SEI Observations and Reference Model for Software Integration Labs

Robert V. Binder

July 11, 2018

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Notices

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.

DM18-0843

Overview

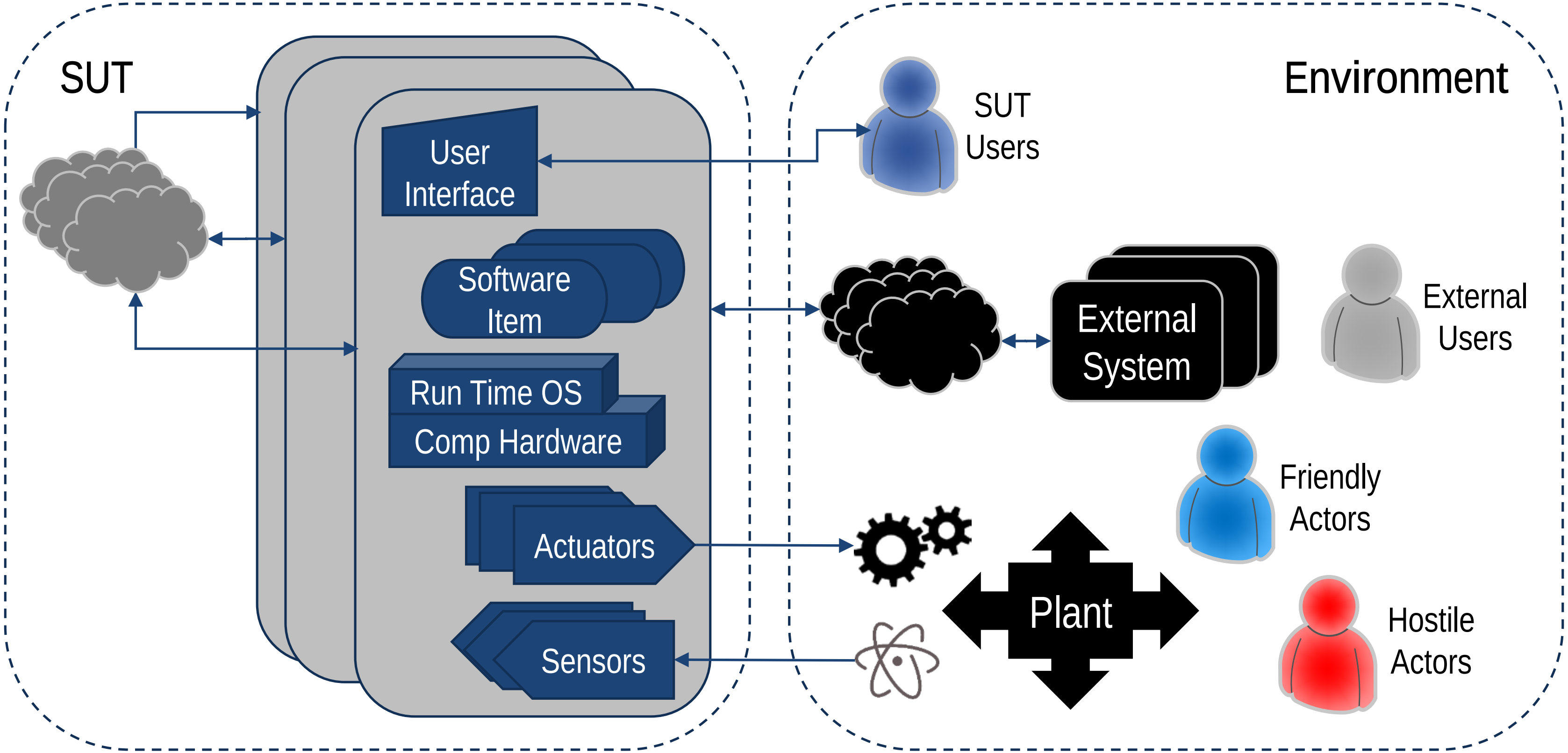
- **What should a Software Integration Lab (SIL) do?**
- **SIL Reference Model**
- **SIL configurations observed at SEI customers**
- **Automotive best practices**
- **Test automation levels and effectiveness**
- **Testing productivity versus effectiveness**

US Army SIL for M153 CROWS System

http://www.ardec.army.mil/armamentsec/facilities/crows.aspx

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

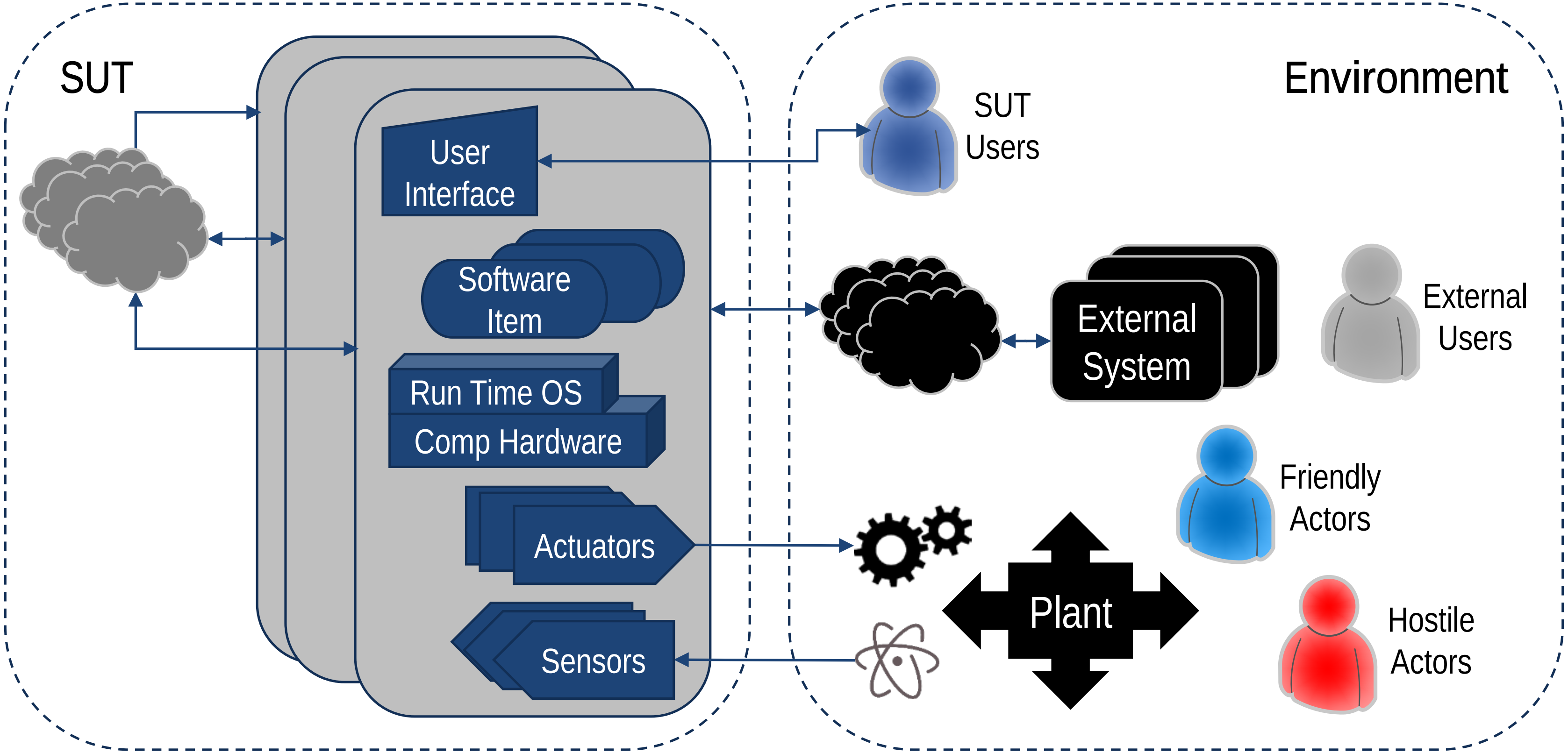
Produce specific, measured, actionable, realistic, and timely evaluation results

