Agile in Government: Executive Overview

October 2019

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Carnegie Mellon University
Software Engineering Institute

Agenda

Today's landscape

Agile basics: meaning behind the vocabulary

Beyond the small team: Agile in the larger ecosystem

Scaled Agile Framework (SAFe)

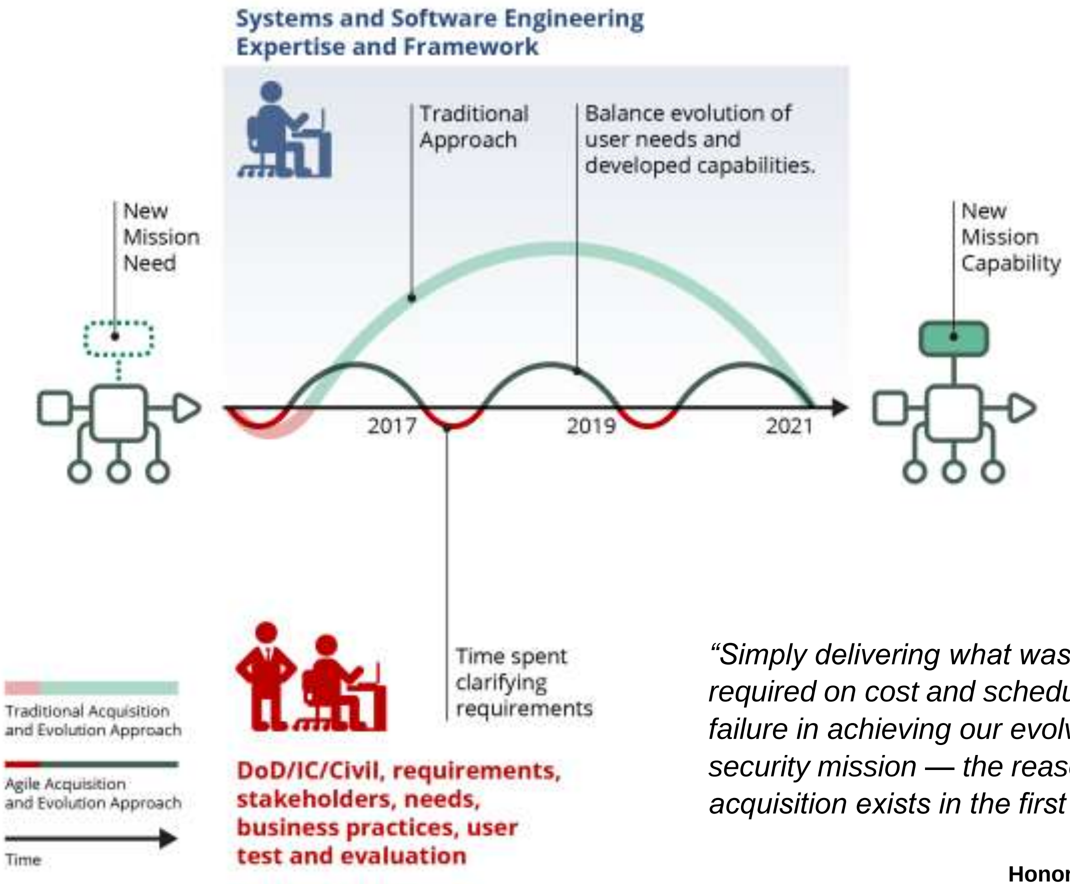
How do we get there: enabling Agile culture

Why does the DoD/Govt care?

Deliver performance at the speed of relevance

Streamline rapid, iterative approaches from development to fielding

National Defense Strategy Summary
Jan 2018

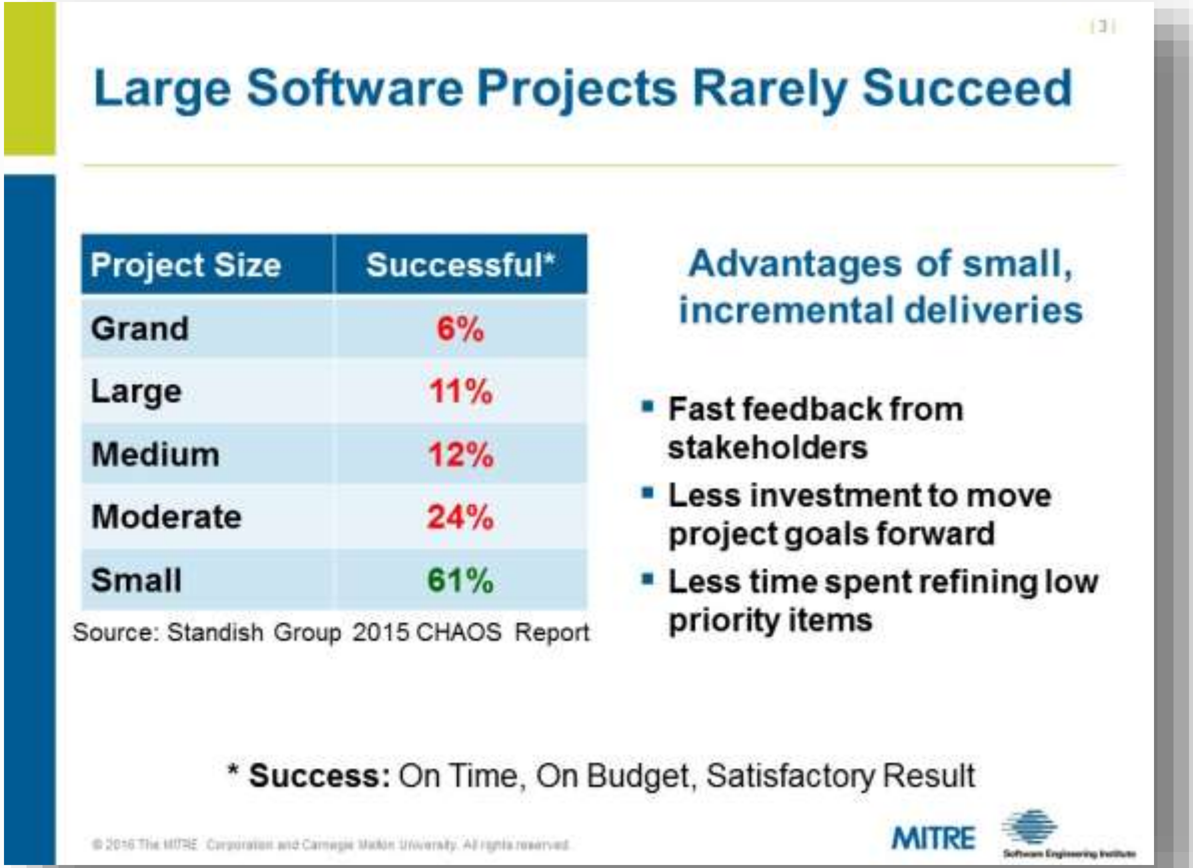

Systems and Software Engineering Expertise and Framework
Traditional Approach
Balance evolution of user needs and developed capabilities.
New Mission Need
New Mission Capability
2017
2019
2021
Time spent clarifying requirements
DoD/IC/Civil, requirements, stakeholders, needs, business practices, user test and evaluation
Traditional Acquisition and Evolution Approach
Agile Acquisition and Evolution Approach
Time

"Simply delivering what was initially required on cost and schedule can lead to failure in achieving our evolving national security mission — the reason defense acquisition exists in the first place."

Honorable Frank Kendall
Under Secretary of Defense (AT&L)
2015 Performance of The Defense Acquisition System

Sample of Reported Results on DoD/Federal Programs

- Quantifiable cost savings and 6-month early delivery
- Significant cost avoidance
- Reduced rework & unplanned releases
- Dramatically increased productivity/capacity, with reduced cost of delivery
- Improved insight into contractor performance and progress
- Early discovery & resolution of Cat 1 defects (one year prior to integration test event)
- Early discovery & resolution of interface issues
- Improved flight test efficiency
- Early insight for end users into functionality of delivered system
- Better responsiveness to users with rapidly fluctuating requirements
- Heightened awareness & collaboration, improved realization of tradeoffs
- Improved workflow management



Large Software Projects Rarely Succeed

Project Size	Successful*
Grand	6%
Large	11%
Medium	12%
Moderate	24%
Small	61%

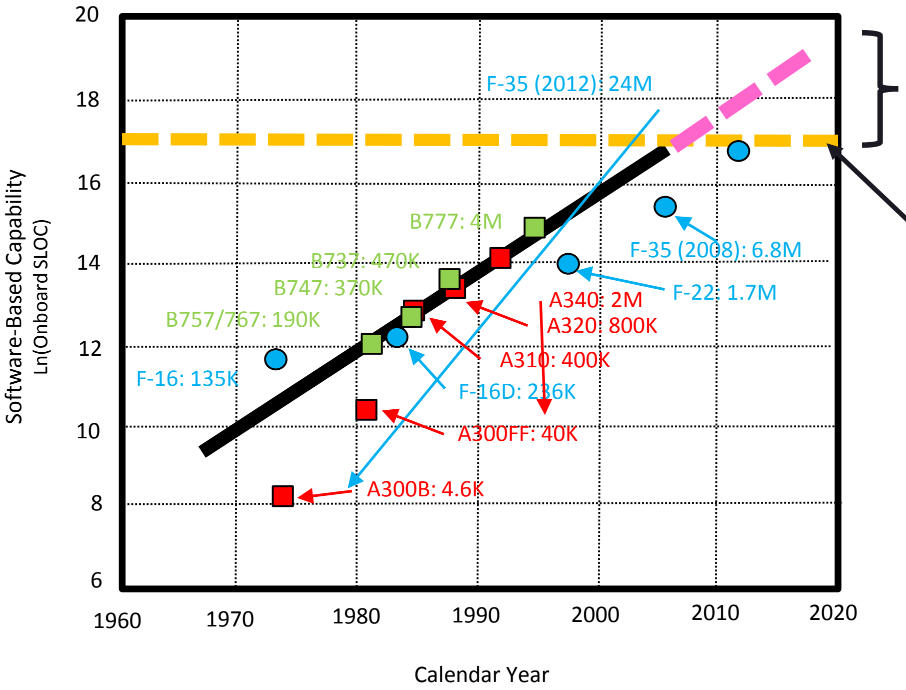
Source: Standish Group 2015 CHAOS Report

Advantages of small, incremental deliveries

- Fast feedback from stakeholders
- Less investment to move project goals forward
- Less time spent refining low priority items

* **Success:** On Time, On Budget, Satisfactory Result

Complex software costs pose a military threat (e.g., in Aviation Software)



We are now in an era where software costs limit military capability

SAVI projects a <u>limit of affordability </u>at 27.5MSLOC or $10B in software costs

Augustine's Law #16
"In the year 2054, the entire defense budget will purchase just one tactical aircraft. This aircraft will have to be shared by the Air Force and Navy 3½ days each per week except for leap year, when it will be made available to the Marines for the extra day."
Norman Ralph Augustine

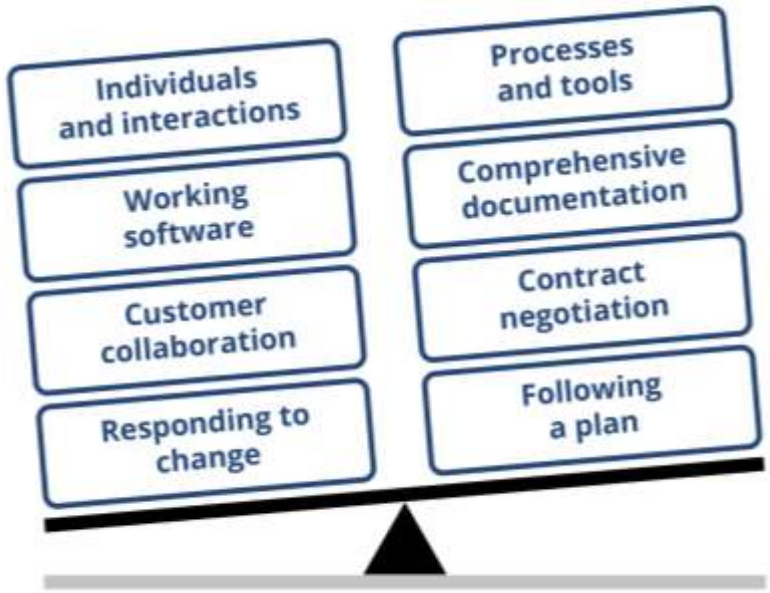
Software as percentage of total system cost : 1997: 45% 2010: 70% 2020: 80+%

SLOC: Source Lines of Code (a proxy measure of software complexity/functionality)

SAVI: System Architecture Virtual Integration (incl. members Airbus, Boeing, Embraer, US FAA/NASA, Honeywell, Rockwell Collins, CMU and UTC)

Agile Manifesto

Through this work we have come to value:



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

Common myth:

The manifesto is often misinterpreted to mean:

no documentation,
no process, and
no plan!

http://www.agilemanifesto.org/

Agile Principles-1

1. Highest priority is satisfy the customer through early and continuous delivery of software.

2. Welcome changing requirements, even late in development…

3. Deliver working software frequently, from a couple of weeks to a couple of months...

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Provide environment and support they need…

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile Principles – 2

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development…a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity—the art of maximizing the amount of work not done—is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Adapted from http://agilemanifesto.org/principles.html

Working Definition of Agile

Business Team
To succeed, all levels of an organization (& the customer) need to support the agile process
Technical Team
customer

Agile (*adj.*): An *iterative* and *incremental* (evolutionary) approach to software development which is performed in a *highly collaborative manner* by *self-organizing teams* within an *effective governance framework* with *“just enough” ceremony* that produces *high quality software* in a *cost effective and timely* manner which *meets the changing needs of its stakeholders*. [Ambler 2013]

[Ambler 2013] Ambler, Scott. *Disciplined Agile Software Development: Definition*. http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm

Some Observable Characteristics of Agile Implementations

Iterative—elements are expected to move from skeletal to completely fleshed out over time, not all in one step

Incremental—delivery doesn’t occur all at once

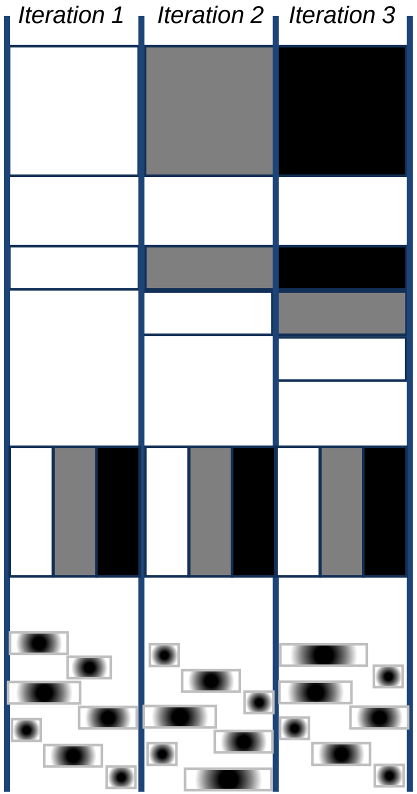
Collaborative—progress is expected to be made by stakeholders and the development team working collaboratively throughout the development timeframe

Loosely-coupled Architecture—multiple self-organizing, cross-functional teams work concurrently on multiple product elements (e.g., requirements, architecture, design, and the like) for multiple loosely coupled product components

Dedicated—team members are allowed to focus on the tasks within an iteration/release as opposed to multi-tasking across multiple projects

Time-boxed or Flow-based—relatively short-duration development cycles that permit changes in scope rather than changes in delivery time frame

Taking an Iterative Approach



Single batch – one process steps per iterations

Multiple batches, one process step per batch per iteration

Multiple batches, complete all work on each batch at the end of each iteration

Further decomposition into smaller packages, with multiple start-to-finish cycles in each iteration.

