TESTABILITY: FACTORS AND STRATEGY

Robert V. Binder
rvbinder@gmail.com

Google Test Automation Conference
Hyderabad
October 28, 2010
Overview

• Why testability matters
• White box testability
• Black box testability
• Test automation
• Strategy
• Q&A
WHY TESTABILITY MATTERS
Testability Economics

- Assumptions
  - Sooner is better
  - Escapes are bad
  - Fewer tests means more escapes
  - Fixed budget – how to optimize

- Efficiency: average tests per unit of effort
- Effectiveness: average probability of killing a bug per unit of effort
- Higher testability: more better tests, same cost
- Lower testability: fewer weaker tests, same cost

Testability defines the limits on producing complex systems with an acceptable risk of costly or dangerous defects.
What Makes an SUT Testable?

- **Controllability**
  - What do we have to do to run a test case?
  - *How hard (expensive) is it to do this?*
  - *Does the SUT make it impractical to run some kinds of tests?*
  - *Given a testing goal, do we know enough to produce an adequate test suite?*
  - *How much tooling can we afford?*

- **Observability**
  - What do we have to do to determine test pass/fail?
  - *How hard (expensive) is it to do this?*
  - *Does the SUT make it impractical to fully determine test results?*
  - *Do we know enough to decide pass/fail/DNF?*
Testability Factors

- Traceability
- Representation
- Requirements
- Specification
- Objectives
- Feasible
- Quantifiable
- IEEE 830

- Configuration Management
- Control Strategy
- Collaboration Packaging
- Architectural Packaging

- User Interface
- Fault Sensitivity
- Performance Tweaks

- Implementation
- Standard Mechanism
- Consistent Usage

- Encapsulation
- Polymorphism
- Inheritance

- Structure
- Complexity

- Memory Management
- External API
- Control Flow

- Time Sensitivity
- Shared Resources

- External Interface
- Determinism

- Test Suite
- User Interface
- Pre-Conditions

- Test Cases
- Post-Conditions

- Interoperability

- Test Design
- Test Execution Log
- Test Result Schema

- Test Procedures
- Test Summary

- Test Suite Management
- Static Code Analyzer

- Test Generator
- Specification-based Test Generator

- Developer Review
- Test Plan

- Built-in Test
- Vertical Integration

- Oracle
- Feasible
- Specification-based

- Reusable
- Tool Based

- Configuration Management Control

- Runtime Trace
- Comparator

- Incident Tracker

- Reporting

- Debug

- Test Bed

- Test Tools
- Initialize
- Input Capture

- Developer Definition
- Code-based Test Generator

- Test Data Generator

- Process Capability
- Consistency of Purpose

- Effectiveness
- Defined and Repeatable

- Customer-oriented

- Test Readiness Assessment
- Adequacy Assessment

- Closed Loop Feedback

- Test Case Development

- Test Rediness Assessment
- Empowerment

- Accountability

- Experience
- Motivation

- Staff Capability

- Design
- Code

- V&V Integration

- Test
- Maintain

- Prototyping

- Inspections

- Reviews

- Integrated Test Strategy

- Vertical Integration

- Rqmts
- Design

- Class Test
- Cluster Test

- Training

- Motivation

- Staff Capability
Testability Factors

Test Suite  Test Tools  Test Process  Testability

Representation  Implementation  Built-in Test
### Examples

<table>
<thead>
<tr>
<th></th>
<th>Controllability</th>
<th>Observability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>widgets, specialized widgets.</td>
<td>Proprietary lock-outs.</td>
</tr>
<tr>
<td><strong>OS Exceptions</strong></td>
<td>100s of OS exceptions to catch, hard to trigger.</td>
<td>Silent failures</td>
</tr>
<tr>
<td><strong>Objective-C</strong></td>
<td>Test drivers can’t anticipate objects defined on the fly</td>
<td>Instrumenting objects on the fly</td>
</tr>
<tr>
<td><strong>Mocks with DBMS</strong></td>
<td>Too rich to mock</td>
<td>DBMS has better logging</td>
</tr>
<tr>
<td><strong>Multi-tier CORBA</strong></td>
<td>Getting all DOs desired state</td>
<td>Tracing message propagation</td>
</tr>
<tr>
<td><strong>Cellular Base Station</strong></td>
<td>RF loading/propagation for varying base station population</td>
<td>Proprietary lock-outs. Never “off-line”</td>
</tr>
</tbody>
</table>
WHITE BOX TESTABILITY
White Box Testability