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

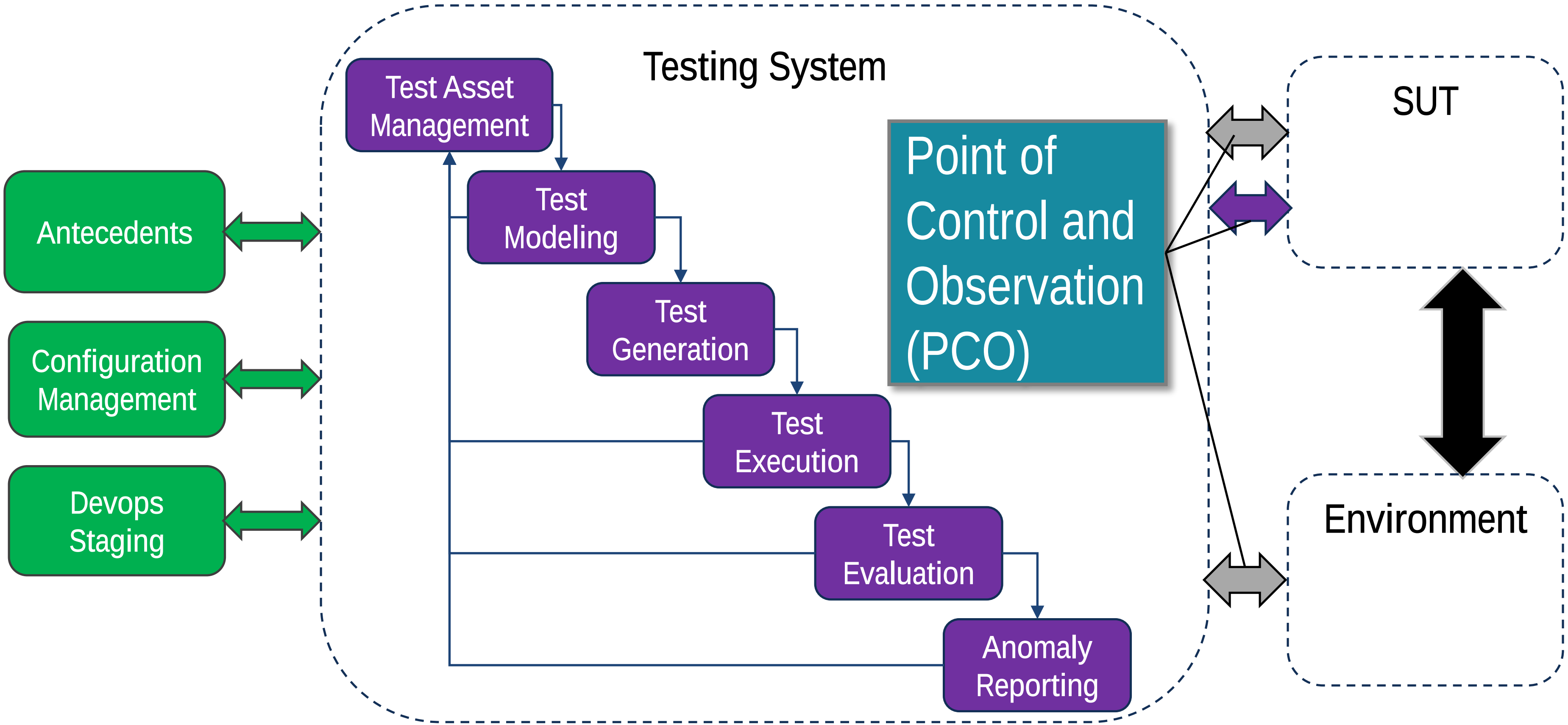
Support upstream and downstream activities

