SEI Observations and Reference Model for Software Integration Labs

Robert V. Binder
July 11, 2018
Notices

Copyright 2018 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.


DM18-0843
Overview

- What should a Software Integration Lab (SIL) do?
- SIL Reference Model
- SIL configurations observed at SEI customers
- Automotive best practices
- Test automation levels and effectiveness
- Testing productivity versus effectiveness
US Army SIL for M153 CROWS System

Images from the web site – SEI did not visit this lab.
SIL Reference Model: the System Under Test and its Environment
What should a Software Integration Lab do?

SIL Goals
Evaluate interoperability and stability of:
- System Under Test (SUT) software items (SI)
- SUT and run-time stack interaction
- SUT and sensors, actuators, peripherals
- SUT and external systems
Support testing of partial SUT configurations, including falsework
Achieve realistic environmental conditions
Check completeness with respect to requirements and architecture
Support development, QT, DT, and OT
Support rapid cycle Devops
What should a Software Integration Lab do?

SIL Non-goals (typical)

White-box software evaluation (maintainability, structure, etc.)
Comprehensive functional testing of SI, OS, or HW
Comprehensive functional testing of SUT
Comprehensive failure/restart/recovery testing
SUT reliability or performance test (MTBF, response time, utilization, etc.)
SUT usability or effectiveness testing, user documentation evaluation
Long duration soak test
Safety testing
Testing physical aspects of mechanical, electrical, or RF components
What should a Software Integration Lab do?

SIL Strategy: Testing System

The **Testing System** is a software-defined environment purpose-built to achieve testing goals for the SUT and its environment.

It should be funded, developed, staffed, and managed as a first class sustainment asset.

**Maximal test automation**

- Test asset management system, all test code under CM control
- Model-based test generation
- Test execution system(s)
- Test objects drive adapter objects that drive [falsework](#) and real SUT interfaces
- User-interface test suites follow Feature-action-control pattern
- Seamless interleaving of manual and traditional test code/procedures
What should a Software Integration Lab do?

SIL Strategy: Test Approach

Design realistic test scenarios to achieve interaction coverage
- Exercise all modes (normal and failure), mode transitions, and duty cycles
- Exercise at least one failure of each sensed/managed mechanical, electrical, or RF interface
- Verify datastore integrity at entry/exit of each mode

Calibrate test artifacts
- Appropriate level of rigor for test artifacts
- Test artifacts should be reusable
- Test artifacts must be maintainable

Living antecedent traceability
What should a Software Integration Lab do?

SIL Strategy: Test Coverage

Test at least once:
- Every externally triggered interaction
- Every internally triggered interaction (e.g., timer)
- Every requirement for an interaction and its observable effects
- Each mode and transition, including failure modes

Evaluate interaction coverage (end-to-end paths)

Test at least pair-wise combinations of inputs, configurations, settings, etc.

Don’t rely on stale regression testing
What should a Software Integration Lab do?

SIL Strategy: Testing System’s Network

Testing System’s network is isolated from SUT network

SUT network(s) provide passive “Tee” for injection and monitoring

Configuration-as-Code and containerization stage both Testing System and SUT

Staging the Testing System for a classified SUT

• Testing System development
  - Development impractical without public internet connectivity
  - Testing System developed in unclassified environment
  - Falsework allows tests to run in unclassified environment
  - One-way data diode or air gap staging to classified Testing System
  - Install Testing System container in classified environment
• SUT container installed into classified environment and tested
What should a Software Integration Lab do?

SIL Strategy: Process

Produce specific, measured, actionable, realistic, and timely evaluation results

Follow quality management standard ISO/IEC 17025 *General requirements for the competence of testing and calibration laboratories*

Support upstream and downstream activities

- Provide design-for-testability guidance and entry requirements to suppliers and developers
- Operate a SIL instance dedicated to developer continuous integration (CI)
- Gate incoming candidates: Accept new SUT version only after upstream CI passes, smoke test passes; test readiness review acceptance
- Continuously evaluate and improve downstream handoff
- Track all integration bug reports; use to evaluate/improve test effectiveness
SIL Reference Model: the System Under Test and its Environment

- **SUT**
  - User Interface
  - Software Item
  - Run Time OS
  - Comp Hardware
  - Actuators
  - Sensors

- **Environment**
  - SUT Users
  - External Users
  - Friendly Actors
  - Hostile Actors
  - External System
  - Plant
Adapters decouple test objects from physical interfaces so same test suites can run on multiple SUT/SIL configurations.
SIL Reference Model: Devops for the Testing System

- **Testing System Release Candidate**
- **Internet**
- **Staging**
- **TS**
- **SUT**
- **Developer SIL Instance**
- **Continuous Integration Tool Chain**
- **SUT Release Candidate SIL Instance**
- **Staging**
- **SUT**
- **Secure SIL Instance**
- **Data Diode**

- **SUT**
- **TS**
- **Staging**
Profiled Software Testing Labs

Composite of multiple SEI engagements

- 25 testing labs
- Data collected for different projects by interviews and visits
- ~80% developing or sustaining weapon systems
- ~20% developing or sustaining enterprise systems
- Includes experience of standing up a SIL at SEI for the SOCOM TALOS program