Complexity,
Non-Deterministic Dependency (NDD)

PCOs
Built-in Test,
State Helpers,
Good Structure
The Anatomy of a Successful Test

- To reveal a bug, a test must
  - Reach the buggy code
  - Trigger the bug
  - Propagate the incorrect result
  - Incorrect result must be observed
  - The incorrect result must be recognized as such

```c
int scale(int j) {  // should be j = j + 1
    j = j - 1;
    j = j/30000;
    return j;
}
```

Out of 65,536 possible values for j, only six will produce an incorrect result: -30001, -30000, -1, 0, 29999, and 30000.
What Diminishes Testability?

- Non-deterministic Dependencies
  - Race conditions
  - Message latency
  - Threading
  - CRUD shared and unprotected data
What Diminishes Testability?

- Complexity
  - Essential
  - Accidental
What Improves Testability?

- Points of Control and Observation (PCOs)
- State test helpers
- Built-in Test
- Well-structured code
PCOs

- Point of Control or Observation
  - Abstraction of any kind of interface
  - See TTCN-3 for a complete discussion
- What does a tester/test harness have to do activate SUT components and aspects?
  - Components easier
  - Aspects more interesting, but typically not directly controllable
- What does a tester/test harness have to do inspect SUT state or traces to evaluate a test?
  - Traces easier, but often not sufficient or noisy
  - Embedded state observers effective, but expensive or polluting
  - Aspects often critical, but typically not directly observable
- Design for testability:
  - Determine requirements for aspect-oriented PCOs
State Test Helpers

- Set state
- Get current state
- Logical State Invariant Functions (LSIFs)
- Reset
- All actions controllable
- All events observable
Implementation and Test Models
Test Plan and Test Size

- **$K$ events**
- **$N$ states**
- With LSIFs
  - $KN$ tests
- No LSIFs
  - $K \times N^3$ tests

```
1 ThreePlayerGame()
2 p1_Start()
3 p2_Start()
4 p3_Start()
5 p1_WinsVolley()
6 p1_WinsVolley() [this.p1_Score() < 20]
7 p1_WinsVolley() [this.p1_Score() == 20]
8 p2_WinsVolley()
9 p2_WinsVolley() [this.p2_Score() < 20]
10 p2_WinsVolley() [this.p2_Score() == 20]
11 p3_WinsVolley()
12 p3_WinsVolley() [this.p3_Score() < 20]
13 p3_WinsVolley() [this.p3_Score() == 20]
14 p1_IsWinner()
15 p2_IsWinner()
16 p3_IsWinner()
17 ~()
```
Built-in Test

- Asserts
- Logging
- Design by Contract
- No Code Left Behind pattern
- Percolation pattern
No Code Left Behind

• Forces
  • How to have extensive BIT without code bloat?
  • How to have consistent, controllable
    • Logging
    • Frequency of evaluation
    • Options for handling detection
    • Define once, reuse globally?

• Solution
  • Dev adds Invariant function to each class
  • Invariant calls InvariantCheck evaluates necessary state
  • InvariantCheck function with global settings
  • Selective run time evaluation
  • const and inline idiom for no object code in production
Percolation Pattern

- Enforces Liskov Subsitutability
- Implement with No Code Left Behind
Well-structured Code

• Many well-established principles

• Significant for Testability
  • No cyclic dependencies
  • Don’t allow build dependencies to leak over structural and functional boundaries (levelization)
  • Partition classes and packages to minimize interface complexity
BLACK BOX TESTABILITY
Black Box Testability

Size, Nodes, Variants, Weather

Test Model, Oracle, Automation
System Size

• How big is the essential system?
  • Feature points
    • Use cases
    • Singly invocable menu items
    • Command/sub commands
  • Computational strategy
    • Transaction processing
    • Simulation
    • Video games
  • Storage
    • Number of tables/views
Network Extent

- How many independent nodes?
  - Client/server
  - Tiers
  - Peer to peer
- Minimum Spanning Tree
  - Must have one at least of each online

MS-TPSO, Transaction Processing System Overview
Variants

• How many configuration options?
• How many platforms supported?
• How many localization variants
• How many “editions”? 

