Building High Assurance Systems with SAFe® 4.0

Agile 2016
By Dean Leffingwell
What is a high assurance system?

High assurance systems are systems that have an unacceptable social or economic cost of failure.

Most high assurance systems are subject to regularity or industry compliance requirements, including Verification and Validation.
**Compliance meets Agile development**

### Regulatory and compliance
- Quality, safety, security, efficacy
- Specifications
- Verification and validation
- Inspections, audits, sign-off
- Documented quality management systems
- Metrics – defects, requirements coverage, code coverage, traceability

### Agile
- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan
- Welcome changing requirements….
**Verification** is the confirmation through objective evidence that the specified requirements have been fulfilled. Verification demonstrates that the design and implementation correctly and completely embody the requirements.

*You built the solution right*

**Validation** is the confirmation via objective evidence that the system performs its intended function. The intended functions and how well the system performs those functions are determined by the customer.

*You built the right solution*

**Compliance** is an organization’s adherence to relevant laws, specifications, and guidelines. Compliance typically requires documented, objective evidence of solution V&V through tests, inspections, analysis, or demonstrations.

*And you have evidence that you have done so*
“A documented software requirements specification (SRS) provides a baseline for both validation and verification. The software validation process cannot be completed without an established software requirements specification (Ref: 21 CFR 820.3(z) and (aa) and 820.30(f) and (g)). “

From the guidance, such a document typically contains:

- All software system inputs, outputs, and functions
- All performance requirements
So what’s to be done?

- Build the solution incrementally
- Organize around value
- Build quality in
- Apply continuous verification and validation
SAFe Version 4.0 has the hooks we need

© 2016 Scaled Agile, Inc. All Rights Reserved.

www.scaledagileframework.com
Build the Solution Incrementally
Apply fast learning cycles

Accelerate knowledge with fast feedback

- Improves learning efficiency by decreasing the time between action and effect
- Reduces the cost of risk-taking by truncating unsuccessful paths quickly
- Facilitated by small batch sizes
- Requires increased investment in development environment

*The shorter the cycles, the faster the learning*

*Principles of Product Development Flow, Don Reinertsen*

Build Incrementally with fast, integrated learning cycles

Waterfall

Requirements
Design
Implementation
Verification

Documents
Documents
Unverified System
System

Incremental

One PDA cycle
Iterate within the product lifecycle
The problem of phase-gate internal milestones

*There was in fact no correlation between exiting phase gates on time and project success…the data suggested the inverse might be true.* —Lean Machine

- Force too early design decisions; encourages false positive feasibility
- Assume a “point” solution exists and can be built right the first time
- Create huge batches, long queues; centralizes requirements and design in program management
Apply objective Milestones

PI Demos routinely deliver objective progress, product, and process metrics
Organize around value
Why organize around value?

Align the organization around projects and product lines.

—Allen C. Ward

1. Fewer handoffs, faster value delivery
2. Easier to build in quality
3. Built-in alignment between the business and software development
4. Optimizing the system as a whole

Result: Faster delivery, higher quality, higher customer satisfaction
Understand the full Value Stream

A fundamental thinking construct in Lean

- Each Value Stream is the sequence of steps used to deliver value to the Customer
- It includes the whole sequence, from concept or customer order to delivery of value and/or receipt of cash
- It contains the people who do the work, as well as the flow of information and materials
Value streams are cross-functional
Implement value with cross-functional Agile Release Trains

Business  Product Mgt/ Sys. Engineering  Hardware  Software  Quality  Testing  V&V

AGILE RELEASE TRAIN

Agile Teams
Synchronize with PI Planning

*Future product development tasks can’t be pre-determined. Distribute planning and control to those who can understand and react to the end results.* — Michael Kennedy, *Product Development for the Lean Enterprise*

- All stakeholders face-to-face (but typically multiple locations)
- Management sets the mission, with minimum possible constraints
- Requirements and design emerge
- Important stakeholder decisions are accelerated
- Teams create—and take responsibility for—plans

For a short video PI planning example, see: https://youtu.be/ZZAtl7nAB1M
Include all stakeholders in planning
Build quality in
SAFe quality practices