Notable

- Some SILs also used for training
- Almost no effective shift-left
- Many challenges for development and validation of simulation falsework
- Upstream testing often superficial
- No explicit design-for-testability
- High friction moving unclass to class
- Automated SILs have highest defect removal efficiency
Profiled Lab Characteristics

**App Type**
- Embedded
- C2
- EIS
- ISR

**Stage**
- Developer
  - QT
  - DT
  - OT

**Scope**
- SI
- LRU
- System

**Configuration**
- CI/CD?
- Component
- Environment
- Sim?
- HIL?

**Effectiveness**
- Test Run Days
- Number of Tests
- % Automated

---

SEI Observations and Reference Model for Software Integration Labs
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A]
This material has been approved for public release and unlimited distribution
Profiled SILs support mostly embedded apps …

… during QT and DT
Configuration and Automation

1/3 use automated regression suites, 1/3 have no automation

1/4 use both Sim + HIL for LRU integration
## Test Automation Levels

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main Benefits</strong></td>
<td>Usability</td>
<td>Rqmt checkoff</td>
<td>Rqmt checkoff</td>
<td>Rqmt checkoff</td>
<td>Maintain model</td>
</tr>
<tr>
<td></td>
<td>Omissions</td>
<td>Repeatable</td>
<td>Repeatable</td>
<td>Repeatable</td>
<td>Generate tests</td>
</tr>
<tr>
<td></td>
<td>Lowest cost</td>
<td>Low cost</td>
<td>Run on demand</td>
<td>Run on demand</td>
<td>Deep coverage</td>
</tr>
<tr>
<td><strong>Concerns</strong></td>
<td>Coverage?</td>
<td>Superficial</td>
<td>Superficial</td>
<td>Superficial</td>
<td>Staffing/Skilling</td>
</tr>
<tr>
<td></td>
<td>No repeat</td>
<td>Long test time</td>
<td>Brittle</td>
<td>SW eng stds</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inconsistent</td>
<td></td>
<td>High maint $</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Risk Reduction</strong></td>
<td>Low</td>
<td>Low-Med</td>
<td>Low-Med</td>
<td>Low-Med</td>
<td>Med-High</td>
</tr>
<tr>
<td><strong>Tech Baseline</strong></td>
<td>None</td>
<td>Some</td>
<td>Some</td>
<td>Some</td>
<td>Deep</td>
</tr>
<tr>
<td><strong>Ownership via</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Test Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Layered, model-based test automation achieves highest ROI and effectiveness.
Automation Levels at Profiled SILs

Good news: All labs follow regular testing process; none ad hoc or skip

- Naïve Automation: 52%
- Layered Automation: 12%
- Model-based Automation: 4%
- No Automation: 32%
## Profiled Labs Productivity Comparison

### Manual Interaction

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU Simulated or Real Hardware-in-the-Loop</td>
<td>150</td>
<td></td>
<td>2200</td>
</tr>
<tr>
<td>LRU Simulated</td>
<td>390</td>
<td>100, 100</td>
<td>500</td>
</tr>
<tr>
<td>Deployed System (all Real)</td>
<td>50</td>
<td>50</td>
<td>200</td>
</tr>
</tbody>
</table>

*Estimated average test points evaluated per day during a test cycle*
## Best Practices: Testing Strategies

<table>
<thead>
<tr>
<th></th>
<th>Profiled Labs Best Practices</th>
<th>Automotive Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage: every requirement at least once</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Model-based Test Generation</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Test Asset Management</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Profile-based Reliability Testing</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Combinatorial Design</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Data-driven promotion/acceptance</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## Best Practices: Testing System Configuration

<table>
<thead>
<tr>
<th></th>
<th>Profiled Labs Best Practices</th>
<th>Automotive Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI build with controllable adapters</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Continuous Integration with LRU simulation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LRU simulation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LRU simulation + Real LRU</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Testable full-up chassis</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Manual testing on system</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Manual testing on system, telemetry/capture</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Field monitoring</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
# Best Practices: Test Execution

<table>
<thead>
<tr>
<th></th>
<th>Profiled Labs Best Practices</th>
<th>Automotive Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated component simulation/emulation</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Automated test harness</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>User Interface automation</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Network traffic monitoring</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Controllable fault injection</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Backup
“Falsework consists of temporary structures used in construction to support spanning or arched structures in order to hold the component in place until its construction is sufficiently advanced to support itself.”
### Falsework for a SIL

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Role in a SIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>SW May be part of SUT. Limits coverage, placeholder for unavailable interface.</td>
</tr>
<tr>
<td>Mock</td>
<td>SW May be part of SUT. Limits coverage, placeholder for unavailable interface.</td>
</tr>
<tr>
<td>Simulation</td>
<td>SW Use a simulator to generate input or accept output.</td>
</tr>
<tr>
<td>Emulation</td>
<td>SW/ HW Use a HIL emulator to generate actuator input or sensor output using high speed digital devices.</td>
</tr>
</tbody>
</table>

