# Balancing the Tension between Lean and Agile

James O. Coplien
Gertrud&Cope
Scrum Foundation
cope@gertrudandcope.com

ARK 2011

# What is Agile?

- We all know the Manifesto
- It comes down to two things:
  - Self-organization
  - Feedback
- An exercise exemplifying Agile

Agile Finland 2009

# The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.


- Kent Beck
- Mike Beedle
- Arie van Bennekun
- Alistair Cockburn
- Ward Cunningham
- Martin Fowler
- James Grenning
- Jim Highsmith
- Andrew Hunt
- Ron Jeffries
- Jon Kern
- Brian Marick
- Robert Cecil Martin
- Steve Mellor
- Ken Schwaber
- Jeff Sutherland
- Dave Thomas






# What is Lean?

- Lean is a more complex system aimed at adding value for the end user
- To do that,
  - We eliminate waste
  - We eliminate inconsistency
  - We smooth out flow
- We are constantly improving: Kaizen
- An exercise to illustrate Lean

# Just a thought...

- Lean:
  - Eliminate Waste
  - Eliminate inconsistency
  - Smooth out flow
- Agile:
  - Favor product over documentation
  - Communication
  - Sustainable pace

Snowden and Boone, A Leader's Framework for Decision Making, Harvard Business Review, Nov. 2007



# Reflection

- What can you learn from the Agile exercise that would help software development?
- What about the same for the Lean exercise?
- What is common between these two exercises?
- What is different?
- Can you see any conflicts between them?

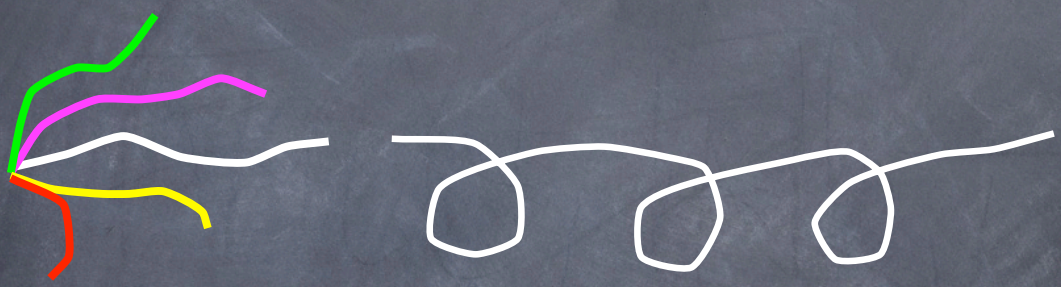
# Lean and Agile Characteristics


By invitation
Thinking
Feed-Forward
High throughput
Monochronic, planned
Develop world models
Process

Inclusive
Doing
Feedback
Low Latency
Polychronic, reactive
Shared world models
People



# Set-based Design: Complementary to Feedback



Ten Alternatives

Refine the one chosen

+ Rework in analysis adds value; rework in manufacturing is cost

# Complex versus Complicated

- Agile: Dealing with complex systems: autopoeietic systems, self-organization; wholes greater than the sum of their parts
- Lean: Dealing with complicated systems. Building a car is complicated but not complex; the whole is the sum of its parts

Snowden and Boone, A Leader's Framework for Decision Making, Harvard Business Review, Nov. 2007



# Standards?

- Agile: Inspect and adapt: anyone can do it, you don't need to ask permission, you are empowered, and you achieve continuous improvement
- Lean: if you have a problem, spend up-front time seeking standards (Toyota Way, principle 6: Standardized Tasks are the Foundation for Continuous Improvement and Employee Empowerment)

Liker, Jeffrey K. The Toyota Way, McGraw-Hill, ©2004, Chapter 12, pp. 140 - 148

# Doing or Thinking?

- Agile: embrace change (but more on this later)
- Lean: Long deliberation and thought with rapid deployment only after a decision is made (The Toyota Way, Principle 13: Make decisions slowly by consensus, thoroughly considering all options)

Liker, Jeffrey K. The Toyota Way, McGraw-Hill, ©2004, Chapter 19, pp. 237 - 249



# Specialization

- XP: No code ownership, no specialization. Scrum: cross-functional team
- Lean: spend years carefully grooming each individual to develop a depth of knowledge (from Toyota Way, Principle 10)

Liker, Jeffrey K. The Toyota Way, McGraw-Hill, ©2004, Chapter 16, pp. 184 - 198

# Rework

- Agile: Refactoring compensates for architectural short-sightedness, maintenance, and emergent requirements (as well as keeping the code clean)
- Lean: Rework in design adds value, while rework in production is waste (Ballard: Negative Iteration, Lean Institute)

Ballard, Glenn, Positive vs Negative Iteration in Design. Lean construction Institute, University of California, Berkeley