• Combinational coverage
  • Try each option with every other at least once (pair-wise)
  • Pair-wise worst case product of option sizes
  • May be reduced with good tools
Weather – Environmental

• The SUT has elements that cannot be practically established in the lab
  • Cellular base station
  • Expensive server farm
  • Competitor/aggressor capabilities
• Internet – you can never step in the same river twice
• Test environment not feasible
  • Must use production/field system for test
  • Cannot stress production/field system
More is Less

- Other things being equal, a larger system is less testable
  - Same budget spread thinner
  - Reducing system scope improves testability

- Try to partition into independent subsystems: additive versus multiplicative extent of test suite

- For example
  - 10000 Feature Points
  - 6 Network Node types
  - 20 options, five variables each
  - Must test when financial markets are open

- How big is that?
95,000 Light Years
1 Pixel ≈ 400 Light Years
Solar System ≈ 0.0025 LY
Understanding Improves Testability

- Primary source of system knowledge
  - Documented, validated?
  - Intuited, guessed?
  - IEEE 830

- Test Model
  - Different strokes
  - Test-ready or hints?

- Oracle
  - Computable or judgment?
  - Indeterminate?
TEST AUTOMATION
Automation Improves Testability

• Bigger systems need many more tests
• Intellectual leverage
  • Huge coverage space
  • Repeatable
  • Scale up functional suites for load test
• What kind of automation?
  • Harness
    • Developer harness
    • System harness
    • Capture-replay
  • Model-based
    • Generation
    • Evaluation
Test Automation Envelope

Reliability (Effectiveness)

- 5 Nines
- 4 Nines
- 3 Nines
- 2 Nines
- 1 Nine

Productivity: Tests/Hour (Efficiency)

- Manual
- Scripting
- Bespoke Model-Based
- Model-Based Vision
STRATEGY
How to Improve Testability?

WBT = BIT + State Helpers + PCOs

- Structure - Complexity - NDDs

maximize

minimize

BBT = Model + Oracle + Harness

- Size - Nodes - Variants - Weather

maximize

minimize
## Who Owns Testability Drivers?

<table>
<thead>
<tr>
<th></th>
<th>Architects</th>
<th>Devs</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBT Model</td>
<td></td>
<td></td>
<td>★</td>
</tr>
<tr>
<td>Oracle</td>
<td></td>
<td></td>
<td>★</td>
</tr>
<tr>
<td>Harness</td>
<td></td>
<td></td>
<td>★</td>
</tr>
<tr>
<td>PCOs</td>
<td></td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variants</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weather</td>
<td>★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WBT BIT</td>
<td></td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>State Helpers</td>
<td></td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>Good Structure</td>
<td></td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td></td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>NDDs</td>
<td></td>
<td>★</td>
<td></td>
</tr>
</tbody>
</table>

Testers typically don’t own or control the work that drives testability.
Strategy

<table>
<thead>
<tr>
<th>High Testability</th>
<th>Emphasize Functional Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Testability</td>
<td>Manage Expectations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High Testability</th>
<th>Incentivize Repeaters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Testability</td>
<td>Emphasize Code Coverage</td>
</tr>
</tbody>
</table>

White Box Testability

Low

High
Q & A
Media Credits

- Diomidis D. Spinellis. *Internet Explorer Call Tree*.
- Unknown. *Dancing Hamsters*.
- Jackson Pollock, *Autumn Rhythms*.
- Brian Ferneyhough. *Plötzlichkeit* for Large Orchestra.
- Microsoft, *MS-TPSO, Transaction Processing System Overview*.
- Holst - The Planets – Uranus. Peabody Concert Orchestra, Hajime Teri Murai, Music Director
- Test Automation Envelope. *mVerify Corporation*. 