Traditional vs. Agile Approaches

Traditional approach

- Is consistent with the acquisition lifecycle provided in typical acquisition guidance
- Works well for
  - programs with stable requirements and environment, with known solutions to the requirements
  - programs with a homogeneous set of stakeholders who communicate well via documents
  - programs for which the technology base is evolving slowly (technology is not expected to be refreshed/replaced within the timeframe of the initial development)

Agile approach works well for

- programs with volatile requirements and environment
- programs where solutions are sufficiently unknown that significant experimentation is likely to be needed
- programs for which the technology base is evolving rapidly
- *programs with stakeholders who can engage with developers in ongoing, close collaboration*

Nidiffer, K. Miller, S. & Carney, D. *Potential Use of Agile Methods in Selected DoD Acquisitions: Requirements Development and Management* (CMU/SEI-2013-TN-0006), September 2013.

Important Points to Remember: Agile Basics

Agile is an iterative, incremental, highly collaborative approach that prioritizes responsible responsiveness to changing conditions and as-built product over projections

- There are many valid ways to implement the principles
- A wide variety of popular engineering methodologies fall under the umbrella of “Agile”

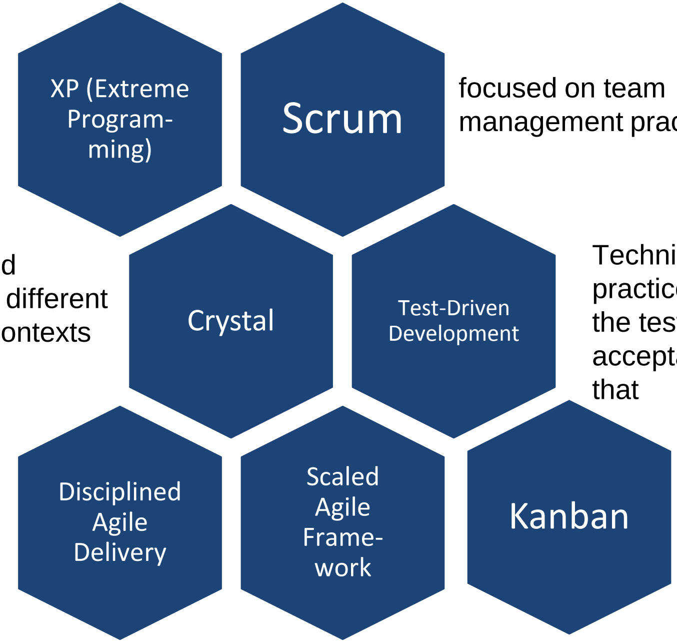
Agile approaches require collaboration across the enterprise to be successful

- Contracts, finance, test, end users…

Agile approaches support fast learning cycles and adaptation to changing conditions/volatility

- Changes in technology, threats, priorities and diverse stakeholders, unknown solutions/experimentation
- Traditional highly sequential (“waterfall”) approaches are well-suited to homogeneous, stable environments with slowly changing requirements

Many Methods Generally Termed “Agile”



XP (Extreme Programming)

focused on team technical practices

Scrum

focused on team management practices

Crystal

Encourages risk-based selection of practices; different patterns for different contexts

Test-Driven Development

Technical and management practices focused on writing the test that proves acceptance, then coding to that

Disciplined Agile Delivery

Originally derived from Rational Unified Process, designed to scale

Scaled Agile Framework

Merger of lean, Kanban, and other Agile methods to support large scale projects

Kanban

Pull-based approach particularly favored for services like security, systems engineering

Agile Principles were Designed & Focused on Small Teams

We operate on a massive scale – ***how does Agile work "in the large"?***

Some considerations when scaling above a few small teams:

- Managing interfaces among the many products/system components that multiple teams are working on…
- Synchronizing releases and events across multiple teams…
- Organizing inventory (backlog) of requirements productively to support the development pace of multiple small teams….
- Dealing with specialty disciplines (UX, security, etc.) that have significant inputs to the evolving product, but aren't needed as full time team members….
- Mindfully specifying architecture ("just enough") and other far-reaching concerns…
- Incorporating high assurance requirements (safety of flight, IA, nuclear surety…)

Foundations of the Scaled Agile Framework® (SAFe®) 4.5

V4.5.0

SCALED AGILE®

We thought we'd be developing like this.

But sometimes it feels like this.

Library of Congress

And our retrospectives read like this:

No way to improve systematically

Too little visibility

Too early commitment to a design that didn’t work

Late delivery

Problems discovered too late

Under-estimated dependencies

Massive growth in complexity

Hard to manage distributed teams

Phase gate SDLC isn't helping reduce risk

Poor morale

Management's challenge

It is not enough that management commit themselves to quality and productivity. …
They must know what it is they must do.

Such a responsibility cannot be delegated.

—W. Edwards Deming

"… and if you can't come, send no one."
—Vignette from *Out of the Crisis*, Deming,1986

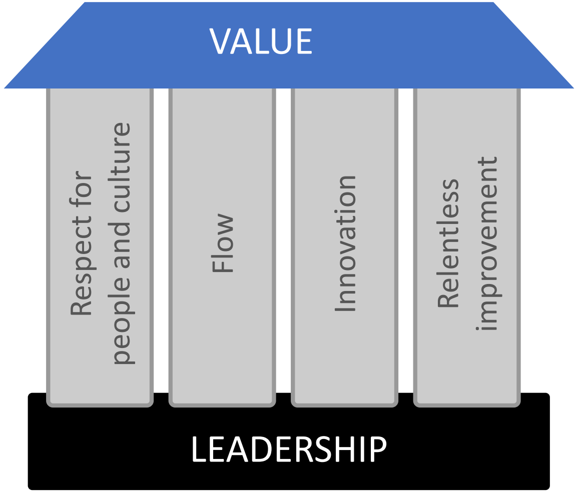
What it is they must do

- Embrace a Lean-Agile mindset
- Implement Lean-Agile practices
- Lead the implementation
- Get results

Embrace a Lean-Agile mindset

Embrace Lean-Agile values

House of Lean



Value in the shortest sustainable lead time

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.

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

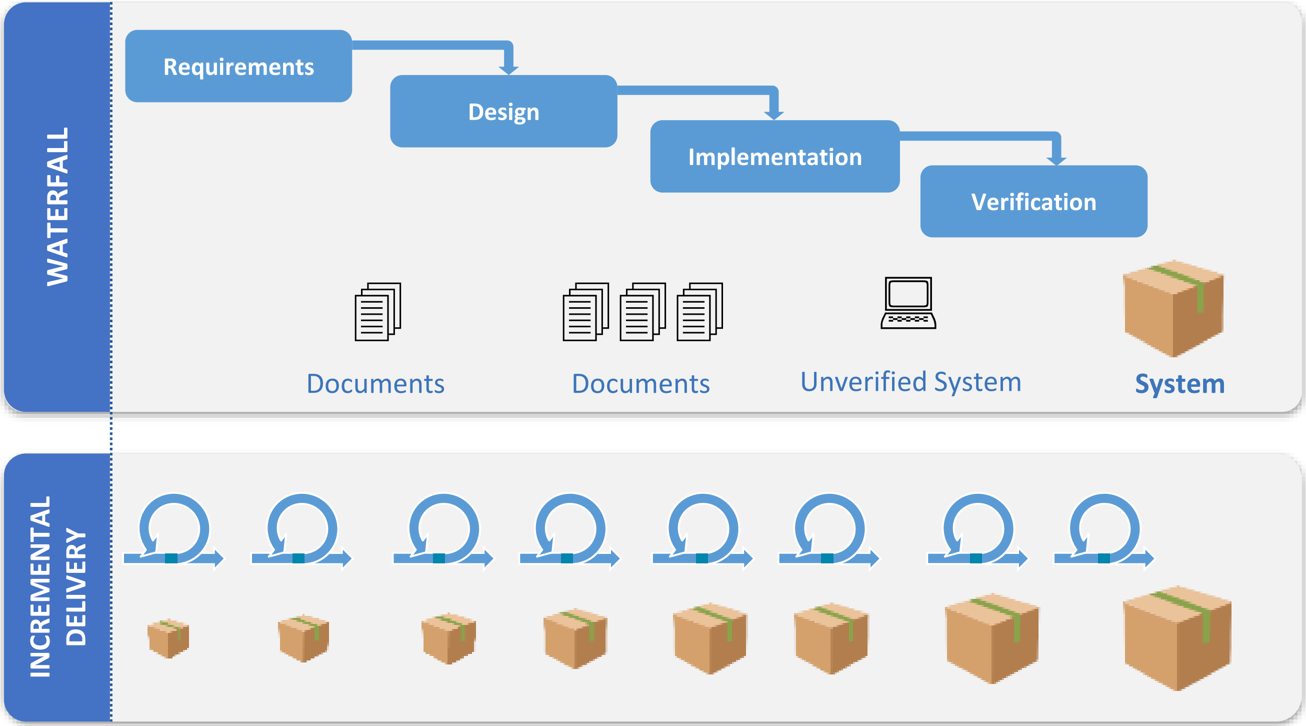
#6 - Visualize and limit WIP, reduce batch sizes, and manage queue lengths

#7 - Apply cadence, synchronize with cross-domain planning

#8 - Unlock the intrinsic motivation of knowledge workers

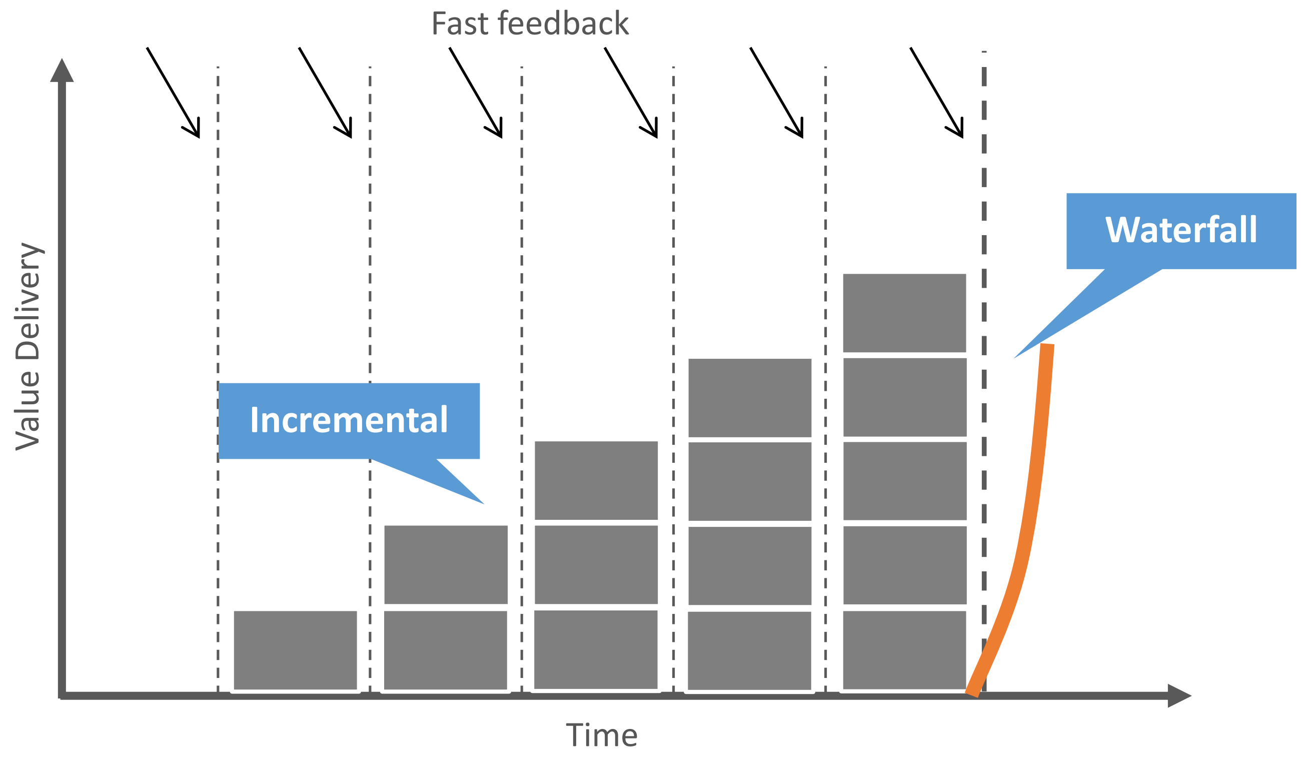
#9 - Decentralize decision-making

Building incrementally accelerates value delivery



And delivers better economics

Early delivery provides fast value with fast feedback



Implement Lean-Agile practices

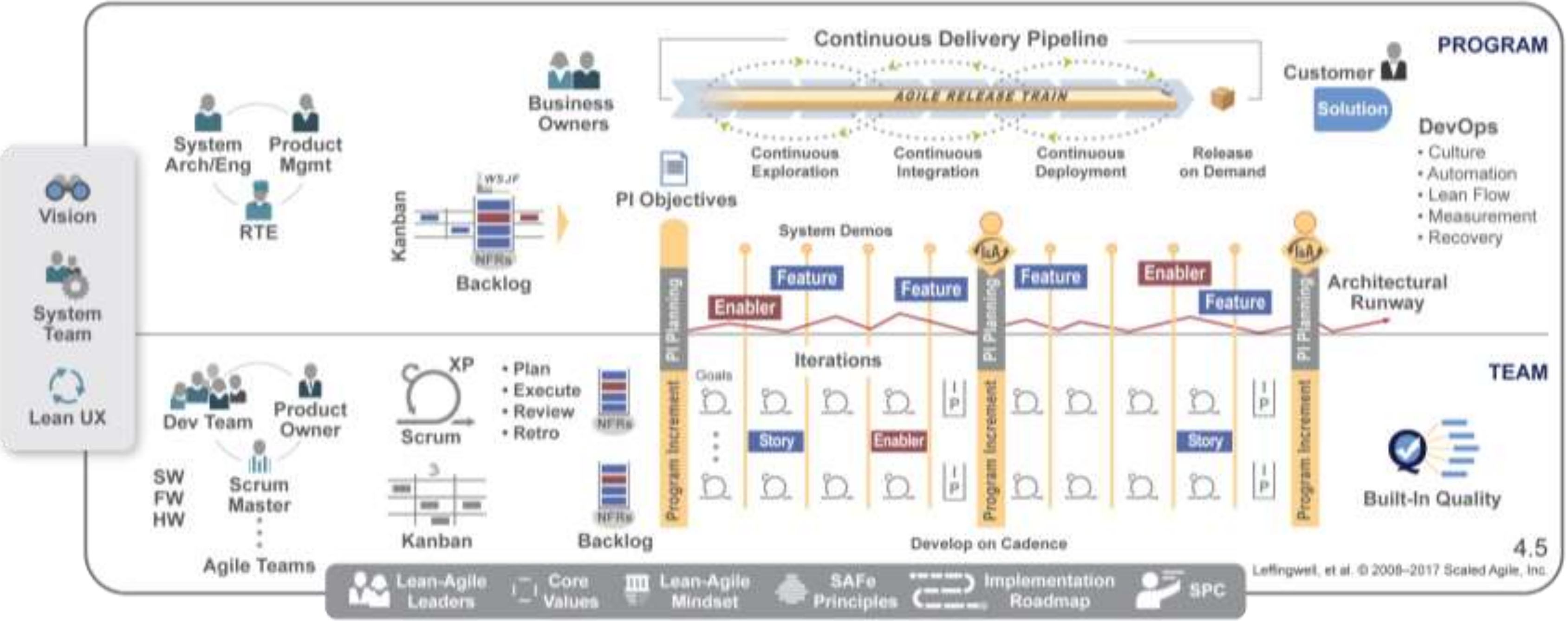
Knowledge for people building the world's most important systems

SAFe® is a freely revealed knowledge base of integrated, proven patterns for enterprise Lean-Agile development.

scaledagileframework.com

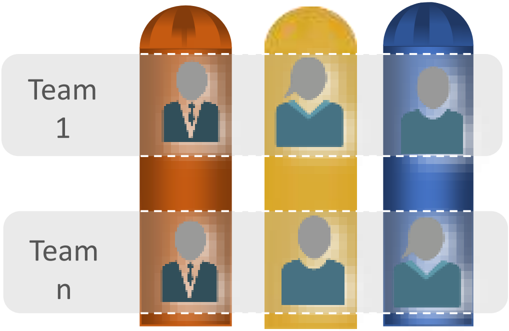
SCALED AGILE®

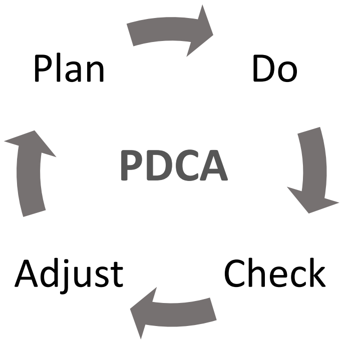
Essential SAFe provides the basis for success