# Last Responsible Moment

- Agile: early decisions are likely to be wrong and cause rework, so defer to the last responsible moment
- Lean: letting a decision go beyond the point where it affects other decisions causes rework, so bring decisions forward to a point where their results don't propagate

Ballard, Glenn, Positive vs Negative Iteration in Design. Lean construction Institute, University of California, Berkeley

Agile Finland 2009

# Summary

- Complex vs Complicated
- Ad-hoc vs Standards
- Doing vs Thinking
- Generalization vs Specialization
- Rework vs Protoyping
- Late versus Early Decisions



# Focus

- Agile: Doing
  - Where is the thinking in Scrum?
  - Scrum allows thinking but neither requires it nor provides techniques for it
  - Agile is about doing just enough to get just enough quality
- Lean: Doing and thinking
  - Lean requires thinking and even planning
  - Lean is about doing the least you can do to achieve excellence

# Repercussions

- Agile fears future change; Lean embraces it
  - Enlist a fire brigade or build a brick city?
- Agile shuns architecture; Lean considers it essential
- Agile avoids committing to agreed standards; Lean is about commitment
- Agile is about individuals; Lean about teams
- Agile diffuses responsibility; Lean encourages it
- Agile puts value in production rework (refactor; do-inspect-adapt); Lean avoids rework (design, inspect-plan-do)



# Reflection

- We can mix Lean and Agile
- Reflect on a mixture of practices, ideas, and philosophies that combine Lean and Agile thinking

# Scrum

- Comes from a Lean legacy, but has many trappings we associate with Agile
- Common problems in Scrum implementations:
  - People don't do the up-front analysis, lean architecture
  - People don't take the value chain out to the end user
  - People think locally rather than in terms of their networks



# Does the Nokia Test test Lean or Agile?

- Iterations
- Expanding scope of Done to deployment
- Up-front specifications w/User Stories
- Product owner who plans
- Up-front Product Backlog
- Up-front estimates
- Business-oriented burndown chart
- Team disruption

# Most Nokia Test points are Lean — not Agile

- Iterations (Agile)
- Expanding scope of Done to deployment (Lean)
- Up-front specifications w/User Stories (Lean)
- Product owner who plans (Lean)
- Up-front Product Backlog (Lean: DSM)
- Up-front estimates (Lean planning)
- Business-oriented burndown chart (Lean)
- Team disruption (Agile)



# Summary: The Lean / Agile Compromise

- Do up-front work — but minimize artefact inventory and be postured for change
  - Prototypes and enabling specifications
  - Architecture
- Make short-term sacrifices for long-term ROI
- Pay strong attention to common strengths, such as short feedback loops
- Use common sense!

# 1. Iterations

- No iterations - 0
- Iterations > 6 - 1
- Variable length < 6 weeks - 2
- Fixed iteration length 6 weeks - 3
- Fix iteration length 5 weeks - 4
- Fixed iteration 4 weeks or less - 10



Back

# 2. Testing

- No dedicated QA - 0
- Unit tested - 1
- Feature tested - 5
- Feature tested as soon as completed - 7
- Software passes acceptance testing - 8
- Software is deployed - 10



Back

# 3. Agile Specification

- No requirements - 0
- Big requirements document - 1
- Poor user stories - 4
- Good requirements - 5
- Good user stories - 7
- Software is deployed - 10
- Just enough, just in time specifications - 8
- Good user stories tied to specifications as needed - 10



Back

# 4. Product Owner

- No Product Owner - 0
- Ignorant product owner - 1
- Meddling product owner - 2
- Detached product owner - 2
- Product owner with clear product backlog estimated by team before Sprint Planning meeting (READY) - 5
- Product owner with release road map based on team velocity - 8
- Product owner who motivates the team - 10



Back

# 5. Product Backlog

- No Product Backlog - 0
- Multiple product backlogs - 1
- Single product backlog - 3
- Product backlog clearly specified and ordered for ROI before sprint planning (READY) - 5
- Product owner with release plan based on Product Backlog - 7
- Product Owner can measure ROI based on revenue, cost per story point, or other metrics - 10



Back

# 6. Estimates

- Product Backlog not estimated - 0
- Estimates not produced by team - 1
- Estimates not produced by planning poker - 5
- Estimates produced by planning poker by team - 8
- Estimate error < 10% - 10



Back

# 7. Burndown Chart

- No burndown chart - 0
- Burndown chart not updated by team - 1
- Partial task burndown - 2
- Using Track Done - 4
- Using Track Story done - 5
- Add 3 points if the team knows velocity
- Add 2 points if Product Owner release plan is based on known velocity



Back

# 8. Team Disruption

- Manager / Project Leader disrupts team - 0
- Product Owner disrupts team - 1
- Managers, Project Leaders or Team Leaders assign tasks - 3
- Have Project Leader and Scrum roles - 5
- No one disrupts team, only Scrum roles - 10



Back