- 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

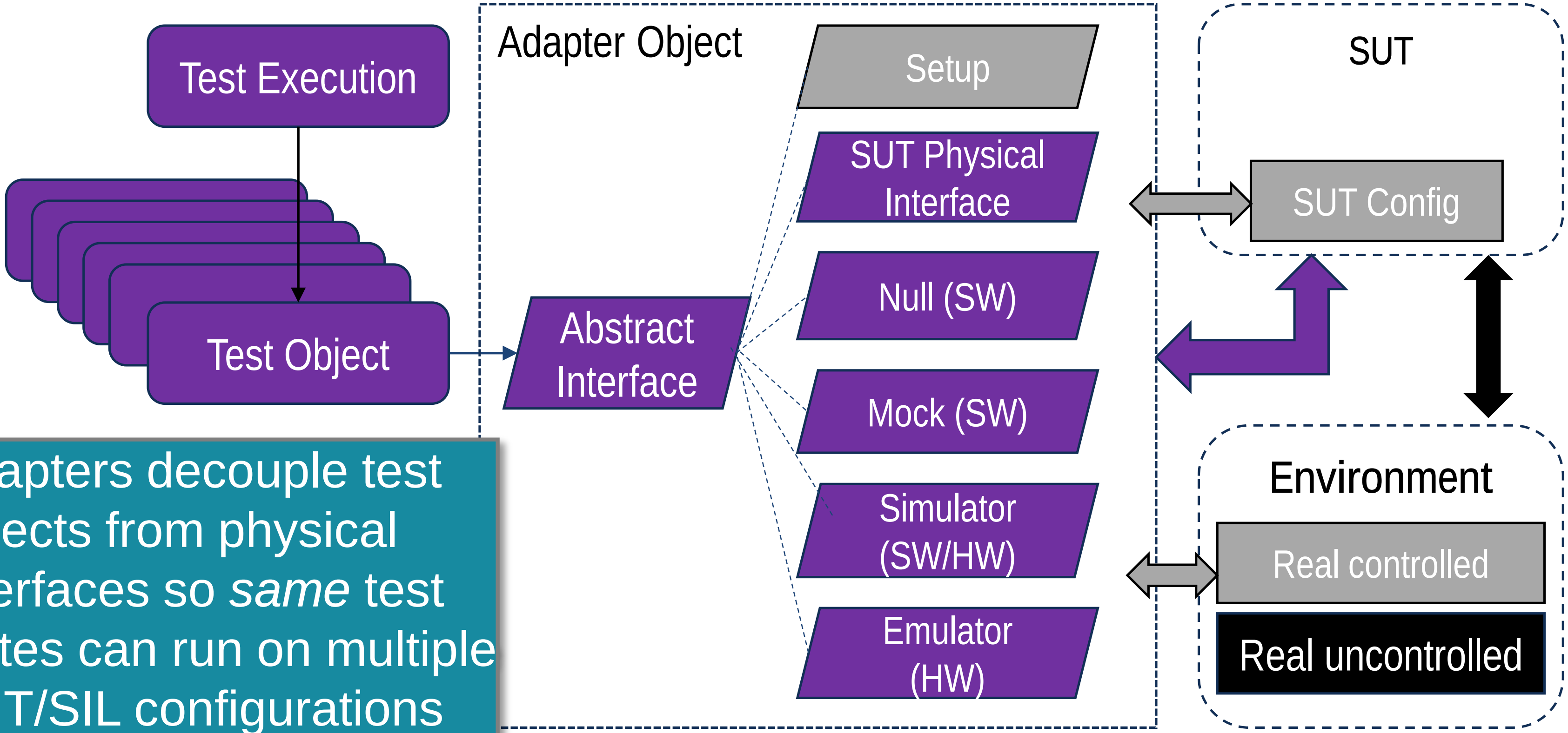
SIL Reference Model: the System Under Test and its Environment