Assure every increment of the solution reflects quality standards

**Software**
- Continuous integration
- Test-First
- Refactoring
- Pair-work
- Collective ownership …

**Hardware**
- Exploratory early iterations
- Model Based Systems Engineering (MBSE)
- Set-Based Design
- Frequent, system-level integration
- Design verification
Architectural Runway

Architectural Runway—existing code, hardware components, etc. that technically enable near-term business features

- Enablers build up the runway
- Features consume it
- Architectural Runway must be continuously maintained
- Use Capacity Allocation (a percentage of train’s overall capacity in a PI) for Enablers that extend the runway
Test first and test automation

Test automation enables incremental development via regression testing

- Automated tests are implemented in the same iteration as the functionality
- The team that builds functionality also automates the tests
- Actively maintain test data under version control
MBSE facilitates emergent specifications

- Generate specifications from models in Solution Intent
- Everyone contributes; everyone takes a systems view
- Ensures single source of truth, both as-is and to-be
Continuous software integration

- Integrate every vertical slice of a user story
- Avoid physical branching for software
- Use development by intention in case of inter-team dependencies
  - Define interfaces and integrate first; then add functionality
Frequent system-level integration and testing

- Frequent integration and testing is the only objective evidence of solution integrity
- Trade-offs are inevitable in terms of:
  - Frequency of integration
  - Depth of integration
  - Fidelity of feedback
Solution Demo drives frequent solution integration

- Frequent solution integration and testing provides the best objective evidence
- A joint responsibility of ARTs’ and VS’s system teams
- Provides early validation and regular risk reduction
- Increases actual velocity
Inspect and Adapt

Every PI, teams systematically address the larger impediments that are limiting quality, compliance and velocity

Agree on the problem to solve

Insufficiently reliable release commitments?

Restate the new problem for the biggest root cause

Insufficient architectural runway

Apply root cause analysis (+ five whys)

Brainstorm solutions

Identify the biggest root cause using Pareto Analysis

Identify improvement Backlog items

© 2016 Scaled Agile, Inc. All Rights Reserved.
Apply continuous verification and validation
Solution Intent supports verification

Requirements Model
- Functional and use cases
- NFRs – safety, security
- Customer, regulatory

Domain Model
- Business process
- Business rules

Solution Model
- Analysis
- Architecture/interfaces

Test Model
- Test plans, test cases, execution results

Traceability

Realization
- Code, components
- Documentation
- Deployment

Compliance docs
Agile testing quadrants support V&V

Functional tests
- Feature/Capability acceptance tests
- Enabler acceptance tests
- Story acceptance tests

Acceptance tests
- Scenario tests
- Exploratory tests
- User acceptance tests
- Alpha and beta tests

System qualities tests
- Performance and load
- Security
- Other NFRs
- Enabler tests

Adapted from Brian Marick, Crispin and Gregory
SAFe requirements model supports continuous verification

**Functional behaviors**
- **Feature**
  - realized by (one to many)
  - tested by
  - Feature acceptance test
- **Story**
  - tested by
  - Story acceptance test
  - Unit test

**Nonfunctional behaviors**
- **Nonfunctional Requirement**
  - tested by
  - System qualities test
Make V&V activities part of regular flow

- Include compliance concerns in Definition of Done (DoD)
- Development continuously verifies
- System demos include validation status towards compliance
Iteration verification

Continuous verification

- Peer review of stories
- Implement stories and tests associated with committed features
- Test Solution with story acceptance tests
- Accept story into baseline
- Update Stories, tests and relationships in Solution Intent

Primarily development team responsibilities
Iteration validation

**Evaluate full system increment**

- Regression test all functional stories and feature acceptance tests
- User/Product Owner validation
- Run NFR tests
- Update compliance documentation

Development team, system team and program shared V&V responsibilities
PI validation

Evaluate full PI system increment

- Regression test all functional stories and feature acceptance tests
- User, PO, and PM validation
- NFR tests
- Update compliance documentation

Program-based V&V responsibilities

(May require handoff to Independent V&V)
Release validation

Fitness for purpose
- Final validation
- Qualification including installation and operation
- User acceptance testing

Supporting documentation
- Installation, user, and other
- Compliance evidence of V&V
Questions?