Nothing beats an Agile Team

- Cross-functional, self-organizing entities that can define, build and test a thing of value
- Applies basic scientific practice: Plan—Do—Check—Adjust
- Delivers value every two weeks



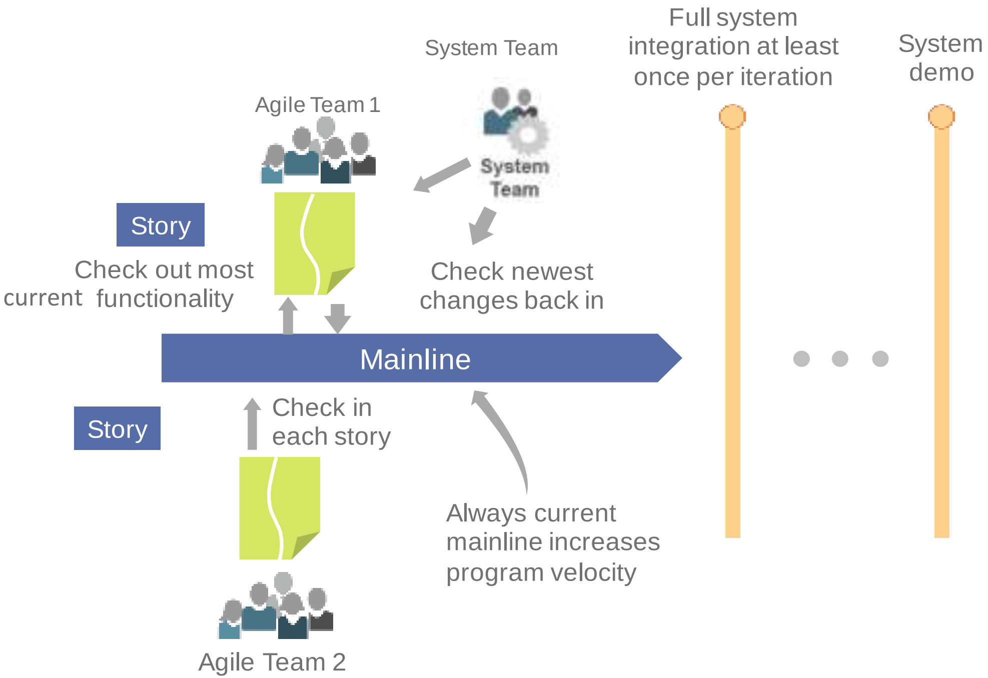


That integrates frequently

Integration points control product development.

— Dantar Oosterwal, The Lean Machine

- Avoid physical branching for software
- Frequently integrate hardware branches
- Use development by intention in for inter-team dependencies

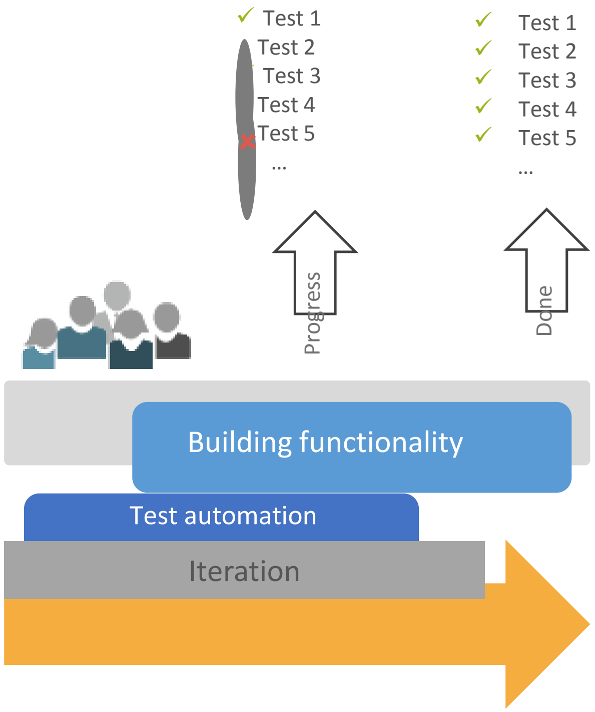


System Team
Full system integration at least once per iteration
System demo
Agile Team 1
System Team
Story
Check out most current functionality
Check newest changes back in
Mainline
Story
Check in each story
Always current mainline increases program velocity
Agile Team 2

Applies test automation

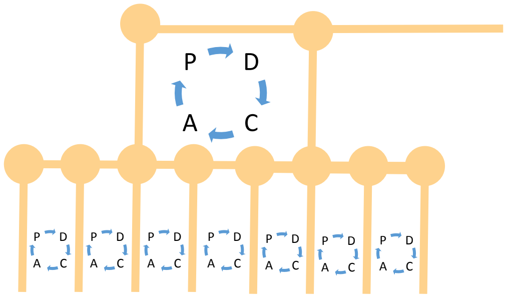
Test automation supports rapid regression testing

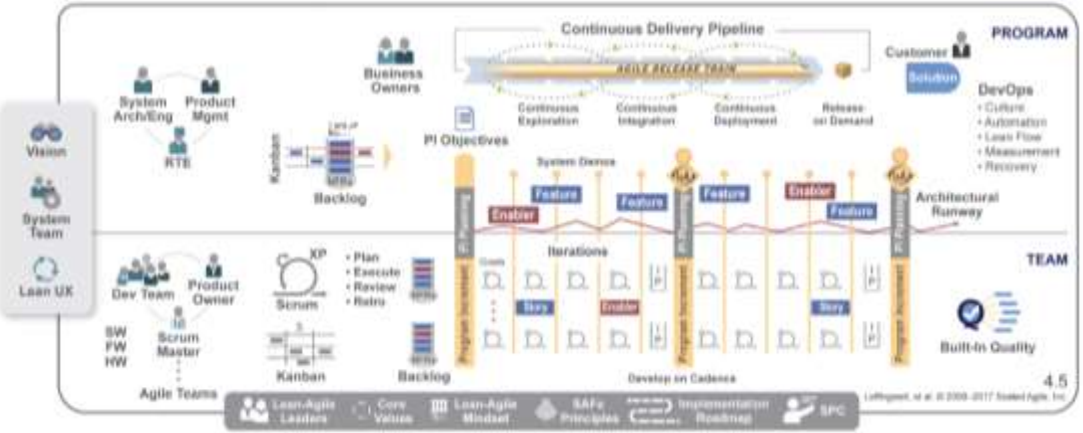
- Implemented in the same iteration
- Maintained under version control
- Passing vs. not-yet-passing and broken automated tests are the *real* iteration progress indicator



Except a team of Agile Teams

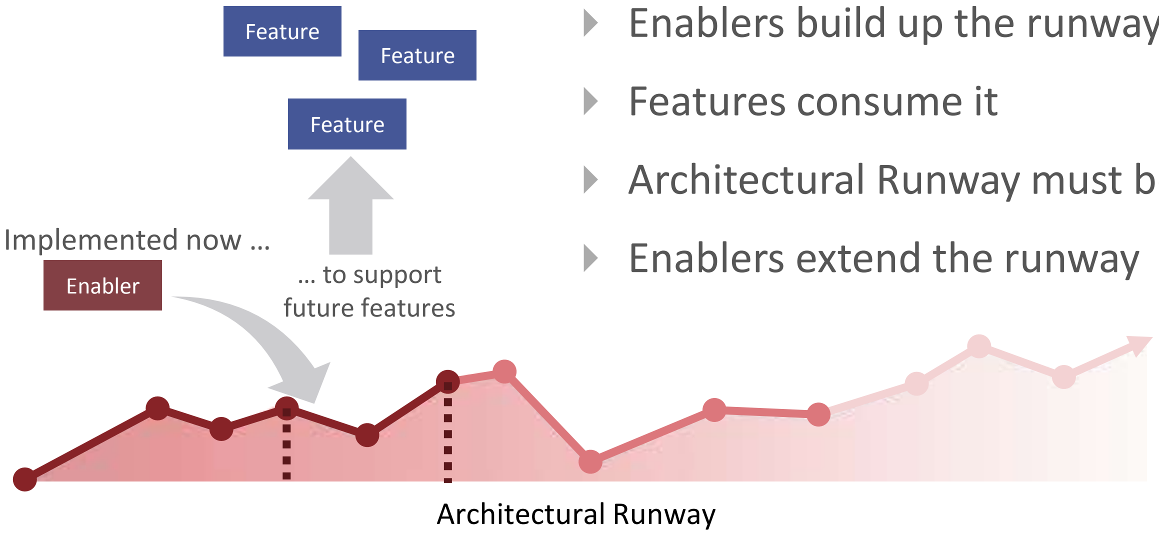
- Align 50-125 practitioners to a common mission
- Apply cadence and synchronization, Program Increments every 6-12 weeks
- Provide Vision, Roadmap, architectural guidance





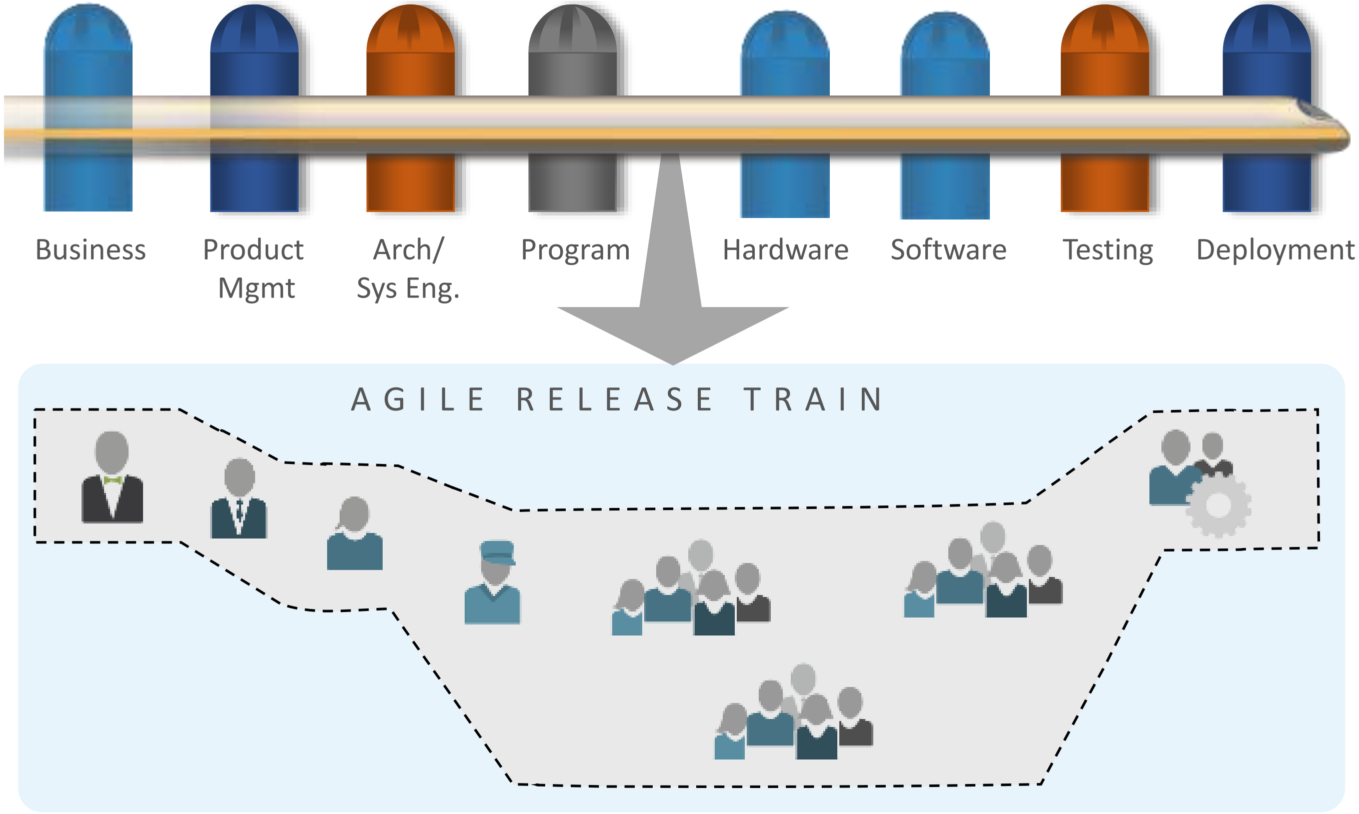
With some Architectural Runway

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
- Enablers extend the runway

Bringing together the necessary people



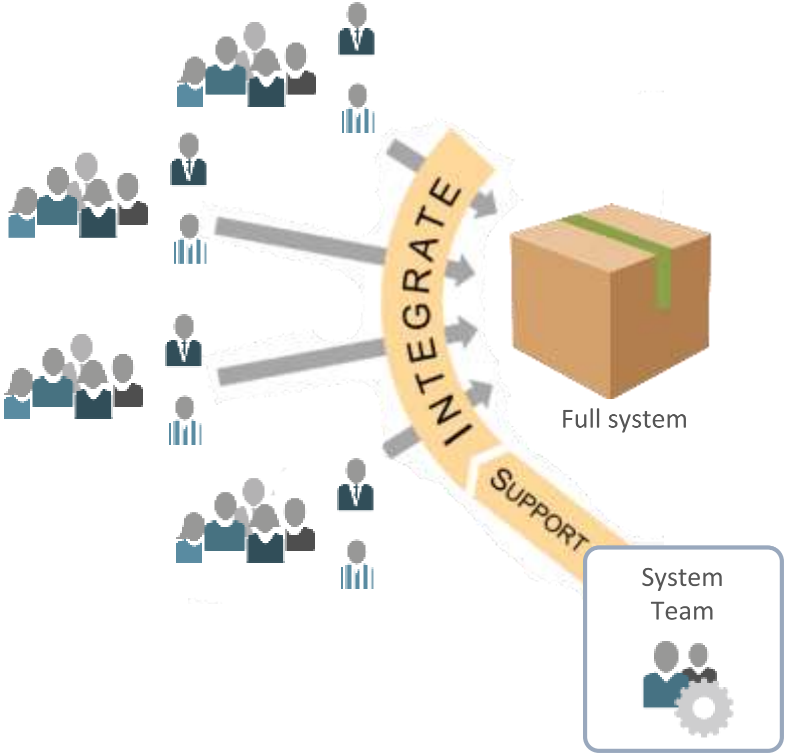
Synchronizes with PI Planning

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

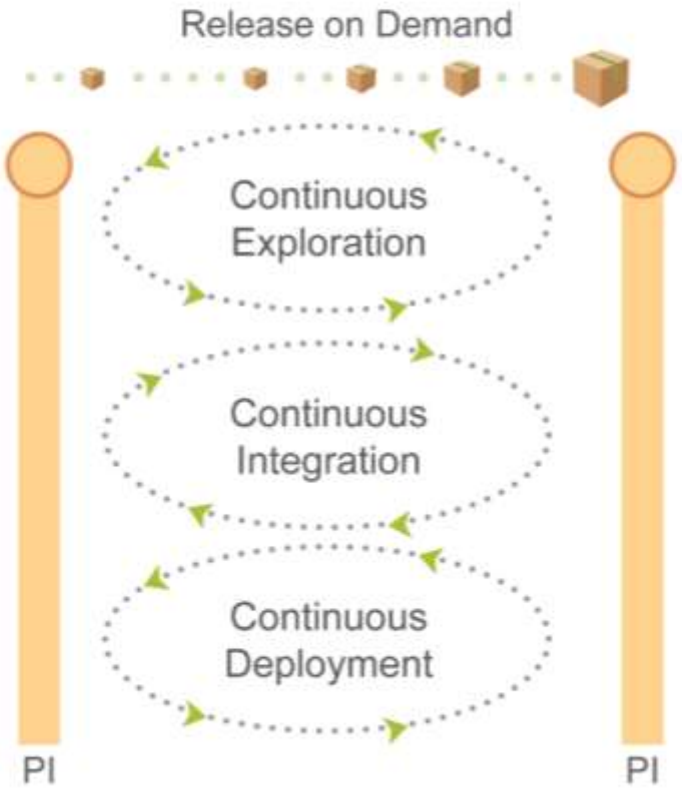
For a short video PI planning example, see: https://youtu.be/ZZAtl7nAB1M