SIL Reference Model: Testing System Architecture

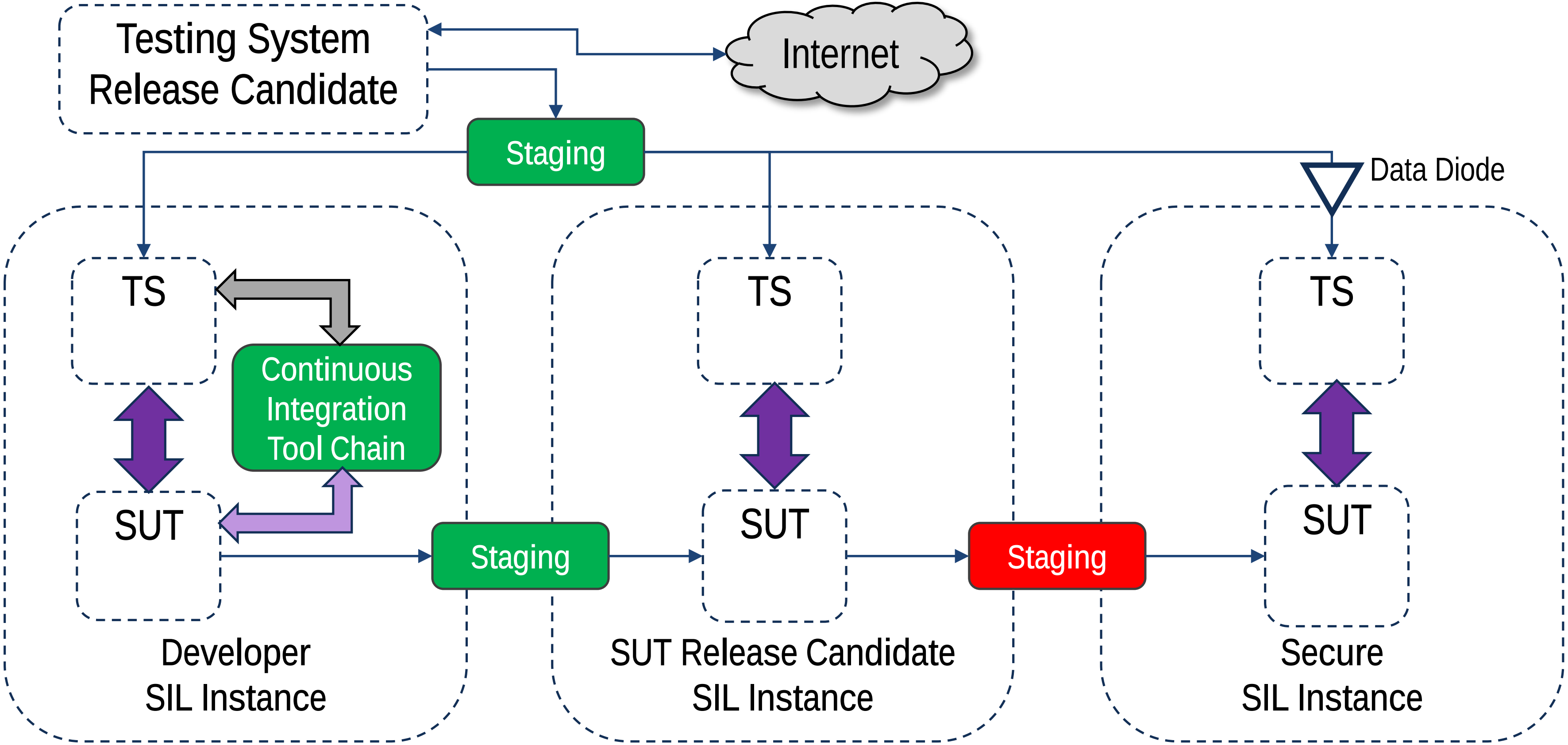


SIL Reference Model: Adapter Framework for all PCOs



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



Profiled Software Testing Labs

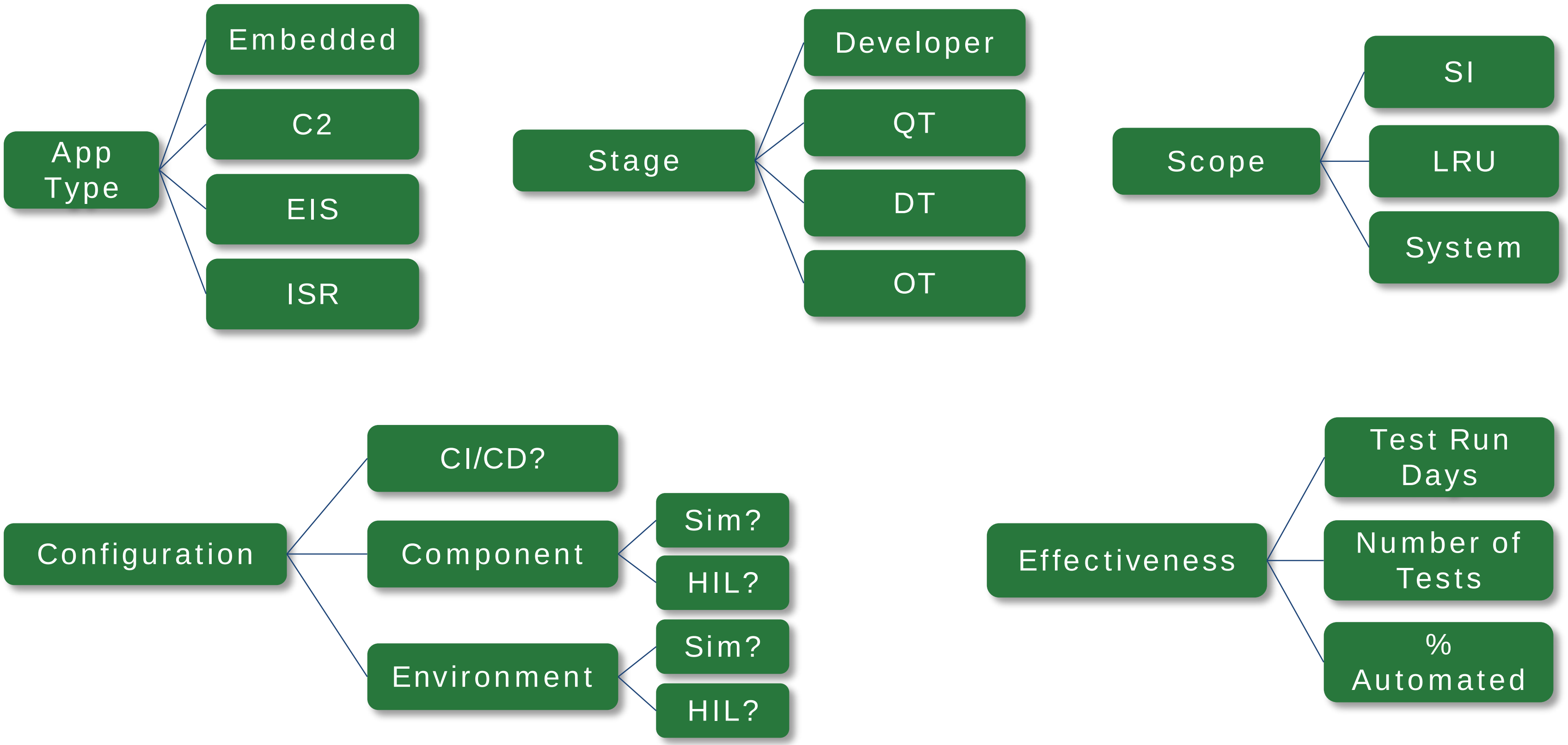
Composite of multiple SEI engagements

- 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

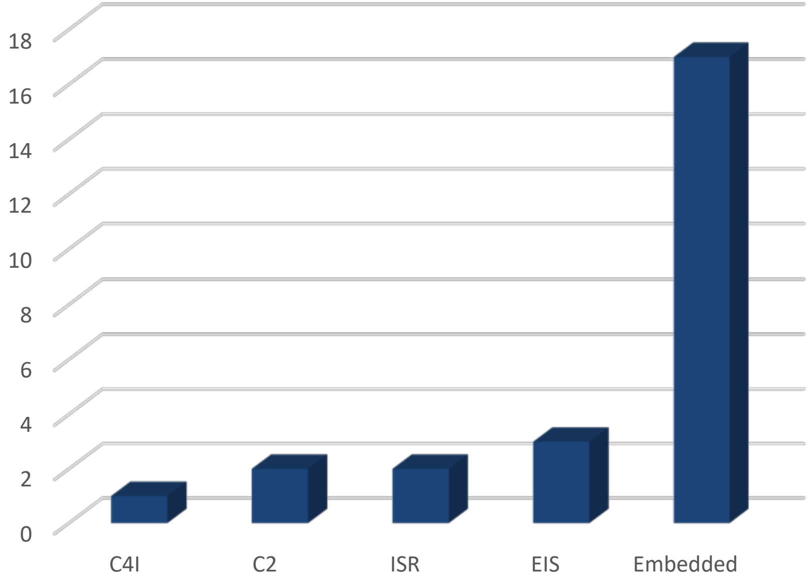
- Some SILs also used for training
- Almost no effective shift-left
- Many challenges for development and validation of simulation falsework
- Upstream testing often superficial
- No explicit design-for-testability
- High friction moving unclass to class
- Automated SILs have highest defect removal efficiency

