Testing Object-Oriented Systems: Lessons Learned

Robert V. Binder
RBSC Corporation    www.rbsc.com

June 15, 2000
Overview

• Lessons Learned
  – Design, automation, and process

• The State of the Art
  – Design, automation, and process

• The State of the Practice
  – Three levels
  – Testing can achieve world class quality
Lessons Learned

• Class/Cluster test design
  – Super/subclass interaction must be tested: test at flattened scope.
  – Design subclass test suites to re-run on superclasses.
  – Design superclass test suites to re-run on subclasses.
  – Test polymorphic servers for LSP compliance.
Lessons Learned

• Class/Cluster test design
  – Exercise each binding of a polymorphic server message
  – Test all parameters for generics
  – Test interface data flow of non-modal classes
Lessons Learned

• Subsystem/system test design
  – Control easily obscured or accidental
    • Complex dependencies between concrete state and message sequence
    • Hierarchic control in state-based subclasses
    • Mosaic modularity at larger scope
  – Model behavior with state machines; achieve transition cover or better
Lessons Learned

- Subsystem/system test design
  - Objects don’t compose (easily)
  - Producer’s framework should not be in consumer’s test scope
  - Minimum system/subsystem test includes
    - Testing exceptions
    - Testing class associations
    - Testing use cases (requires testable content)
Lessons Learned

• Test Automation
  – Encapsulation and mosaic modularity decrease controllability and observability
  – Design-by-contract/assertions is the only practical counter-measure for inherent non-determinism and loss of testability
Lessons Learned

• Test Automation
  – Avoid stubs: increase scope of the IUT or test in bottom-up order
  – Design test harness to exploit the structure and particulars of the system under test
  – Complete app = app components + test components under CM control
Lessons Learned

• Process
  – Inspect for omissions and inconsistencies, test for everything else
  – Design for testability
    • Implement hierarchic architecture patterns
    • Eliminate or encapsulate cyclic dependencies
    • Assert class invariants, at least
  – Support reuse with complementary producer/consumer testing strategies
Lessons Learned

• Process
  – 3 to 6 development increments
    • Developer class/cluster test -- Run tests locally, design tests globally: “unigration”
    • XP: “Continuous integration, relentless testing”
    • Independent build/integration group tests completed increment
    • Test suites must be regress-able
  – System testing on final increment
State of the Art

- Representation
- Design for Testability
- Test Design
- Test Automation
SOA: Representation

• Best Practices
  – Syntropy, Design by Contract
  – UML/OCL 1.0
  – Design Patterns

• Challenges
  – Architecture
  – Limits of cartoons
  – Test design as software engineering
SOA: Design for Testability

• Best Practices
  – Frameworks/libraries with assertions
  – Lakos’ levelizable architecture
  – Percolation pattern
  – OS/400 test framework
SOA: Design for Testability

• Challenges
  – Seamless language support
  – OO testability an oxymoron?
  – Entropy horizon about 24 months
SOA: Test Design

• Best Practices
  – Test design patterns

• Challenges
  – Intra-class coverage
  – Polymorphic paths
  – Validated failure metrics/fault models
## Test Design Pattern

- New pattern schema for test design

<table>
<thead>
<tr>
<th>Name/Intent</th>
<th>Context</th>
<th>Fault Model</th>
<th>Strategy</th>
<th>Entry Criteria</th>
<th>Exit Criteria</th>
<th>Consequences</th>
<th>Known Uses</th>
</tr>
</thead>
</table>

- **Test Model**
  - Test Procedure
  - Oracle
  - Automation

*Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley.
Test Design Patterns

• Method Scope
  – Category-Partition
  – Combinational Function
  – Recursive Function
  – Polymorphic Message

• Class/Cluster Scope
  – Invariant Boundaries
  – Modal Class
  – Quasi-Modal Class
  – Polymorphic Server
  – Modal Hierarchy
Test Design Patterns

- Subsystem Scope
  - Class Associations
  - Round-Trip Scenarios
  - Mode Machine
  - Controlled Exceptions

- Reusable Components
  - Abstract Class
  - Generic Class
  - New Framework
  - Popular Framework
Test Design Patterns