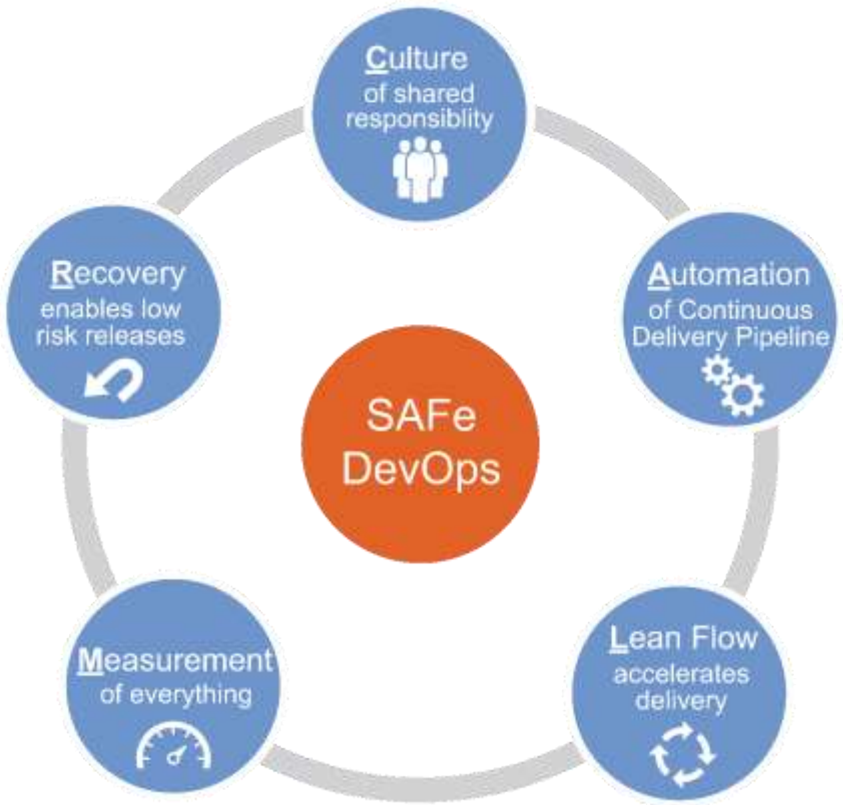
Demonstrates the full system every two weeks



- An integrated solution demo
- Objective milestone
- Demo from the staging environment, or the nearest proxy

Continuously delivers value to customers with DevOps




Culture of shared responsiblity
Automation of Continuous Delivery Pipeline
Lean Flow accelerates delivery
Measurement of everything
Recovery enables low risk releases
SAFe DevOps

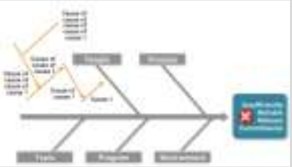
Inspects and Adapts every PI

Every PI, teams systematically address the larger impediments that are limiting velocity.

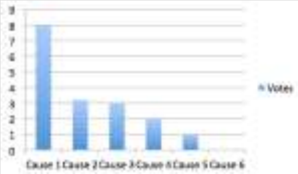
Agree on the problem to solve

Insufficiently reliable release commitments?

Apply root cause analysis (+ five whys)



Identify the biggest root cause using Pareto Analysis

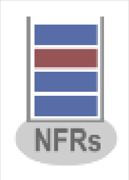


Restate the new problem for the biggest root cause

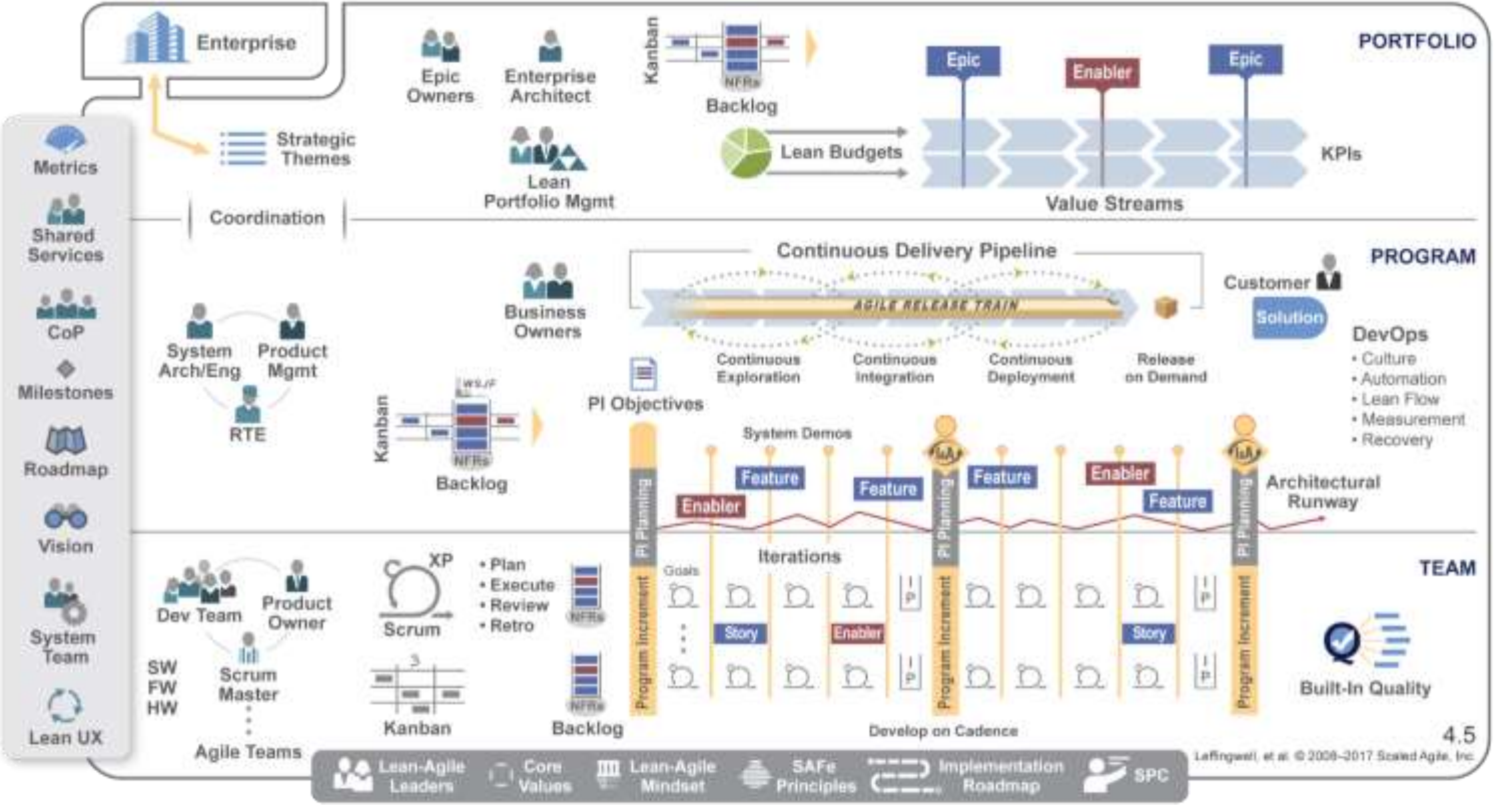
Insufficient architectural runway

Brainstorm solutions

Identify improvement Backlog items

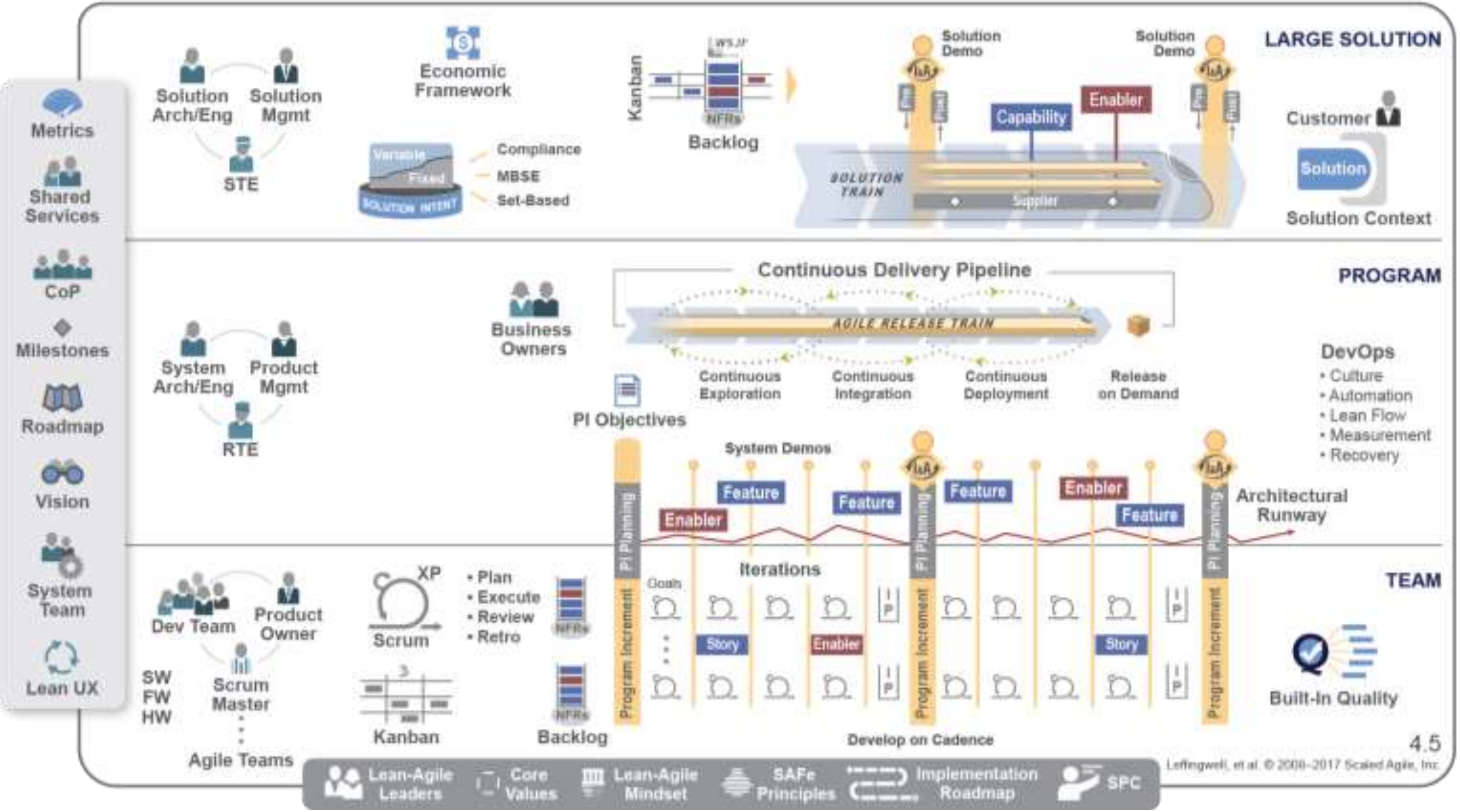


Portfolio SAFe aligns strategy and execution

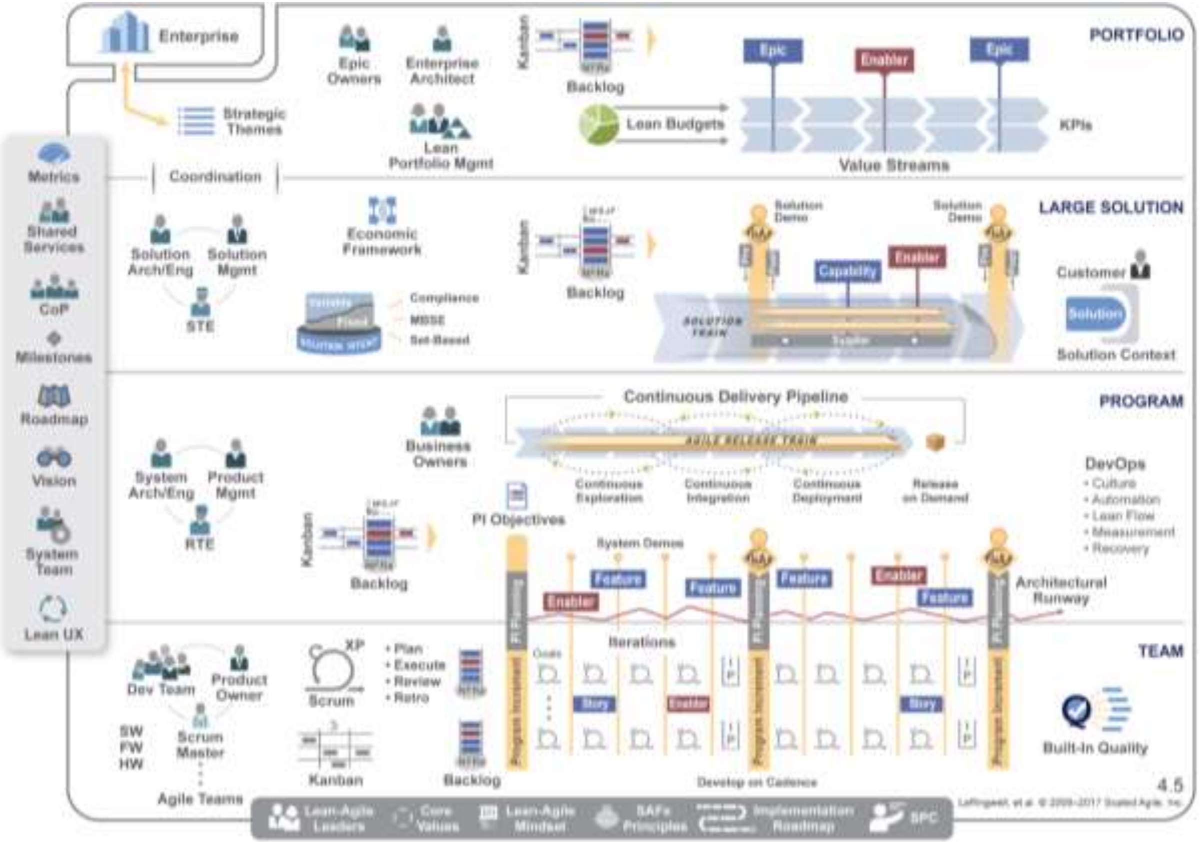


Leffingwell, et al. © 2008-2017 Scaled Agile, Inc.