Profiled Lab Characteristics

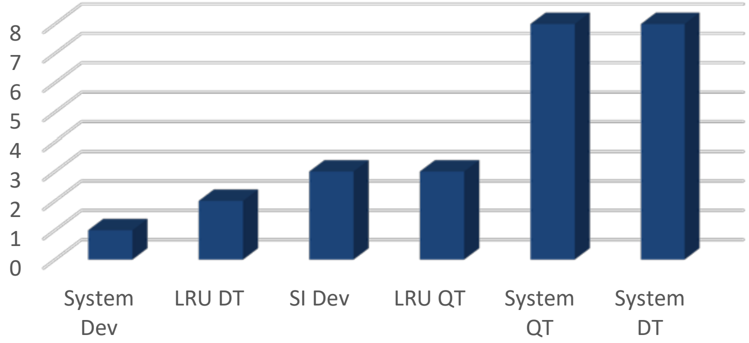


Application Type and Stage Used

Profiled SILs support mostly embedded apps …

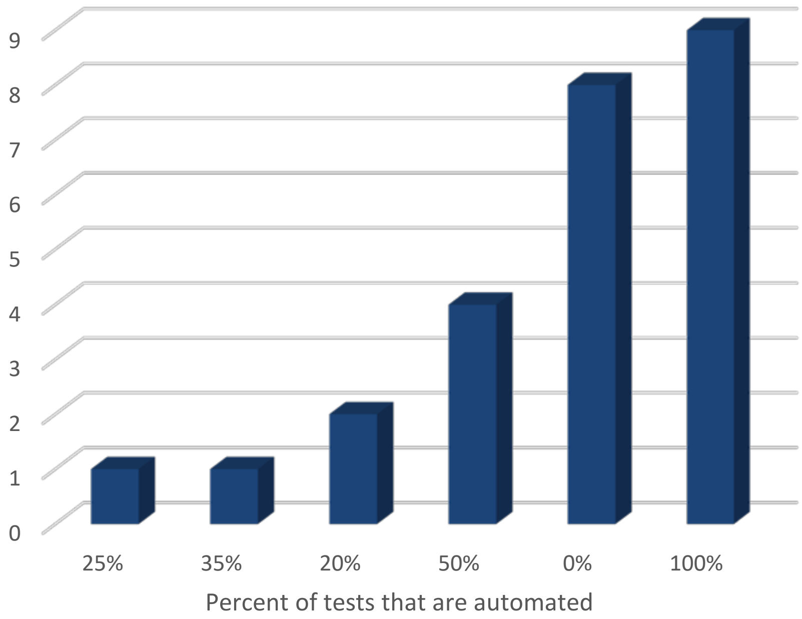


… during QT and DT

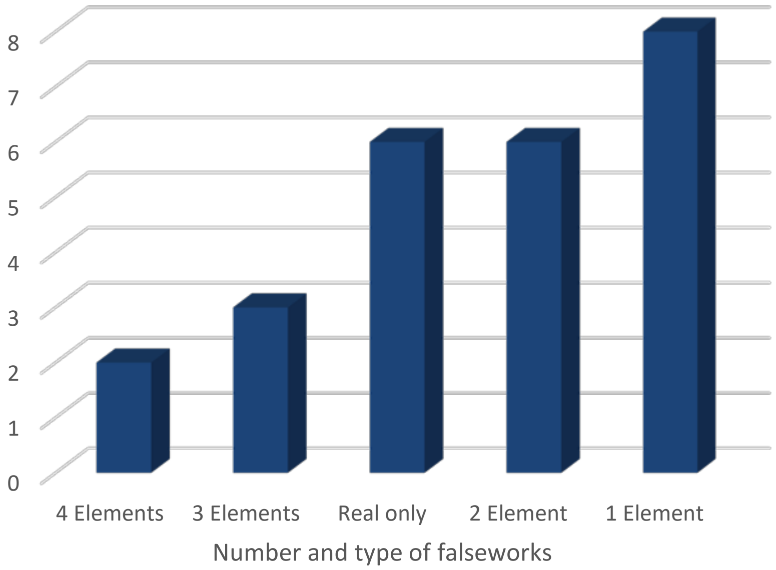


Configuration and Automation

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



1/4 use both Sim + HIL for LRU integration

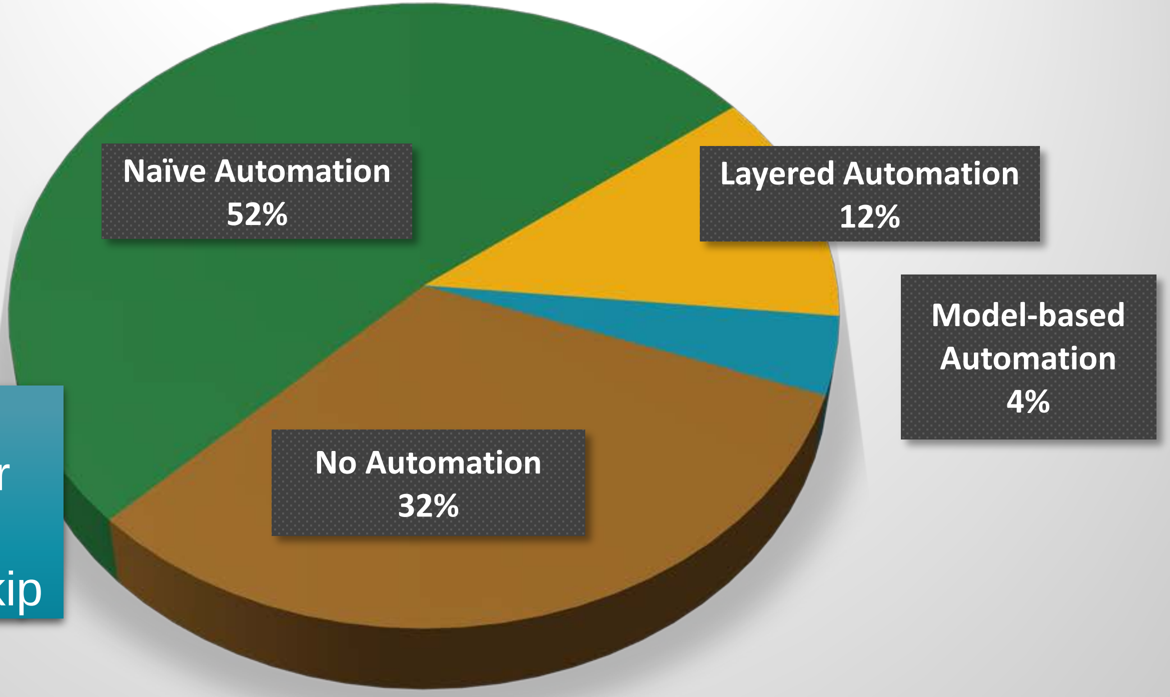


Test Automation Levels

	Ad Hoc Manual	Repeatable Manual	Naive Automation	Layered Architecture Automation	Model-based Automation
Main Benefits	Usability Omissions Lowest cost	Rqmt checkoff Repeatable Low cost	Rqmt checkoff Repeatable Run on demand	Rqmt checkoff Repeatable Run on demand Lower maint $	Maintain model Generate tests Deep coverage Lowest $/test
Concerns	Coverage? No repeat Inconsistent	Superficial Long test time	Superficial Brittle High maint $	Superficial SW eng stds	Staffing/Skilling
Risk Reduction	Low	Low-Med	Low-Med	Low-Med	Med-High
Tech Baseline Ownership via Test Design	None	Some	Some	Some	Deep

