis
- System/Architecture analysis
- Design and implementation

CIM: Computation Independent Model
PIM: Platform Independent Model
PSM: Platform Specific Model

Profiles

Application

Transformations:
- CIM → PIM
- PIM → PSM
- PSM → Code

Traceability:
- CIM → PIM
- PIM → PSM
- PSM → Code

Code generation

Building & deploying
Model Driven Architecture (MDA)

- Much of the focus of MDA is put on transformations of different kinds, usually from platform independent models to platform specific models (and sometimes to code).
- There are two areas that are often glossed over when talking about MDA:
  - testing
  - requirements
Test development

• Tests must be developed just as other applications
  – unit tests, integration tests, and system tests
  – most likely, testing consumes most of your available time

• MDA does not negate the need for testing
  – increases how much can be automated
  – makes it possible to apply tests earlier in the lifecycle

• In addition, we must not only ensure that what we build is correct, but also that we are building the right thing
  – traceability to requirements
A common testing framework

• Given that MDA relies on modeling, we should have a standard way to express tests
  – that works with models at different levels of abstraction,
  – with code, and
  – also be possible to map to existing testing frameworks

• UML 2.0 Testing Profile, defined by the OMG
The UML 2.0 Testing Profile

• A UML 2.0 profile specified by the OMG
  – draws on experiences from test frameworks and test languages, such as JUnit and TTCN-3

• Adds concepts that are related to testing
  – test cases, test components, verdicts

• Provides a framework for designing, executing, and analyzing tests
  – test generation
  – primarily geared towards black box testing

• The profile can be applied for
  – unit testing
  – integration testing
  – system testing
  – regression testing
Terminology

- **SUT**: System Under Test
  - the (sub-)system, class or component that should be tested

- **Test case**
  - a behavior designed to test the SUT

- **Test component**
  - a realization of one or more test cases

- **Test context**
  - the container for the test cases, the SUT, and the test components
Describing tests in UML

• Tests can be created using any UML behavior
  – interactions
  – state machines
  – activities

• Typically, sequence diagrams are used when defining tests in UML
  – describe the interactions between test components and SUTs
  – based on stimuli and responses
Sequence diagrams

- Sequence diagrams have been significantly revised in UML 2.0
  - a major reason was to accommodate test descriptions
- Interaction uses
  - references to other interactions
- Combined fragments
  - alternative
  - iteration
  - optional
  - …and several other variations
Expressing timers & duration constraints

interaction ProcessCommsMessageTestCase2 {1/1}

tc : TestComponent1

ResponseTimer(5)

{lReqPosition::Init()}

{lCurrPosition::InitAck()}

ResponseTimer(5)

{lCommsIn::voiceAndDataCommsIn("Authorised_1", "London", dp.getChecksum(invalid), "Load")}

{<1.5}

{lCommsOut::nak()}
Defaults and wildcards

- Sometimes, you do not want to bother about messages that are not relevant for a test case – specify a default
- Defaults are typically used to specify exceptions to normal functionality, and may apply to
  - messages
  - test cases
  - test contexts
Data pools

• While testing, it is quite common that you want to run the same tests with different parameter values
  – such as boundary values and “typical values”
• The concept of a data pool is used to manage this situation
  – a collection of values or objects
  – a selector operation is used to extract values from the data pool

ResponseTimer(5)

ICommsIn::voiceAndDataCommsIn("Authorised_1", "London", dp.getChecksum(invalid), "Load")
Setting verdicts

- During a test case, there are different ways to affect the verdict of a test case
  - another message than the one that is expected in a test case is received → fail
  - setVerdict() is a call that can be used to explicitly set a verdict
Executing tests

- One of the main ideas of MDA is that it is possible to execute models.
- With this capability comes the possibility to drive the execution based on tests:
  - automated testing of generated code
  - automated testing of models
- As a result, problems can be detected and addressed much earlier in the development lifecycle.
A testing chain

1. Select SUT
2. Create test context
3. Add test cases
4. Build test application
5. Run test application
6. Evaluate test results

- Original model
- Test model
- Intermediate test model
- Test application
- Test Log
Analyzing the test results

• May be as simple as running all tests and evaluate the outcomes against the expected outcomes
  – daily (or nightly) test runs
  – based on a table
• Can also use the test cases to drive the system under test in simulation mode
  – “debug” the model or the code to find out what went wrong

<table>
<thead>
<tr>
<th>TestCase</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestCase1</td>
<td>pass</td>
</tr>
<tr>
<td>TestCase2</td>
<td>inconclusive</td>
</tr>
<tr>
<td>TestCase3</td>
<td>pass</td>
</tr>
<tr>
<td>TestCase4</td>
<td>fail</td>
</tr>
</tbody>
</table>
Reusing existing sequence diagrams as test cases

• Test cases can of course be created from scratch

• But, often there are existing model (fragments) that can be reused
  – a trace from a simulation
  – a requirement

• These can be automatically mapped into test case sequence diagrams
Regression testing

• Regression testing follows naturally when you can turn an execution trace into a test case
  – whenever you simulate or debug the application you can turn the trace into test case

• Allows you to retrace your steps anytime to make sure that nothing has been broken by later changes
  – should primarily be used when you know that the trace exhibits proper behavior (pass)
  – when such a test case fails, you either need to update the test case or correct whatever change was made to the system
Requirements and Traceability

User Requirements Specification

System Requirements Specification (Use Case Modeling)

Use Case Specification Modeling

Test Modeling

Refine, extend, reuse

Traceability
Summary

• The UML 2.0 Testing Profiles brings testing to the modeling level
  – increase productivity through automation
  – increase quality by eliminating error-prone processes
  – aid communication, collaboration and visibility
  – introduce testing at an earlier stage
  – enables the use of a common language throughout the development process
Thank you!