Large Solution SAFe coordinates ARTs with a Solution Train

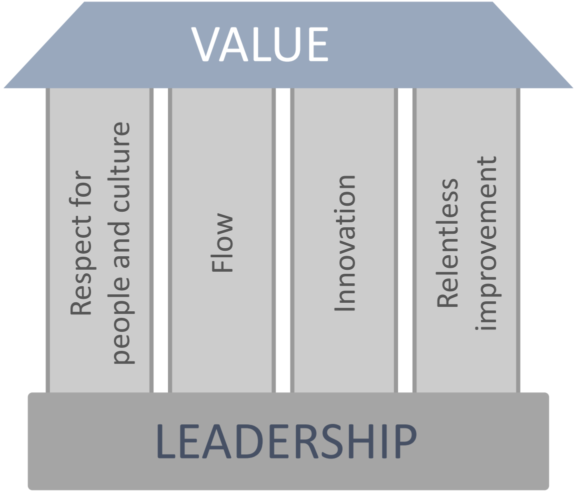


Full SAFe for large enterprises



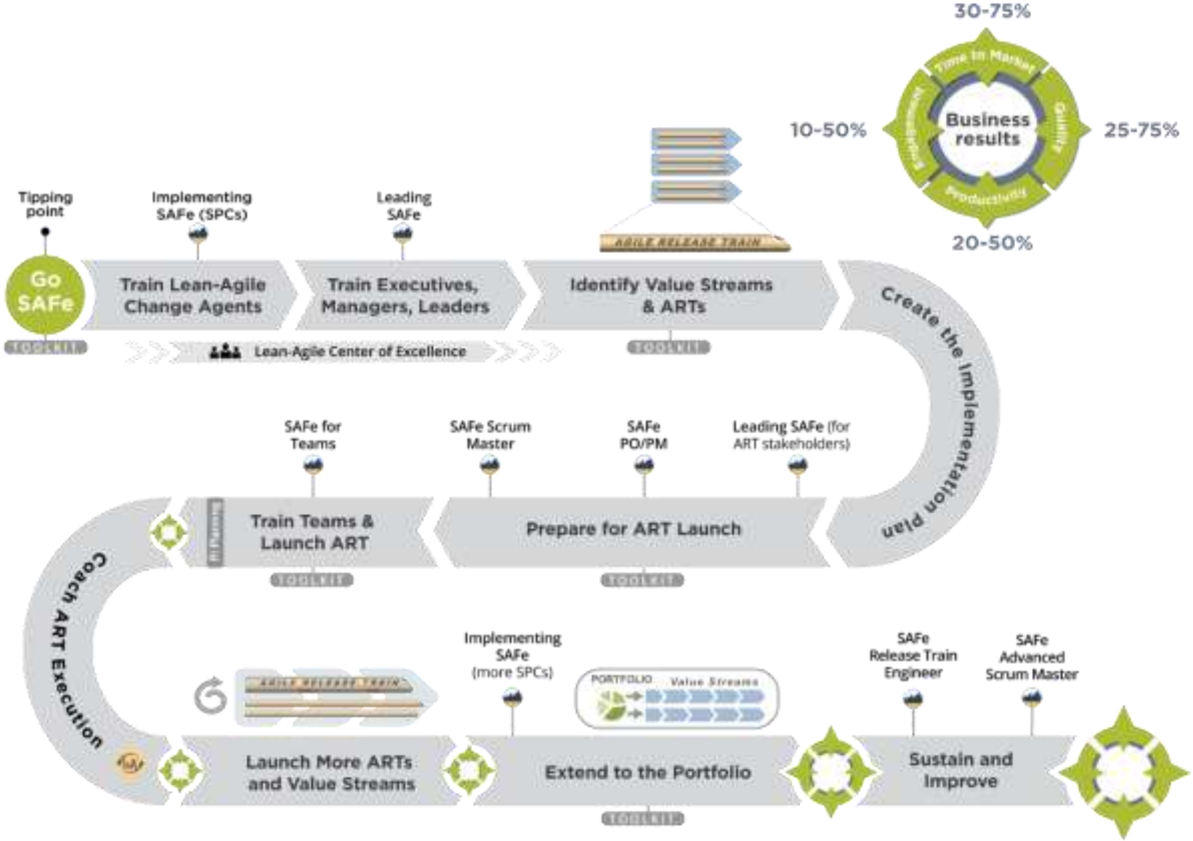
Lead the implementation

Leadership foundation



> *People are already doing their best; the problems are with the system. Only management can change the system.*
>
> *—W. Edwards Deming*

Implementation Roadmap



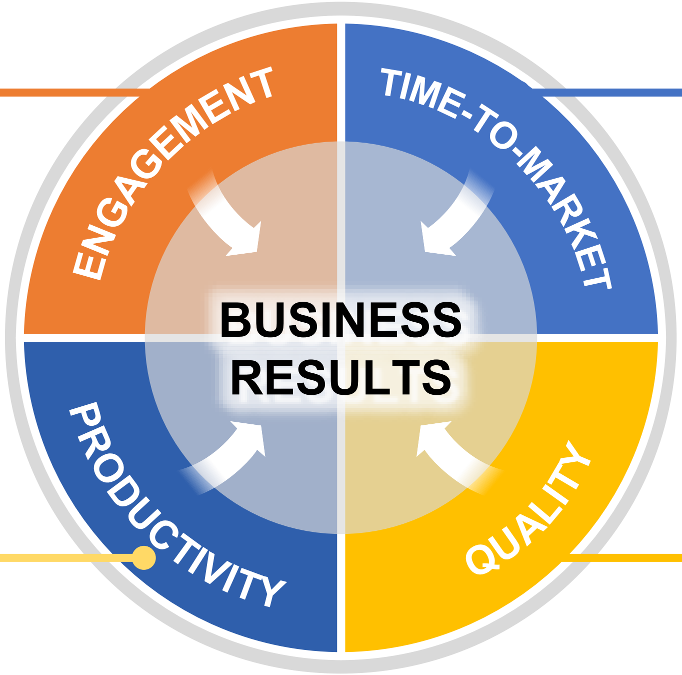
© 2008–2017 Scaled Agile, Inc.

Get results

Business results

10 – 50% happier, more motivated employees

30 – 75% faster time-to-market

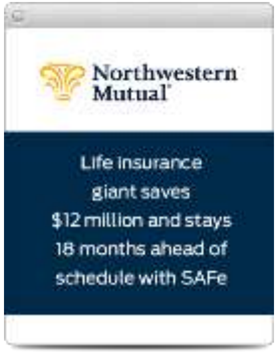


20 – 50% increase in productivity

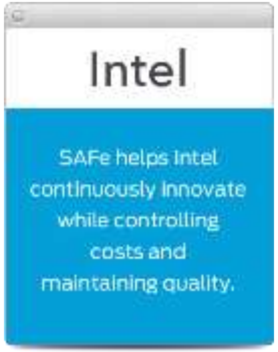
25 – 75% defect reduction

See ScaledAgileFramework.com/**case-studies**

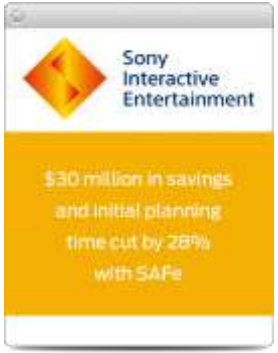
Financial Services / Electronics / Software / Telecom / Retail & Distribution / Government / Healthcare / Insurance / Medical Technology / Pharmaceutical / Media / Manufacturing / COTS Software / Customer Care & Billing / Outsourcing


Northwestern Mutual

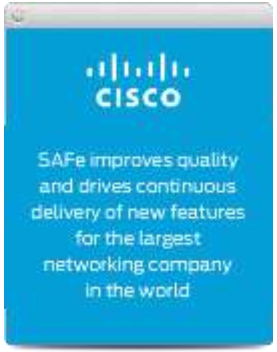
Life insurance giant saves $12 million and stays 18 months ahead of schedule with SAFe


Intel

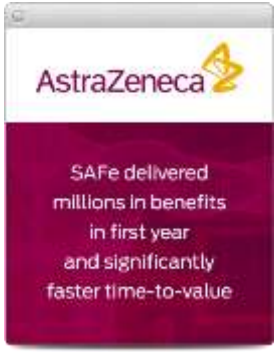
SAFe helps Intel continuously innovate while controlling costs and maintaining quality.


Sony Interactive Entertainment

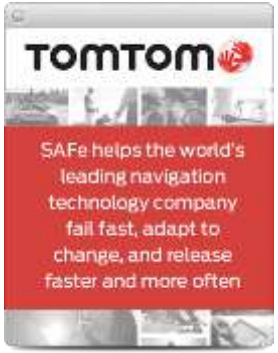
$30 million in savings and initial planning time cut by 28% with SAFe


CISCO

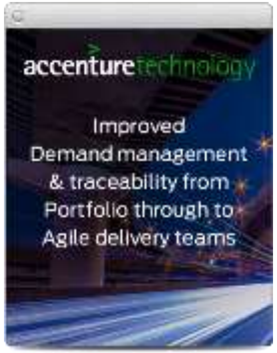
SAFe improves quality and drives continuous delivery of new features for the largest networking company in the world


AstraZeneca

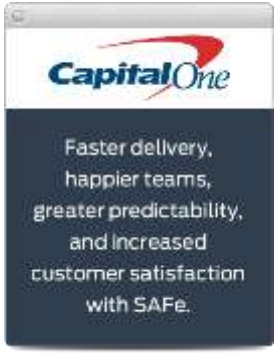
SAFe delivered millions in benefits in first year and significantly faster time-to-value


TOMTOM

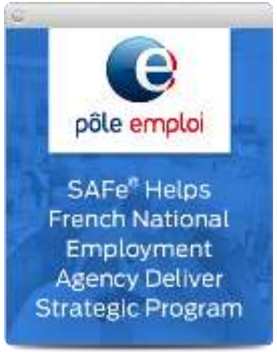
SAFe helps the world's leading navigation technology company fail fast, adapt to change, and release faster and more often


accenture technology

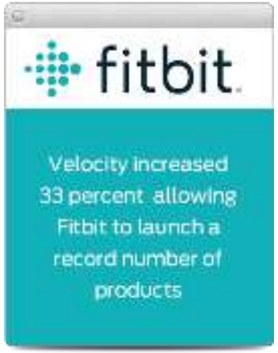
Improved Demand management & traceability from Portfolio through to Agile delivery teams


Capital One

Faster delivery, happier teams, greater predictability, and increased customer satisfaction with SAFe.


pôle emploi

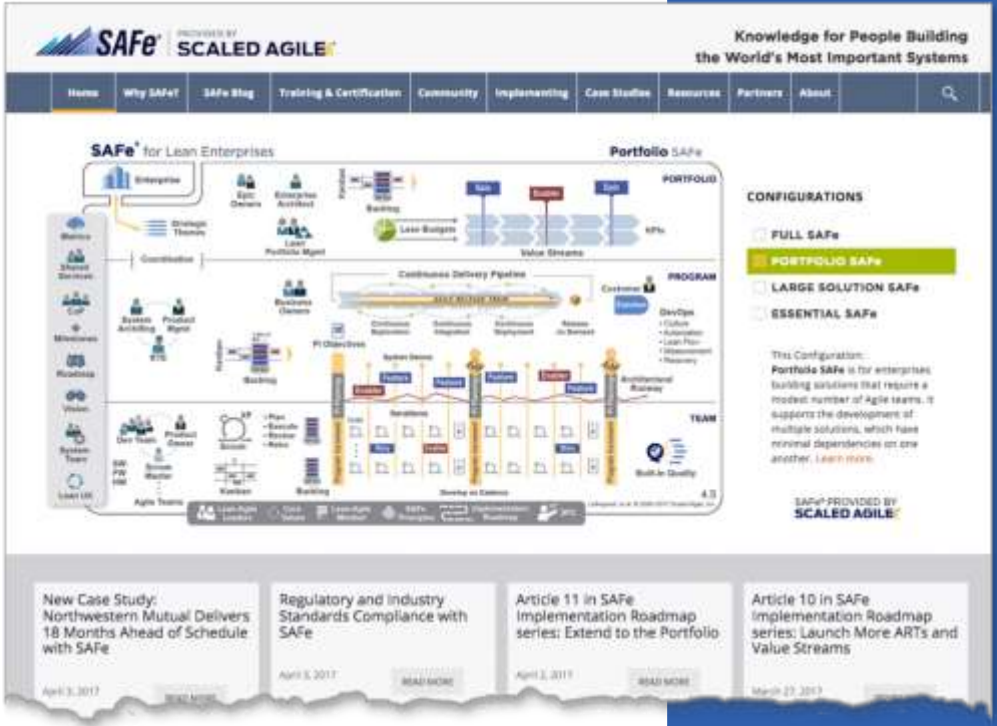
SAFe® Helps French National Employment Agency Deliver Strategic Program


fitbit

Velocity increased 33 percent allowing Fitbit to launch a record number of products

See ScaledAgileFramework.com/**case-studies**

Gain the Knowledge



Knowledge for People Building
the World's Most Important Systems

CONFIGURATIONS

FULL SAFe

PORTFOLIO SAFe

LARGE SOLUTION SAFe

ESSENTIAL SAFe

This Configuration:
Portfolio SAFe is for enterprises building solutions that require a modest number of Agile teams. It supports the development of multiple solutions, which have minimal dependencies on one another. Learn more

SAFe® PROVIDED BY SCALED AGILE

New Case Study: Northwestern Mutual Delivers 18 Months Ahead of Schedule with SAFe
April 3, 2017 READ MORE

Regulatory and Industry Standards Compliance with SAFe
April 3, 2017 READ MORE

Article 11 in SAFe Implementation Roadmap series: Extend to the Portfolio
April 2, 2017 READ MORE

Article 10 in SAFe Implementation Roadmap series: Launch More ARTs and Value Streams
March 27, 2017

ScaledAgileFramework.com

Explore the SAFe knowledge base and find free resources:

- Articles
- Guidance
- Presentations
- White papers
- Videos
- Case studies

ScaledAgile.com

Find SAFe

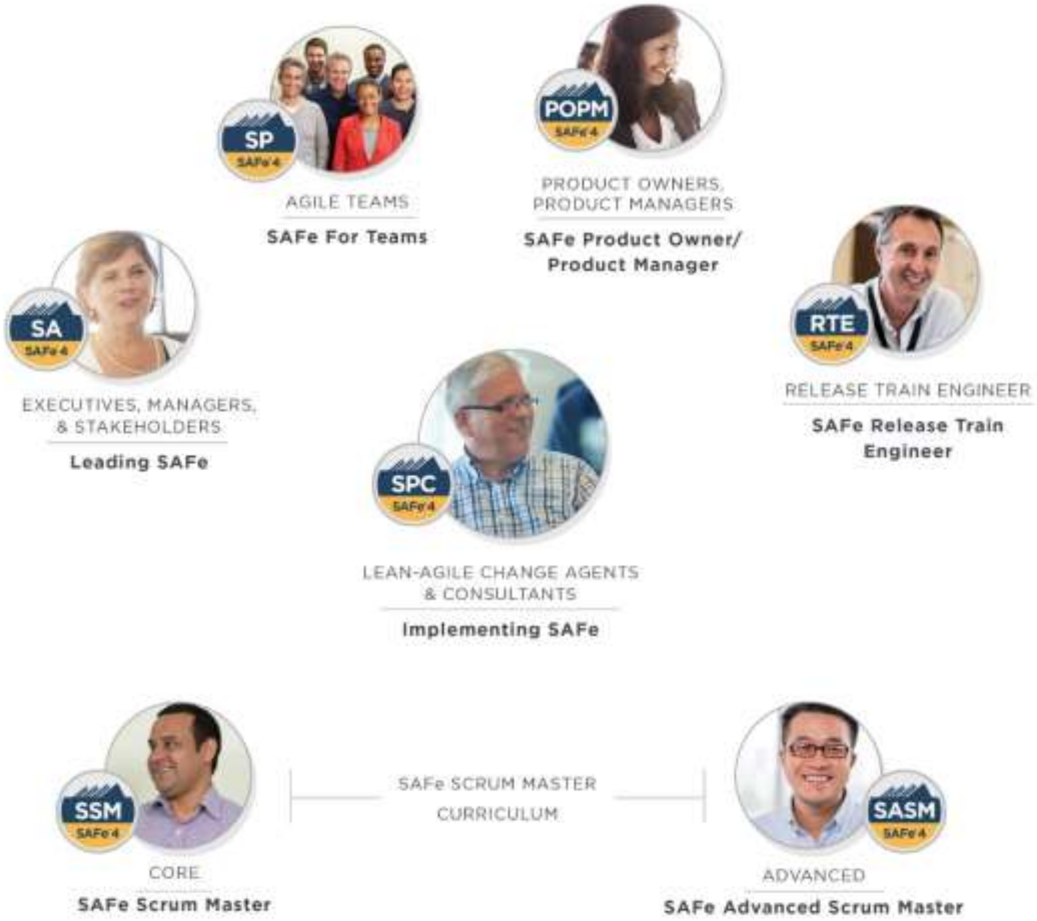
training worldwide

CORE

- Leading SAFe
- SAFe for Teams
- SAFe Scrum Master
- SAFe PO/PM

ADVANCED

- SAFe Advanced Scrum Master
- Implementing SAFe
- SAFe Release Train Engineer



SP SAFe 4
AGILE TEAMS
SAFe For Teams

POPM SAFe 4
PRODUCT OWNERS, PRODUCT MANAGERS
SAFe Product Owner/ Product Manager

SA SAFe 4
EXECUTIVES, MANAGERS, & STAKEHOLDERS
Leading SAFe

RTE SAFe 4
RELEASE TRAIN ENGINEER
SAFe Release Train Engineer

SPC SAFe 4
LEAN-AGILE CHANGE AGENTS & CONSULTANTS
Implementing SAFe

SSM SAFe 4
CORE
SAFe Scrum Master

SAFe SCRUM MASTER CURRICULUM

SASM SAFe 4
ADVANCED
SAFe Advanced Scrum Master

SAFe® for Lean Enterprises

180,000

SAFe certified professionals in 100+ countries

150

Scaled Agile Partners in 50 countries

70% US *Fortune* 100 enterprises have SAFe certified professionals

2 million

Annual visitors to SAFe and Scaled Agile websites

PLEDGE 1%

Pledged 1%

Scaled Agile stock equity & employee time to Pledge 1% campaign

SAFe:

Freely available knowledge base, downloads, and resources for people building the world’s most important software and systems

SAFe SUMMIT

Freely Available

SAFe’s knowledge base is freely available at **scaledagileframework.com**

Configurable

SAFe is able to accommodate enterprises of all sizes and industries

Fastest Growing Method

- **11th Annual State of Agile Report by VersionOne**
- **2017 Scaling Agile Report by cPrime**

SAFe cited as preferred solution for scaling Agile, making SAFe the most popular scaling method above Scrum, Scrum of Scrums, and all other frameworks

SAFe® | PROVIDED BY SCALED AGILE®

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

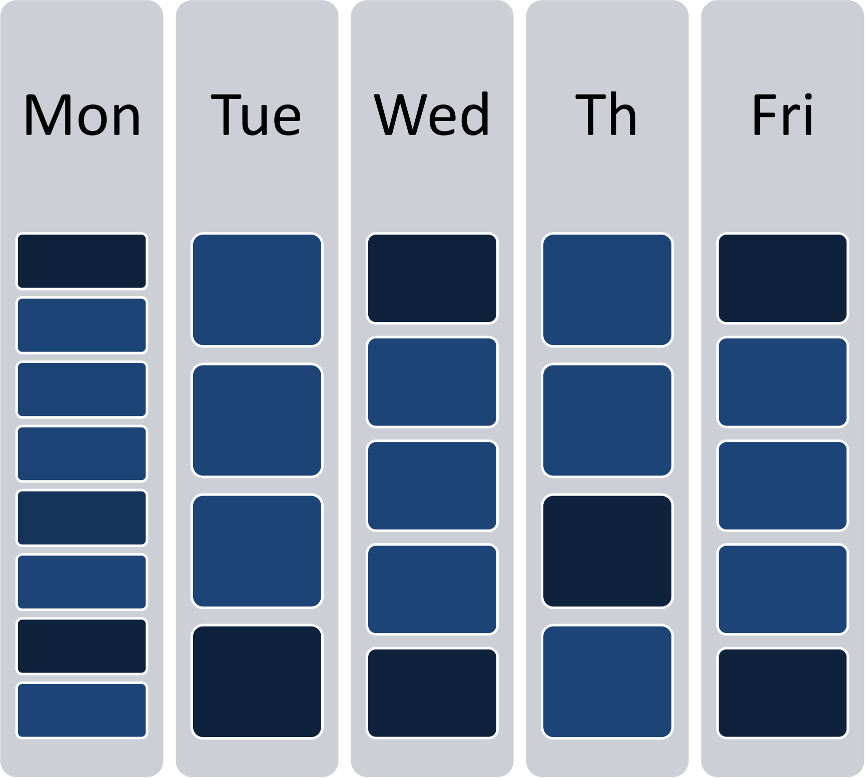
#7 - Apply cadence, synchronize with cross-domain planning

#8 - Unlock the intrinsic motivation of knowledge workers

#9 - Decentralize decision-making

Used with permission.

Utilization is the Wrong Goal



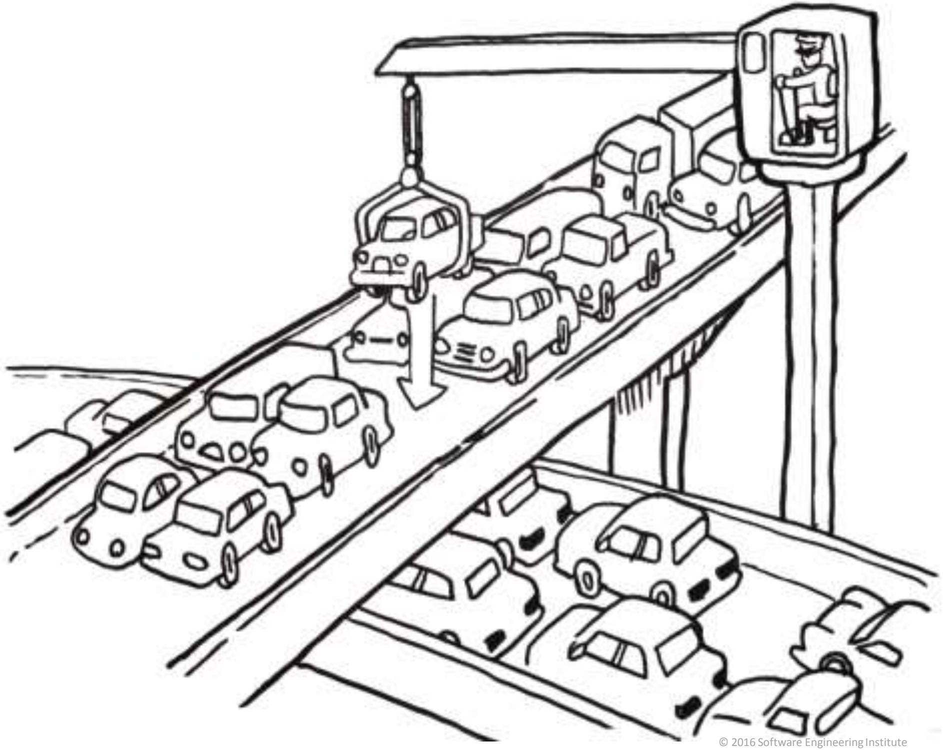
100% Utilization:

- Magnifies the impact of variation
- Maximizes task-switching overhead
- Assures slower overall progress

Change is inevitable, plan to learn

Multi-tasking is a myth we don’t accurately comprehend

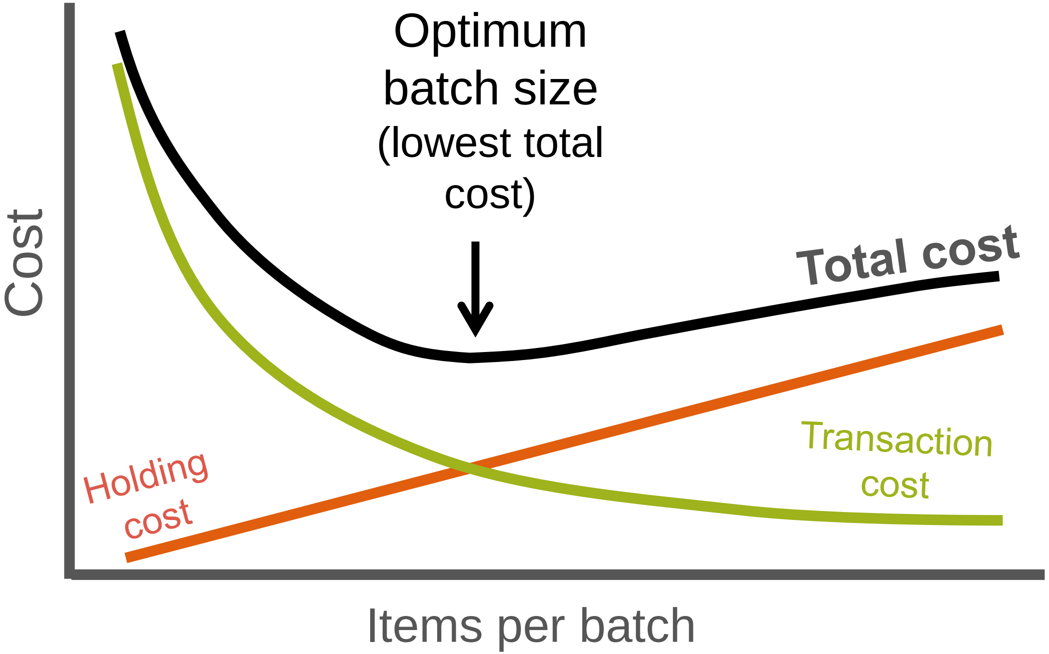
Maximum Utilization is Counterproductive



© 2016 Software Engineering Institute

Finding optimum batch size

Optimum batch size is an example of a U-curve optimization.

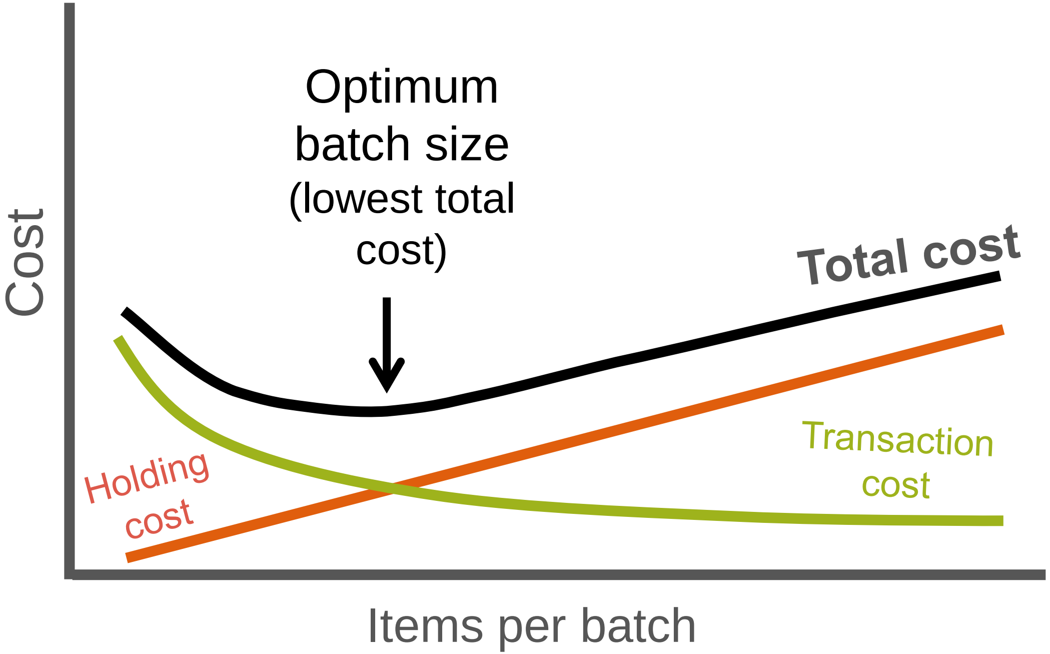


Principles of Product Development Flow, Don Reinertsen

- *Total* costs are the sum of holding costs and transaction costs
- Higher transaction costs shift optimum batch size higher
- Higher holding costs shift batch size lower

Reducing optimum batch size

Reducing transaction costs reduces total costs, and shifts optimum batch size lower.

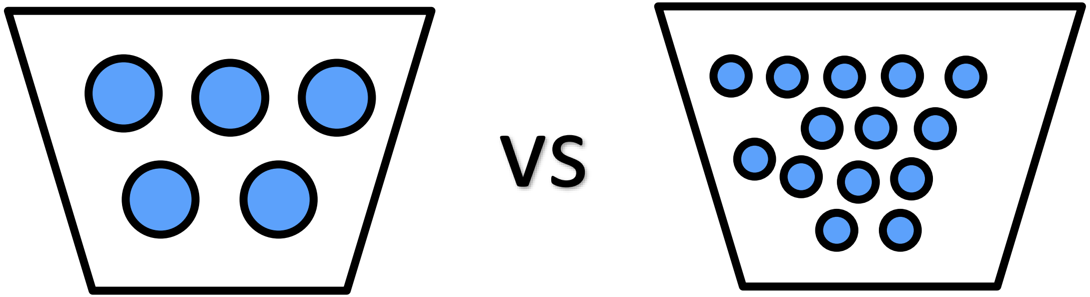


Principles of Product Development Flow, Don Reinertsen

- Reducing batch size:
  - Increases predictability
  - Accelerates feedback
  - Reduces rework
  - Lowers cost
- Batch size reduction probably saves twice what you think

Reducing transaction costs example
https://youtu.be/RRy_73ivcms
2:09

Story Splitting is an Enabler of Smaller Batch Size Too



Splitting stories requires engineering judgment

Besides Longer Cycle Time, Queues are Just Generally Bad

The Principle of Queueing Waste: Queues are the root cause of the majority of economic waste in product development.

Queues create:

- Longer Cycle Time
- Increased Risk
- More Variability
- More Overhead
- Lower Quality
- Less Motivation

Principles of Product Development Flow,
Don Reinertsen

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

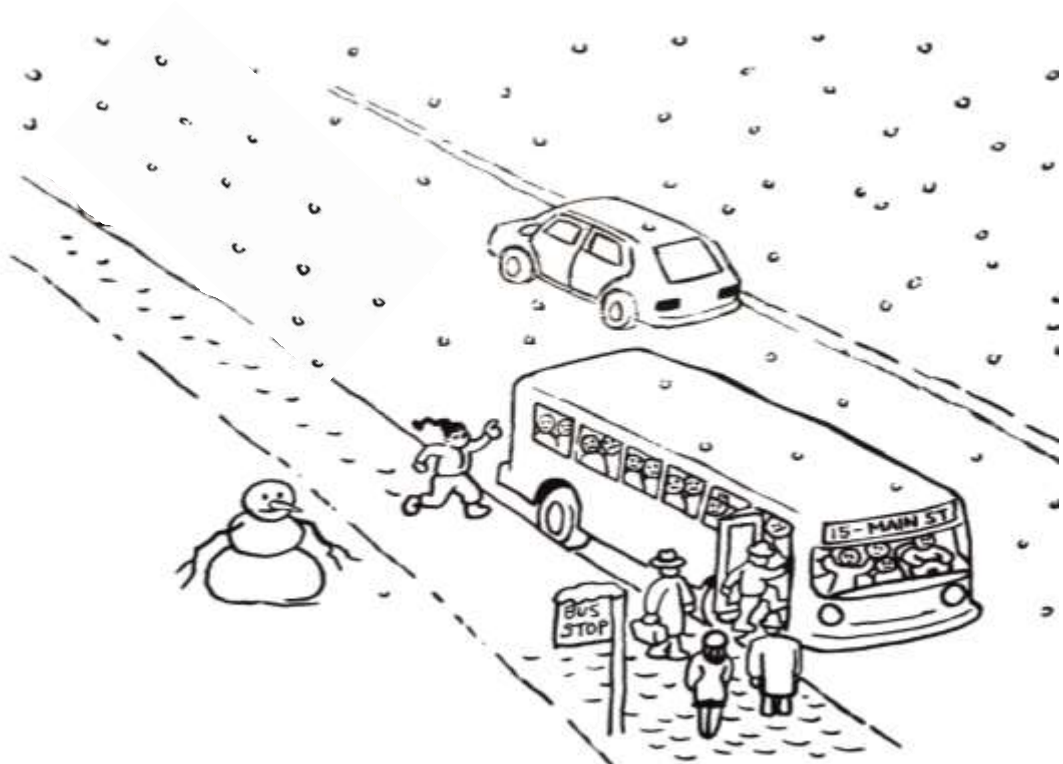
→ #7 - Apply cadence, synchronize with cross-domain planning

#8 - Unlock the intrinsic motivation of knowledge workers

#9 - Decentralize decision-making

Used with permission.