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

Profiled Labs Productivity Comparison

Test Execution Automation

Falsework/Real Configuration

	Manual Interaction No Assist	Manual Interaction Software Assisted	Automated Test Case Execution
LRU Simulated or Real Hardware-in-the-Loop	150		2200 500 200
LRU Simulated	390 70	100, 100 60	
Deployed System (all Real)	50 20	50 50	

(*nnn*): Estimated average test points evaluated per day during a test cycle

Best Practices: Testing Strategies

	Profiled Labs Best Practices	Automotive Best Practices
Coverage: every requirement at least once	✓	
Model-based Test Generation		✓
Test Asset Management		✓
Profile-based Reliability Testing		✓
Combinatorial Design		✓
Data-driven promotion/acceptance		✓

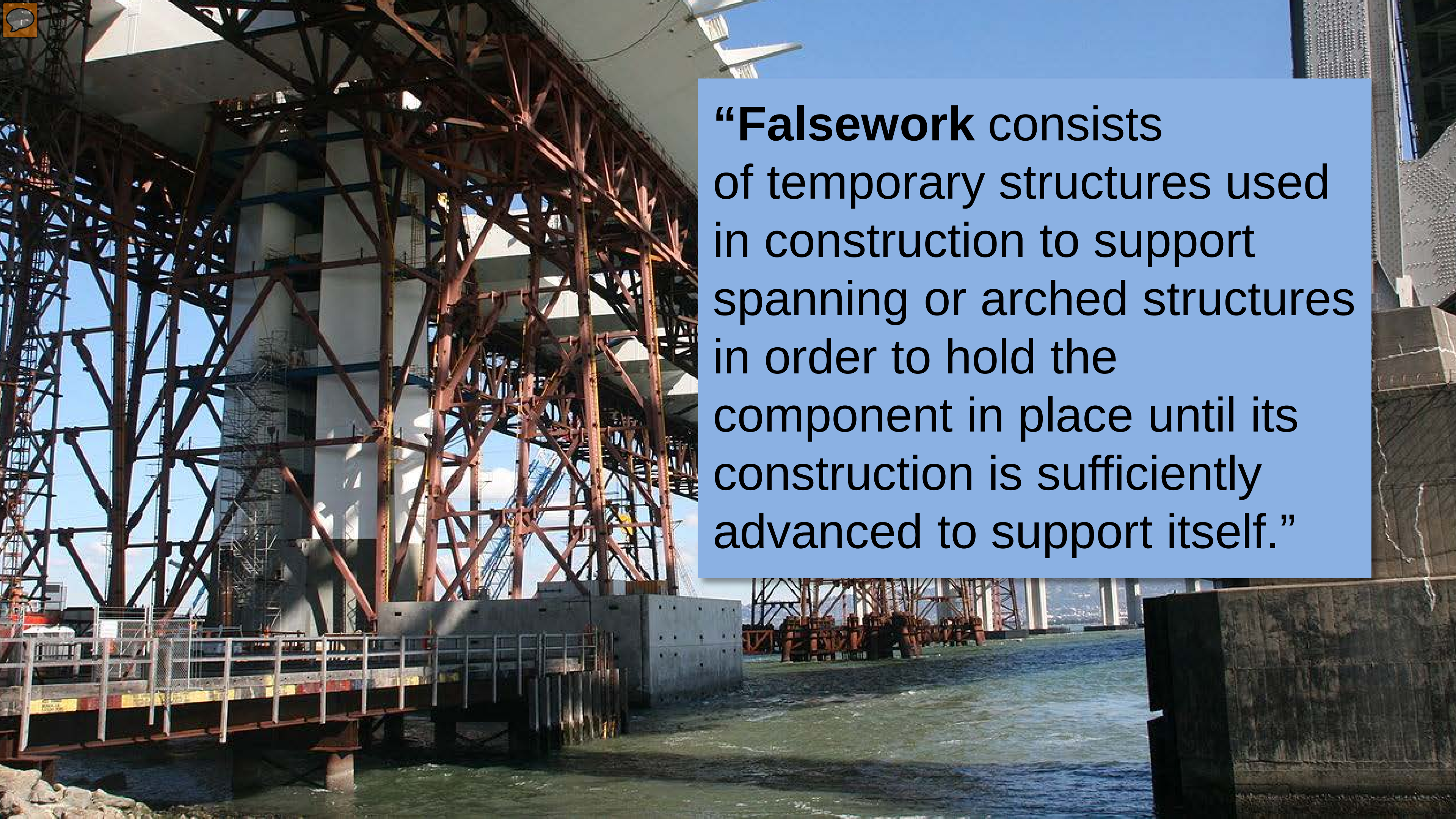
Best Practices: Testing System Configuration

	Profiled Labs Best Practices	Automotive Best Practices
SI build with controllable adapters		✓
Continuous Integration with LRU simulation		✓
LRU simulation	✓	✓
LRU simulation + Real LRU	✓	✓
Testable full-up chassis		✓
Manual testing on system	✓	✓
Manual testing on system, telemetry/capture		✓
Field monitoring		✓

Best Practices: Test Execution

	Profiled Labs Best Practices	Automotive Best Practices
Integrated component simulation/emulation	✓	✓
Automated test harness	✓	✓
User Interface automation	✓	✓
Network traffic monitoring	✓	✓
Controllable fault injection	✓	✓

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

	Purpose		Role in a SIL
Null	Placeholder	SW	May be part of SUT. Limits coverage, placeholder for unavailable interface.
Mock	Implement subset of SI behavior.	SW	May be part of SUT. Limits coverage, placeholder for unavailable interface.
Simulation	Mimic selected behaviors of an SI, sensor, actuator, external system, or user.	SW	Use a simulator to generate input or accept output.
Emulation	Replicate all behaviors of a component, sensor, actuator, or external system with hard real-time constraints.	SW/ HW	Use a HIL emulator to generate actuator input or sensor output using high speed digital devices.