- Intra-class Integration
  - Small Pop
  - Alpha-Omega Cycle

- Integration Strategy
  - Big Bang
  - Bottom up
  - Top Down
  - Collaborations
  - Backbone
  - Layers
  - Client/Server
  - Distributed Services
  - High Frequency
Test Design Patterns

• System Scope
  – Extended Use Cases
  – Covered in CRUD
  – Allocate by Profile

• Regression Testing
  – Retest All
  – Retest Risky Use Cases
  – Retest Profile
  – Retest Changed Code
  – Retest Within Firewall
SOA: Test Automation

• Best Practices
  – Design patterns for test automation
  – Automatic driver generation
  – Simple coverage analyzers
SOA: Test Harness Patterns

- Test Case Implementation
  - Test Case/Test Suite Method
  - Test Case/Test Suite Class
  - Catch All Exceptions

- Test Control
  - Server Stub
  - Server Proxy

- Test Drivers
  - TestDriver Super Class
  - Percolate the Object Under Test
  - Symmetric Driver
  - Subclass Driver
  - Private Access Driver
  - Test Control Interface
  - Drone
  - Built-in Test Driver
SOA: Test Harness Patterns

- **Test Execution**
  - Command Line Test Bundle
  - Incremental Testing Framework (e.g. JUnit)
  - Fresh Objects

- **Built-in Test**
  - Coherence idiom
  - Percolation
  - Built-in Test Driver
SOA: Test Automation

• Challenges
  – Validated failure metrics/fault models
  – Tool capability gaps
    • No support for OO-specific coverage
    • Very weak specification-based test generation
    • Weak support for test harness generation
State of the Practice

• Best Practices
  – Testing by scope (about 10%)
  – Many embedded/real-time shops
  – Extreme Programming

• Challenges
  – High-frequency/short cycle development
  – Naïve test design
  – Tool capability gaps
SOP: Testing by Poking Around

- About 70% of all organizations
- Characteristics
  - Testing done at developer discretion
  - No test entry/exit criteria
  - High tolerance for low quality
SOP: Testing by Poking Around

• Improvement Strategy
  – Assess limits of improvability
  – Train developers in basic test design
  – Install basic tool set:
    • Coverage analyzer
    • Memory leak detector
    • Test harness framework/generator (e.g. Junit)
SOP: Testing by Use Cases

- About 20% of all organizations
- Complies with “Unified Process” test approach
- Characteristics
  - Assumes objects “just work”
  - System test from use cases
  - Frustrated with chronic bugginess
SOP: Testing by Use Cases

• Improvement Strategy
  – Achieve exit criteria for indicated class/cluster test patterns
  – Use appropriate component/subsystem test design patterns.
  – Develop testable use cases
  – Implement test automation to support regression testing
**SOP: Testing by Scope**

- About one in ten
- Characteristics
  - Test design corresponds to scope
  - Scope-specific test entry/exit criteria
  - Appropriate testing at all scopes
  - Effective test automation
  - Stable, repeatable process
SOP: Testing by Scope

- Improvement Strategy
  - Internal test design pattern-mining
  - Design for testability
  - Advanced test automation
  - Quantified closed loop feedback
Best Practice Examples

• Stepstone Corporation

• Ericsson CEE Project

• *Testing was the primary quality technique*
Stepstone Corporation

• ICpack 201 -- Objective-C class library
• Inspections for all classes
• Extensive automated test harness developed for each complex class
• No systematic test design
Ericsson CEE

- 75 KLOC C++ cellular support application
- Systematic testing at class, cluster, and system scope
- No other verification techniques used
Achieving World Class OO Quality

• Best-in-Class level:
  – An average of “less than 0.025 user-reported defects per function point” in the first year after release

• World Class = 10x Best in Class

## Achieving World Class OO Quality

<table>
<thead>
<tr>
<th>Organization/Language</th>
<th>KLOC</th>
<th>FP</th>
<th>Major Post Release Bugs</th>
<th>Bugs/FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stepstone Objective-C</td>
<td>12</td>
<td>414</td>
<td>5</td>
<td>0.0121</td>
</tr>
<tr>
<td>Ericsson C++</td>
<td>75</td>
<td>1364</td>
<td>7</td>
<td>0.0051</td>
</tr>
</tbody>
</table>
Summary

- Lessons learned: OO testing requires unique test design approaches
- State of the art: expressed in patterns
- State of the practice: world class quality can be achieved through testing