Applying Lean-Agile practices Large, Engineered Systems

Harry Koehnemann
SAFe Consultant and Fellow
harry@scaledagile.com
Agenda

- Overview of Lean-Agile principles
- Apply Lean-Agile principles to engineered systems
  1. Align on a common cadence
  2. Organize around value
  3. Plan at multiple levels
  4. Manage change
  5. Build the solution incrementally
  6. Build quality in
Why was the Wright Flyer an example of Lean Engineering?
SAFe Lean-Agile principles

#1-Take an economic view
#2-Apply systems thinking
#3-Assume variability; preserve options
#4-Build incrementally with fast, integrated learning cycles
#5-Base milestones on objective evaluation of working systems
#6-Visualize and limit WIP, reduce batch sizes, and manage queue lengths
#7-Apply cadence, synchronize with cross-domain planning
#8-Unlock the intrinsic motivation of knowledge workers
#9-Decentralize decision-making
Assume variability, preserve options

*Aggressively evaluate alternatives. Converge specifications and solution set.*
—Allen Ward

- You cannot possibly know everything at the start
- Requirements must be flexible to make economic design choices
- Preservation of options improves economic results
Apply fast, cadence-based learning cycles

Product development is the process of converting uncertainty to knowledge
—Dantar P. Oosterwal

Integration points control product development
- Integration points accelerate learning
- Development can proceed no faster than the slowest learning loop
- Improvement comes through synchronization of design loops and faster learning cycles

The Lean Machine: How Harley Davidson Drove Top-Line Growth and Profitability with Revolutionary Lean Product Development
—Dantar P. Oosterwal
1) Align everyone on a common cadence

Driven by early decisions and fixed schedule

Driven by learning and feedback
2) Organize around value
Organizing around value at scale

- Aligned on a common cadence
- Get comfortable with Collective Ownership
- Requires Continuous Integration
- Leverage Community of Practices / Scrum of Scrums
3) Plan at multiple levels

- Outer levels less defined, committed
- Inner levels more understood, detailed, committed

- Multi-year vision, milestones, events, and roadmap
- Short-term (3-4 PI) estimate of Features and Milestones
- Committed Features and Enabler for current PI
Make Systems Engineering work part of agile flow

- In Lean-Agile, all work is flow-based and performed in small batches.
- Consequently, SE activities must be part of flow.
Define intent and roadmap to move from as-is to to-be

- Evolve the intent and roadmap based on learning

New knowledge
- Market/business changes
- Technical discoveries
- Lean UX – feedback

Increment goals
- New Features
- Experiments

Specifications  Roadmap  TO-BE  AS-IS  Solution
Evolve together
How do we sequence the work?

**High risk**
- High learning – could cancel the project
- High risk - impacts budget or schedule

**High value**
- Core value (MVP)
- High value (MMF)

**Understand dependencies**
- Priority inversion for dependent items

**Consider ART capacities**
- Capacity limited by ART throughput
Validate assumptions early with MVPs

- Don’t assume point solutions
- Explore alternatives through exploration activities to gain knowledge
- Build minimum solution to gain desired knowledge (MVP)
- Utilize proxies for parts of the system not yet built
Mitigate risks using Set-Based Design (SBD)

- Keep requirements and design options open as long as possible
- Explore alternatives to arrive at the *optimal* decision, not the *first* decision
Record and communicate knowledge with tradeoff curves

- Characterize the fundamental tradeoffs governing system performance
- Test, measure, and record those decisions in limit curves
- Understand the relationships between conflicting design parameters
- Intentionally vary parameters to understand limits of what is feasible
Exploration is continuous

- Learning performed in small batches
- Gain knowledge at last responsible moment
4) Build the solution incrementally

- Frequent integration provide fast feedback and new knowledge
- Trade-offs are inevitable in terms of:
  - Frequency of integration
  - Depth of integration
  - Fidelity of feedback
Invest in infrastructure and practices to lower integration cost

- Large hidden delays in the build-integrate-test-deploy process
- Strive to automate the entire end-to-end process for the entire system

*Principles of Product Development Flow, Don Reinertsen*
Align functional and physical roadmaps

- Hardware teams create proxies for their learning – coordinate them
- Strive for early, end-to-end solution mockup that matures in fidelity over time
- Hardware teams responsible for supporting incremental demonstrations
5) Build quality (and compliance) in

Traditional testing (V-Model) delays feedback

- Determine Feature
- Test Feature
- Write Story
- Test Story
- Write Code
- Test Code
Shift testing left for fast and continuous feedback

Determine Feature → Test Feature → Write Story → Test Story → Write Code → Test Code

... always testing...

Shift Testing Left

FEATURE TESTS (BDD)
Behavior-Driven Development (BDD)

STORY TESTS (BDD)
Behavior-Driven Development (BDD)

CODE TESTS (TDD)
Test-Driven Development (TDD)
Early, automated tests build a balanced test portfolio

- Test Pyramid advocates many small, low-level, automate tests and fewer large, manual tests

![Test Pyramid Diagram]

**Traditional testing (Find defects)**
- Large (Slow)
- Medium
- Small (Fast)

**Agile testing (Prevent defects)**
- Small (Fast)
- Medium
- UI

End-to-end UI tests

External services

Individual classes

$\$\$

$\$

© Scaled Agile, Inc.
Test automation builds quality and compliance in

*Give teams automated scripts instead of checklists*

- Automate tests in the same iteration as the functionality
- Include tests for safety, security, performance, quality, etc.
- Invest in automated testing infrastructure to improve flow
- Actively maintain test data under version control

![Diagram showing test automation processes and statuses](image-url)
Perform verification and validation continuously

Evaluate full system increment

- Regression test all functional stories, NFRs, and feature acceptance tests
- Tested on end-to-end test environment
- User/Product Owner validation
- Update V&V tests
- Generate compliance docs and check progress towards acceptance

Development teams, system team and program shared V&V responsibilities
Lean-Agile is hyper-focused on built-in quality

Architect/Design Quality

- Design for testability with components and interfaces
- Abstraction, encapsulation, SOLID
- Set-based design

Code (artifact) Quality

- Test-First – TDD, BDD
- Pair work
- Collective ownership
- Refactoring
- Standards

System Quality

- Align with BDD
- Communicate with MBSE
- Continuous delivery pipeline

Release Quality

- Component-level and team-level release-ability
- Immutable infrastructure
- Continuous V&V and compliance
Summary

- Lean-Agile principles apply to engineered systems through…
  - Align on a common cadence
  - Organize around value
  - Manage change
  - Build the solution incrementally
  - Build quality in
Questions
Thank you!

Harry Koehnemann
SAFe Consultant and Fellow
harry@scaledagile.com