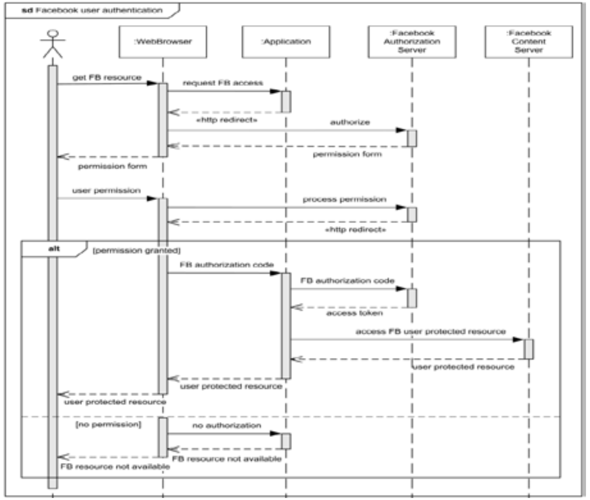
In software testing, falsework refers to stubs, mocks, fakes, “Service Virtualization,” generated or programmed simulations, and high fidelity hardware-based emulation.

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

	SI	RTS	HW	XS
SI	✓	✓	✓	✓
RTS			✓	
HW			✓	✓
XS	✓			



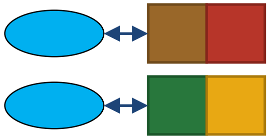
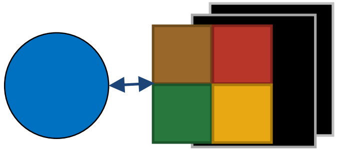
The structure of the system must be mapped to be sure interactions are covered. Some good sources are:

- Interface Control Documents (ICDs)
- Sequence diagrams
- Code analyzers like [SciTools Understand]
- Runtime logs and traffic
- User documentation

The Test Automation Tool Chain

There are many hundreds of COTS, FOSS, and GOTS software testing tools

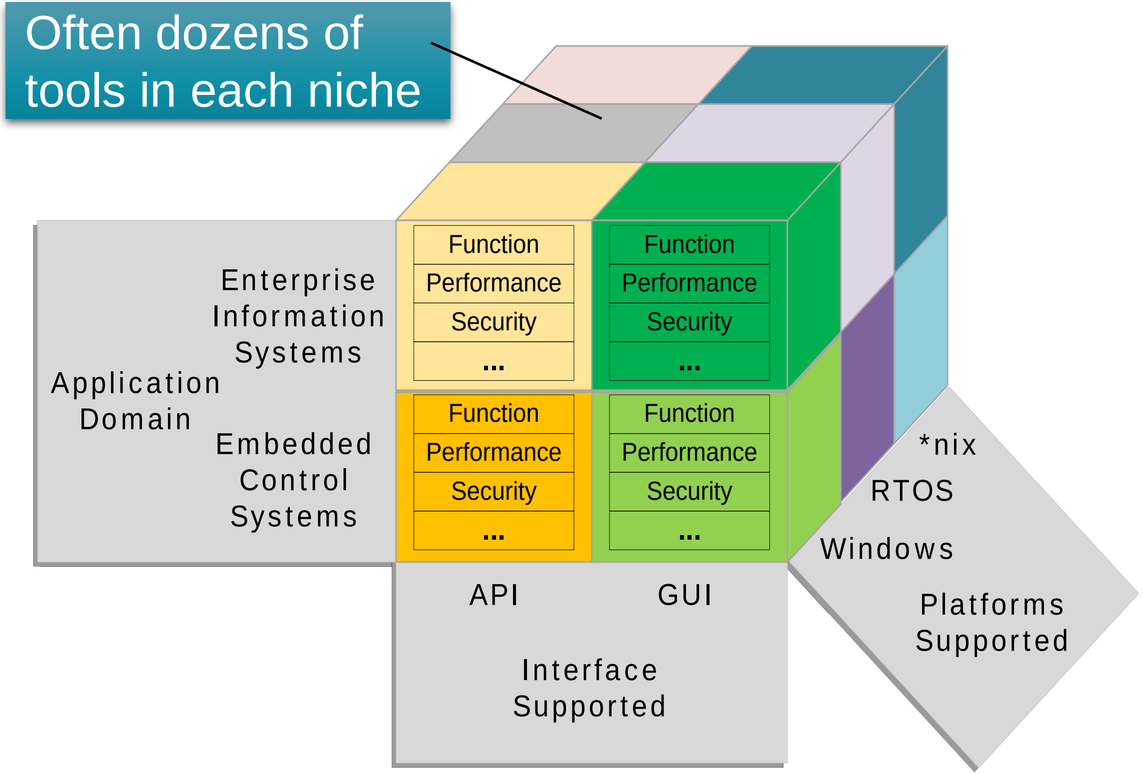
Testing scope, tooling, focus, lanes

Scope		*Example Tool*	*Focus*	CTR SQT	PO FAT	DTO	OTO
Developer, Unit, SI		Junit SonarCube	Functions	✓			
Component, SI, Subsystem		Selenium SOAP UI	Features, User stories	✓	✓	✓	
System, SoS		TestStand Jmeter	Use cases, Performance, Mission threads		✓	✓	✓