Cadence Enhances Predictability



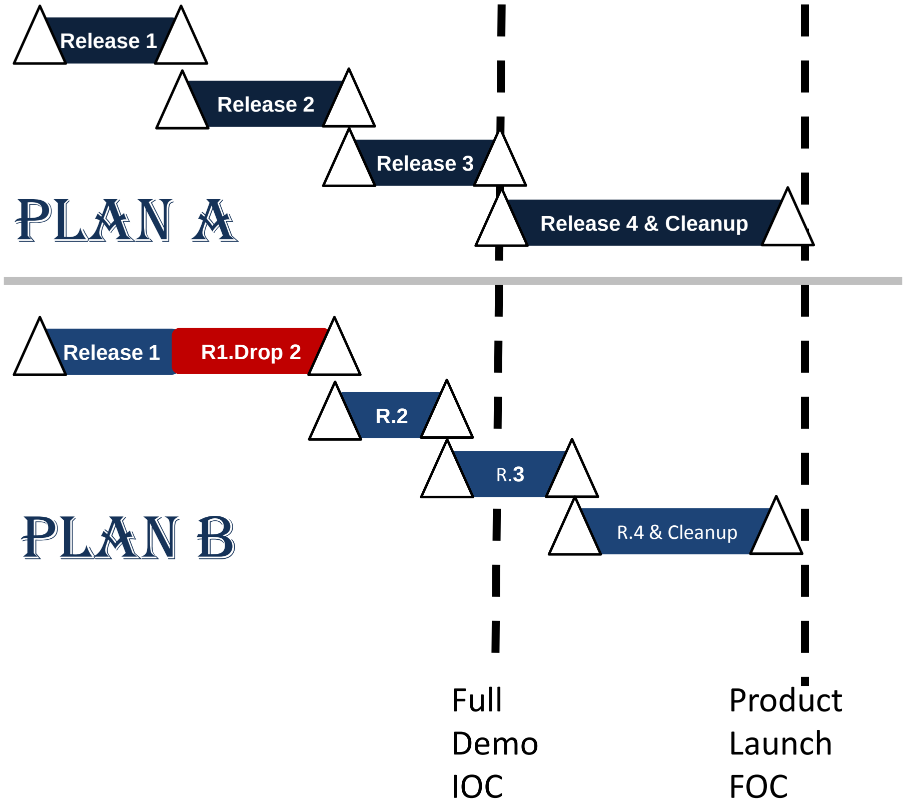
© 2016 Software Engineering Institute

A Late Bus:

- Makes people scramble to get aboard
- They don't know when the next one will get here

Then the next bus comes along empty

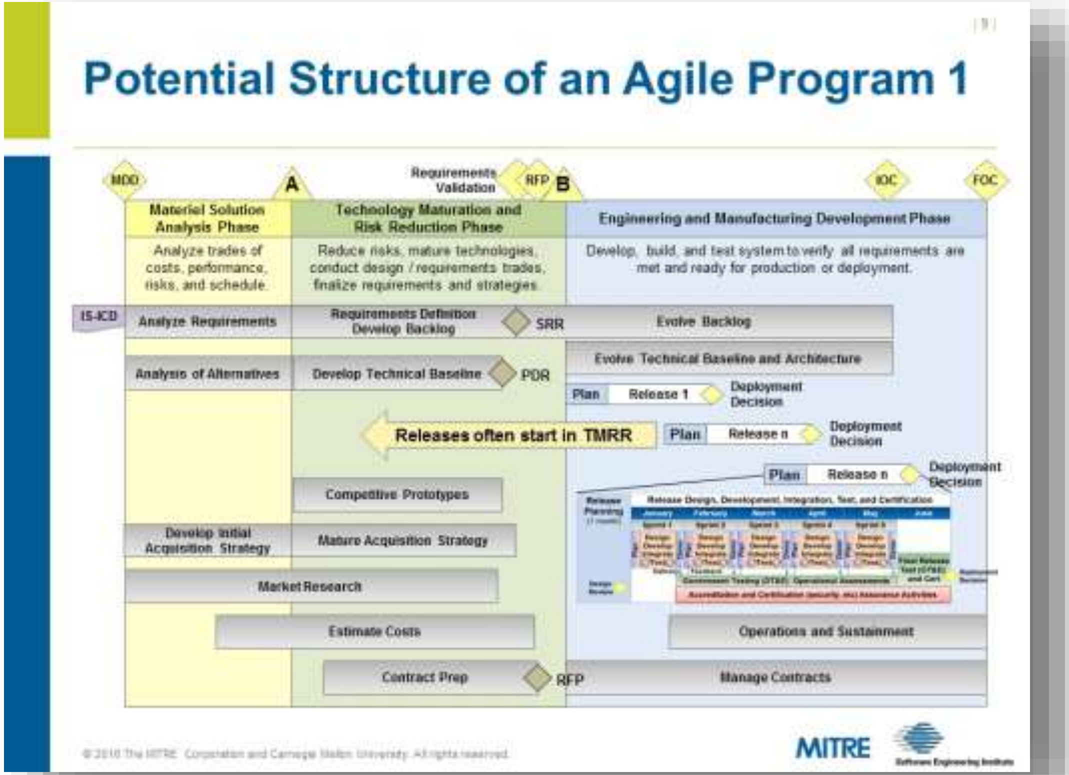
Late Releases Become "Feature Magnets"



As things start to slip

- Influential people get 'their priorities' moved up, rather than deferred
- Pressure increases on early releases
- Functions slated for final release can't be guaranteed…

How SAFe Might Translate into a DoD Acquisition Environment



Potential Structure of an Agile Program 1

MDD

A

Requirements Validation

RFP

B

IOC

FOC

Material Solution Analysis Phase

Analyze trades of costs, performance, risks, and schedule.

Technology Maturation and Risk Reduction Phase

Reduce risks, mature technologies, conduct design / requirements trades, finalize requirements and strategies.

Engineering and Manufacturing Development Phase

Develop, build, and test system to verify all requirements are met and ready for production or deployment.

IS-ICD

Analyze Requirements

Requirements Definition Develop Backlog

SRR

Evolve Backlog

Evolve Technical Baseline and Architecture

Analysis of Alternatives

Develop Technical Baseline

PDR

Plan | Release 1 | Deployment Decision

Releases often start in TMRR

Plan | Release n | Deployment Decision

Plan | Release n | Deployment Decision

Release Planning (1 month)

Release Design, Development, Integration, Test, and Certification

January | February | March | April | May | June

Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Sprint 5

Design Develop Integrate (Test)

Final Release Test (OT&E) and Cert

Deployment Decision

Design Review

Government Testing (DT&E), Operational Assessments

Accreditation and Certification (security, etc) Assurance Activities

Competitive Prototypes

Develop Initial Acquisition Strategy

Mature Acquisition Strategy

Market Research

Estimate Costs

Operations and Sustainment

Contract Prep

RFP

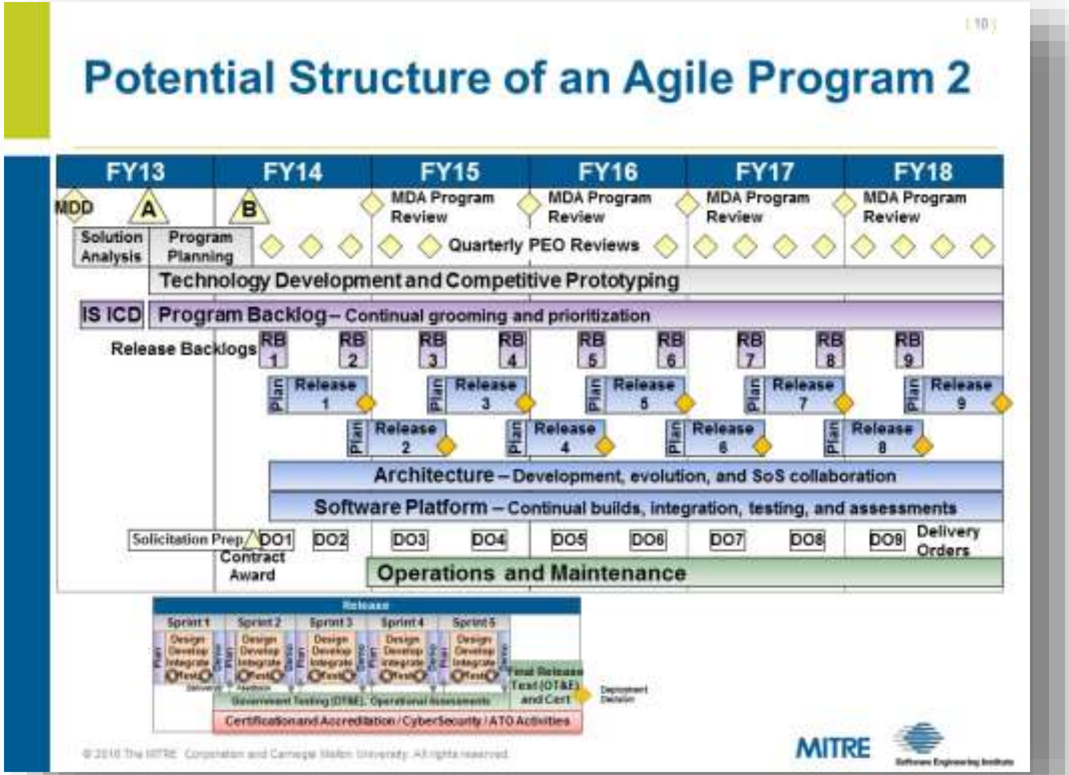
Manage Contracts

© 2018 The MITRE Corporation and Carnegie Mellon University. All rights reserved.

MITRE

Software Engineering Institute

A More Detailed Look at a Possible Agile Implementation in DoD



[10]

Potential Structure of an Agile Program 2

FY13 FY14 FY15 FY16 FY17 FY18

MDD A B

MDA Program Review

Solution Analysis

Program Planning

Quarterly PEO Reviews

Technology Development and Competitive Prototyping

IS ICD

Program Backlog – Continual grooming and prioritization

Release Backlogs RB 1 RB 2 RB 3 RB 4 RB 5 RB 6 RB 7 RB 8 RB 9

Plan Release 1 · Plan Release 2 · Plan Release 3 · Plan Release 4 · Plan Release 5 · Plan Release 6 · Plan Release 7 · Plan Release 8 · Plan Release 9

Architecture – Development, evolution, and SoS collaboration

Software Platform – Continual builds, integration, testing, and assessments

Solicitation Prep

Contract Award

DO1 DO2 DO3 DO4 DO5 DO6 DO7 DO8 DO9 Delivery Orders

Operations and Maintenance

Release

Sprint 1 Sprint 2 Sprint 3 Sprint 4 Sprint 5

Plan Design Develop Integrate Test Demo

Delivery Feedback

Government Testing (DT&E), Operational Assessments

Final Release Test (OT&E) and Cert

Deployment Decision

Certification and Accreditation / CyberSecurity / ATO Activities

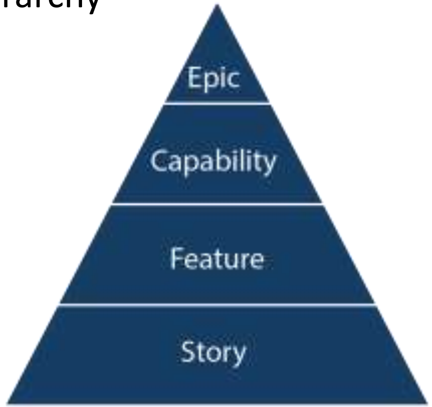
© 2018 The MITRE Corporation and Carnegie Mellon University. All rights reserved.

MITRE

Software Engineering Institute

How do we think & talk about requirements?

SAFe Requirements Hierarchy



Typical hierarchy (from SAFe, in this case):

- Epic – could be analog to contract-level requirements
- Capability – could be analog to System Level requirements
- Feature– could be analog to software capability requirements
- Story – could be analog to software component level requirements or below

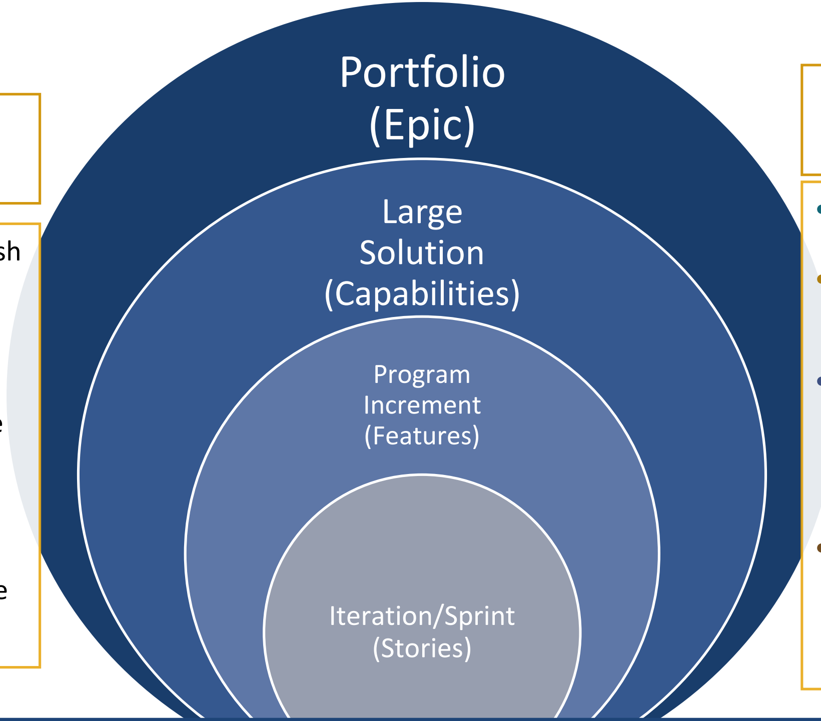
One of the decisions to make is how different levels of requirements will be treated

- One dependency is how the software part of the program interacts with systems engineering/other stakeholders
- Another criterion is how requirements change will be accommodated
  - Level at which allocated baseline is established is crucial to having appropriate flexibility for requirements evolution

Addressing Requirements at Multiple Levels (SAFe Terminology)

Issues in Expressing Requirements

- **Portfolio**: Conops level, trying to establish Business/Enabling **Epics**
- **Program/Large Solution**: moving from "shall" statements to **Capabilities**
- **Release**: Decomposing Capabilities into meaningful **Features** that are executable in a few iterations; translating Features into User & Enabling **Stories** that can be allocated to iterations (sprints)
- **Iteration**: "slicing" **Stories** in such a way that meaningful working software can be produced in short (2-3 week) iterations



Issues in Governing Requirements

- **Portfolio**: Assuring that the value stream is representative of operations
- **Large Solution**: assuring that acquisition and users or their representatives are engaged and relevant
- **Release**: Assuring that Product Managers (or Chief Product Owners) are actively engaged in refining and prioritizing stories and features ahead of the development teams
- **Iteration**: Assuring that Product Owners appropriately represent user needs and management goals when interacting with development teams

Where should acquisition program offices be *controlling* and/or *participating?*

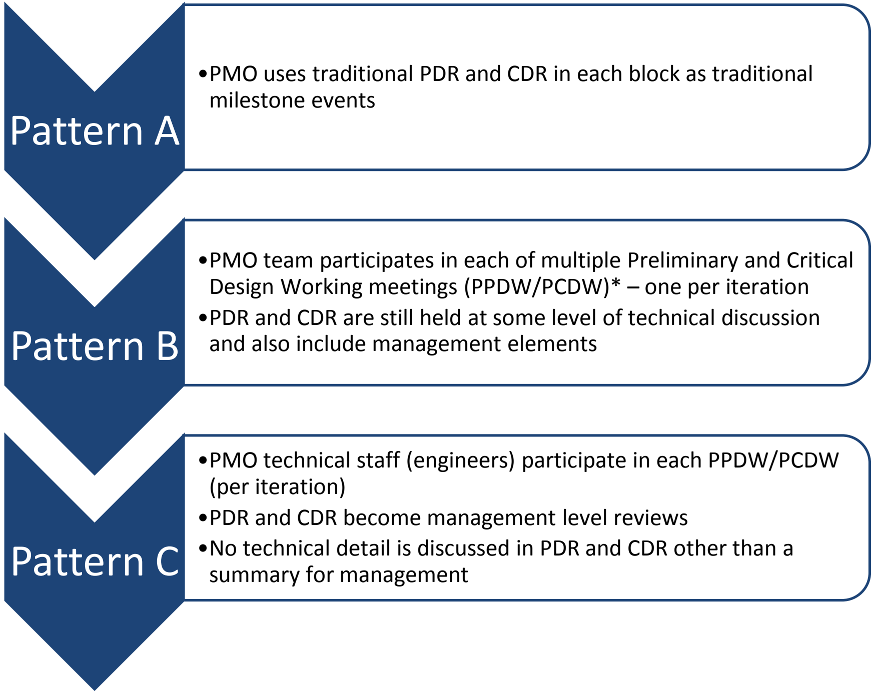
One of Top Questions SEI Hears about Agile

How do I accommodate Technical Reviews like PDR (preliminary design review), CDR (critical design review), etc.?