In software testing, falsework refers to stubs, mocks, fakes, “Service Virtualization,” generated or programmed simulations, and high fidelity hardware-based emulation.
Interface Coverage Matrix

Each cell represents a possible interaction between actor types

- All cells are candidates because any cell can initiate an interaction
- Table shows a minimal subset for a typical system
- Some systems may have only a few interactions; some have all

<table>
<thead>
<tr>
<th></th>
<th>SI</th>
<th>RTS</th>
<th>HW</th>
<th>XS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RTS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>XS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The structure of the system must be mapped to be sure interactions are covered. Some good sources are:

- Interface Control Documents (ICDs)
- Sequence diagrams
- Code analyzers like SciTools Understand
- Runtime logs and traffic
- User documentation
There are many hundreds of COTS, FOSS, and GOTS software testing tools
Testing scope, tooling, focus, lanes

<table>
<thead>
<tr>
<th>Scope</th>
<th>Example Tool</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer, Unit, SI</td>
<td>Junit, SonarCube</td>
<td>Functions</td>
</tr>
<tr>
<td>Component, SI, Subsystem</td>
<td>Selenium, SOAP UI</td>
<td>Features, User stories</td>
</tr>
<tr>
<td>System, SoS</td>
<td>TestStand, Jmeter</td>
<td>Use cases, Performance, Mission threads</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CTR</th>
<th>PO</th>
<th>FAT</th>
<th>DTO</th>
<th>OTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Which testing tool(s) is right for your job(s)?

Tools are specialized for:

- Testing purpose
- Target interface, IDE, programming language
- Application domain: transaction processing, embedded, mobile app...
- Runtime stack(s) of target and tool
Test Automation Reference Architecture: Java/Cloud stack

Requirements Management
Model-based Systems Engineering

BDD/ATDD Support
Combinatorial Design
Model-based Testing

Dev Test Framework
Code Coverage Analyzer
Static Analyzer
Mutation Testing
Fault Injection

Test Runner
Web UI Harness
Dot Net UI Harness
Smartphone Harness
SOA/API Harness

Load Generation
Performance Monitor
Network Capture

Bug Tracking
Test Asset Management
Application Life Cycle Management

Configuration Management
Continuous Integration
Continuous Deployment

Maven
Selenium
Ranorex
Appium
TestMaker

Silk Performer
Nagios
WireShark

FalseWork

Virtualize

Representative tools are not recommendations. Many others for each slot

JBehave
ACTS
Smartesting

JUnit
Klocwork
Fortify
Pit
Byteman

Italics indicate advanced capability

Company, product, and service names used in this slide are for illustrative purposes only. All trademarks and registered trademarks are the property of their respective owners.
Test Automation Reference Architecture: C++/RTOS stack

Company, product, and service names used in this slide are for illustrative purposes only. All trademarks and registered trademarks are the property of their respective owners.

Representative tools are not recommendations. Many others for each slot.

 Italics indicate advanced capability.
## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>CM</td>
<td>Configuration Management</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial off the Shelf</td>
</tr>
<tr>
<td>CPU</td>
<td>Computer Processor Unit</td>
</tr>
<tr>
<td>DT</td>
<td>Developmental Testing</td>
</tr>
<tr>
<td>DTO</td>
<td>Developmental Testing Organization</td>
</tr>
<tr>
<td>FOSS</td>
<td>Free open source software</td>
</tr>
<tr>
<td>FQT</td>
<td>Factory Qualification Test</td>
</tr>
<tr>
<td>GOTS</td>
<td>Government off the shelf</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>HIL</td>
<td>Hardware in the Loop</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>ICD</td>
<td>Interface Control Document</td>
</tr>
<tr>
<td>IDE</td>
<td>Interactive Development Environment</td>
</tr>
<tr>
<td>LRU</td>
<td>Line Replaceable Unit</td>
</tr>
<tr>
<td>MBT</td>
<td>Model-based Testing</td>
</tr>
<tr>
<td>MTBF</td>
<td>Mean time between failures</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OT</td>
<td>Operational Testing</td>
</tr>
<tr>
<td>OTO</td>
<td>Operational Testing Organization</td>
</tr>
<tr>
<td>QT</td>
<td>Qualification Testing</td>
</tr>
<tr>
<td>RCA</td>
<td>Root Cause Analysis</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-time operating system</td>
</tr>
<tr>
<td>SI</td>
<td>Software Item</td>
</tr>
<tr>
<td>SIL</td>
<td>Software Integration Lab</td>
</tr>
<tr>
<td>SoS</td>
<td>System of Systems</td>
</tr>
<tr>
<td>SQT</td>
<td>System/Software Qualification Test</td>
</tr>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TS</td>
<td>Testing System</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>GOTS</td>
<td>Government off the shelf</td>
</tr>
</tbody>
</table>
Contact Information

Organization
Carnegie Mellon University
Software Engineering Institute
Client Technical Services Division

Robert Binder
rvbinder@sei.cmu.edu
+1 412-268-1549

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612, USA

Web
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257