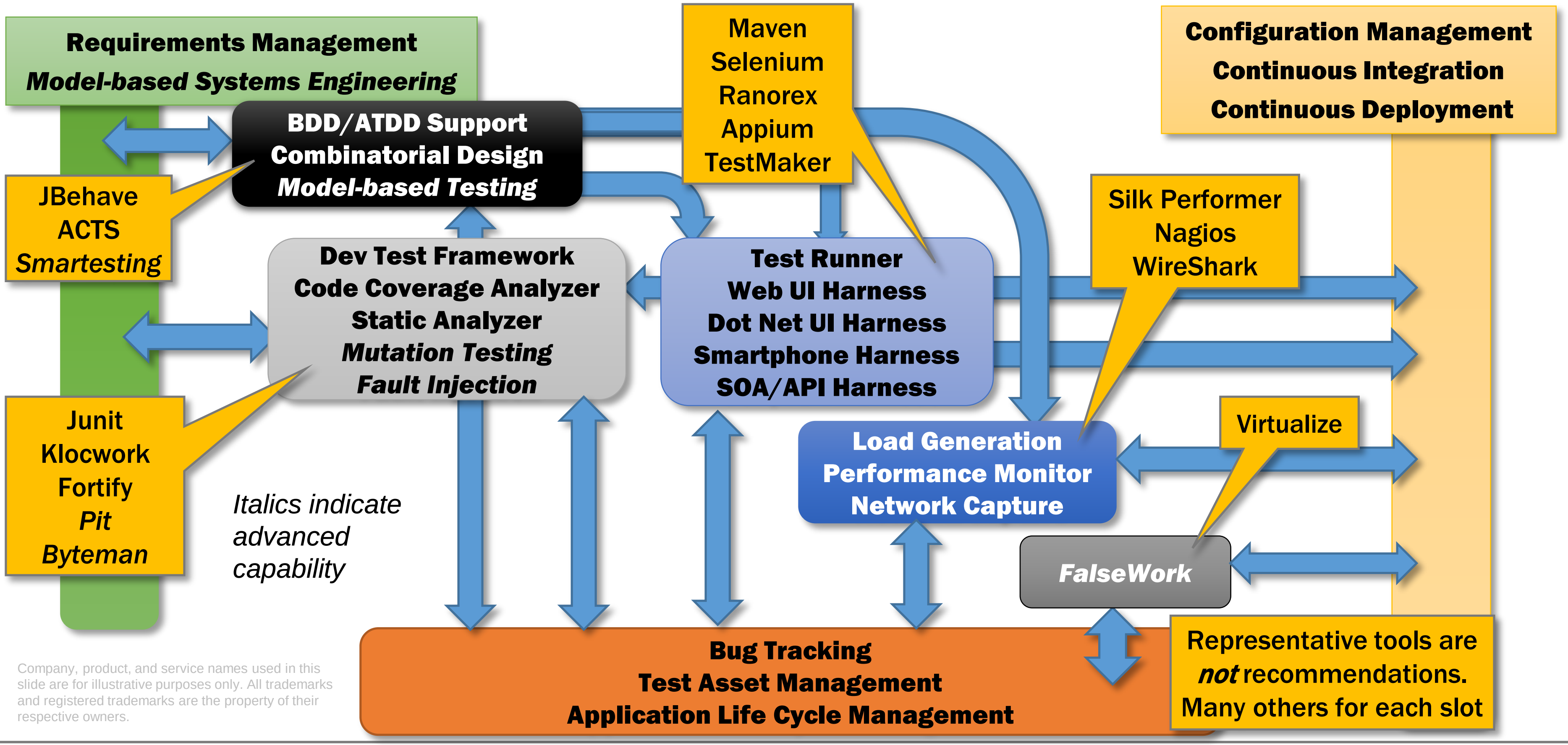
Which testing tool(s) is right for your job(s)?

Tools are specialized for:

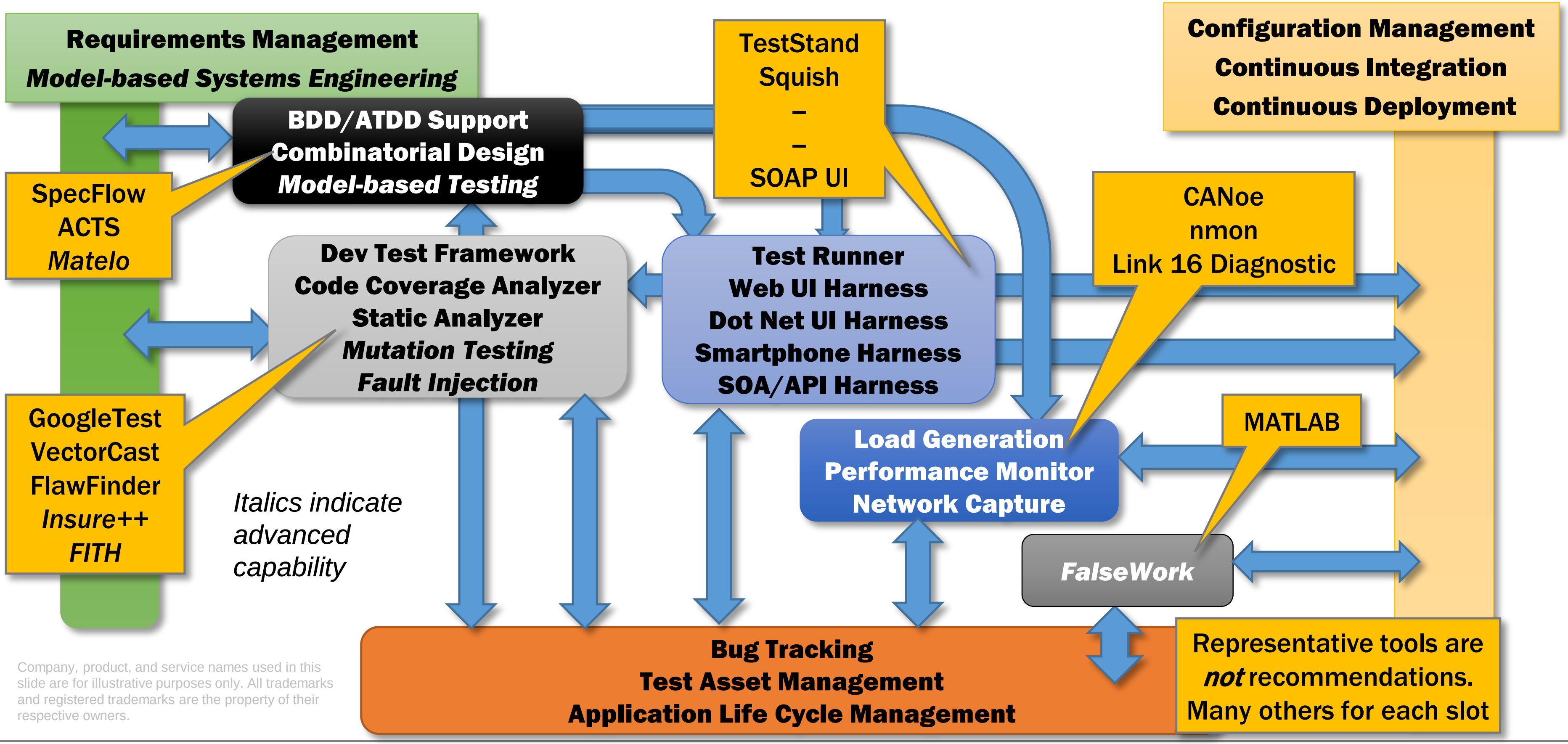
- Testing purpose
- Target interface, IDE, programming language
- Application domain: transaction processing, embedded, mobile app…
- Runtime stack(s) of target and tool



Test Automation Reference Architecture: Java/Cloud stack



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.

Acronyms

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

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