- Especially if contract was formulated as traditional and program office or developer wants to use Agile after the fact

"LOW CEREMONY"
"SEMI-FORMAL"
"HIGH CEREMONY"
INCREMENTAL REVIEW FORMALITY

S3 Patterns in Agile Settings for PDR, CDR Design/Execution



Pattern A

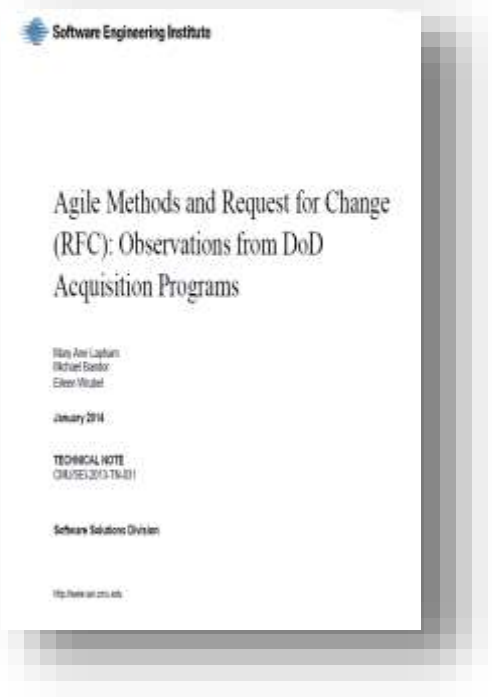
- PMO uses traditional PDR and CDR in each block as traditional milestone events

Pattern B

- PMO team participates in each of multiple Preliminary and Critical Design Working meetings (PPDW/PCDW)* – one per iteration
- PDR and CDR are still held at some level of technical discussion and also include management elements

Pattern C

- PMO technical staff (engineers) participate in each PPDW/PCDW (per iteration)
- PDR and CDR become management level reviews
- No technical detail is discussed in PDR and CDR other than a summary for management



Software Engineering Institute

Agile Methods and Request for Change (RFC): Observations from DoD Acquisition Programs

Mary Ann Lapham
Michael Bandor
Eileen Wrubel

January 2014

TECHNICAL NOTE
CMU/SEI-2013-TN-031

Software Solutions Division

http://www.sei.cmu.edu

*PPDW=Partial Preliminary Design Walkthrough;
PCDW=Partial Critical Design Walkthrough

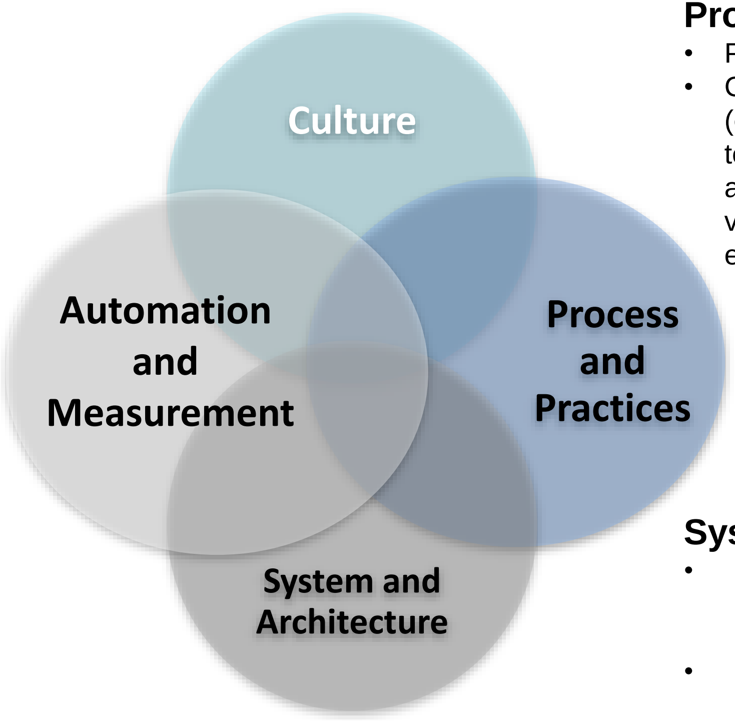
Multiple Dimensions of DevOps

Culture

- Developer and Ops collaborate (Ops includes security)
- *Developers* and Operations support releases beyond deployment
- Dev and Ops have access to stakeholders who understand business and mission goals

Process and Practices

- Pipeline streamlining
- Continuous-delivery practices (e.g., continuous integration; test automation; script-driven, automated deployment; virtualized, self-service environments)



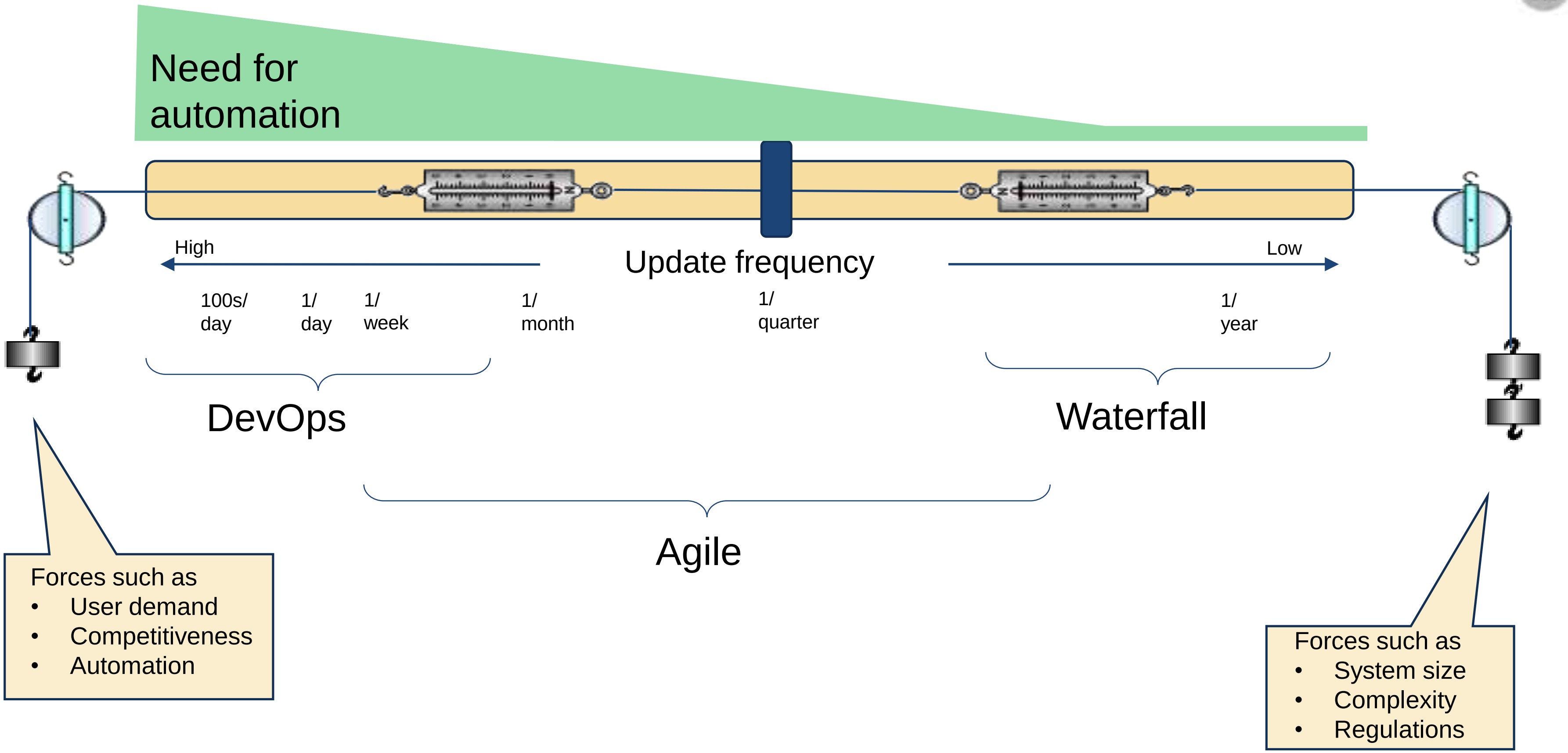
Automation/ Measurement

- Automate repetitive and error-prone tasks (e.g., build, testing, and deployment maintain consistent environments)
- Static analysis automation (architecture health)
- Performance dashboards

System and Architecture

- Architected to support test automation and continuous-integration goals
- Applications that support changes without release (e.g., late binding)
- Scalable, secure, reliable, etc.

The Agile, DevOps, Waterfall Continuum



Need for automation

High

Update frequency

Low

100s/ day

1/ day

1/ week

1/ month

1/ quarter

1/ year

DevOps

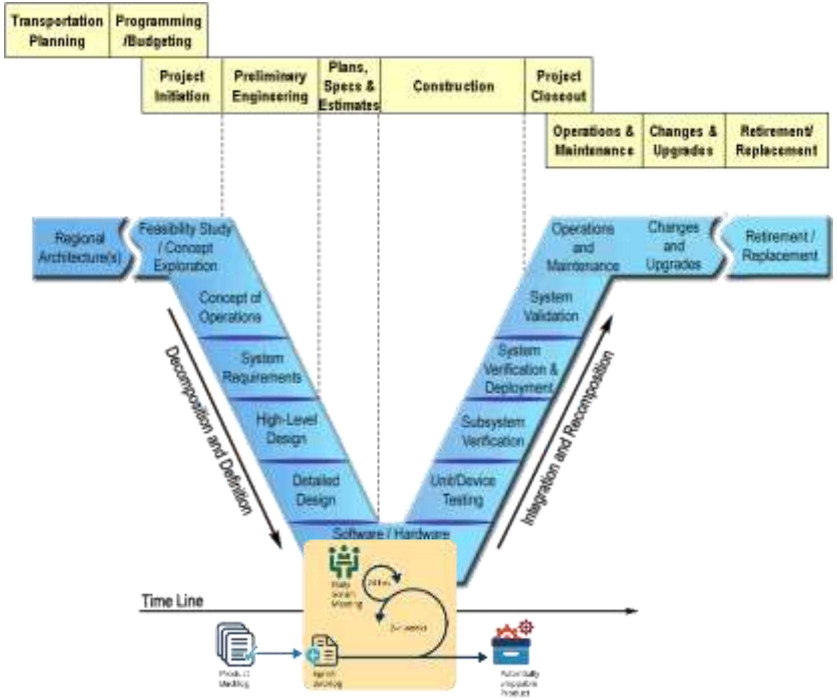
Waterfall

Agile

Forces such as
- User demand
- Competitiveness
- Automation

Forces such as
- System size
- Complexity
- Regulations

The Classic Engineering “V Model”



Source: Palmquist, Steve, et al. Parallel Worlds:

This isn’t enough

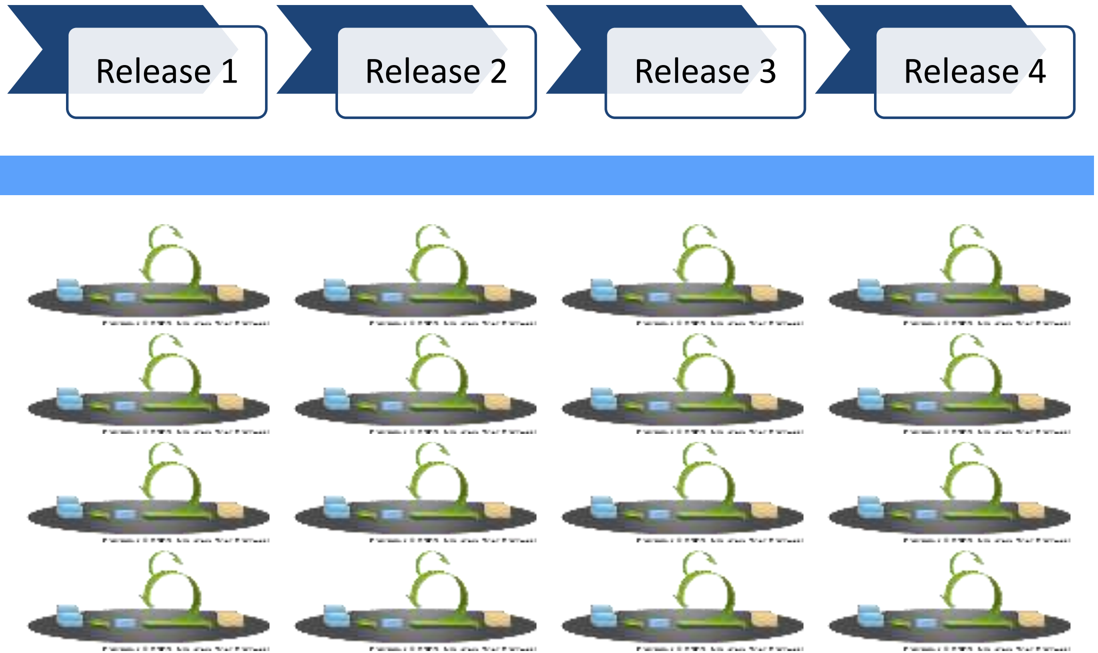
Optimizing one part of the process:

- Doesn’t optimize the whole process
- Simply exposes roadblocks by other parts of the process

“Agile at the bottom of the V”
loses benefits of agility:

- Too many decisions are made too early
- No learning opportunities

Program Level vs. Team Level Measures



Geared to External Stakeholders

Release 1

Release 2

Release 3

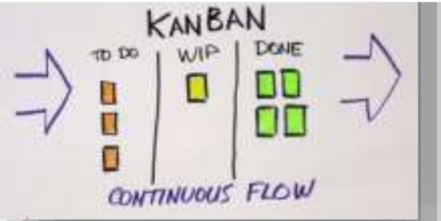
Release 4

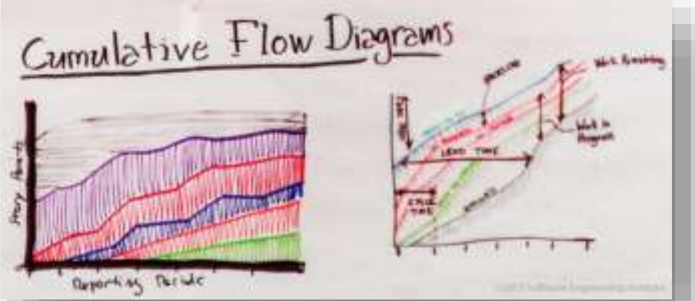
Intended to Serve Needs of the Team Typically Not Shared Out-side the Team

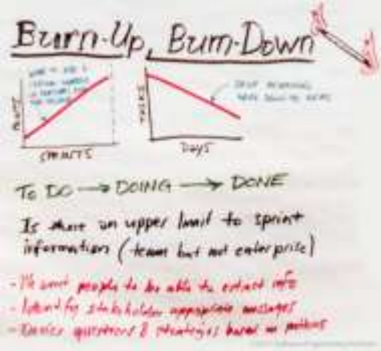
Typical Team Measures for Agile Development

Metrics used by and for the development team

- Kanban Board for Task Tracking
- Sprint Burn-Down Charts
- Release Burn-Up Charts
- Velocity Tracking
- Cumulative Flow Diagrams







Is there an upper limit to sprint information (team but not enterprise)

- We want people to be able to extract info
- Tailored for stakeholder appropriate messages
- Devise questions & strategies based on metrics

Program Level Measures

Because teams focus on delivering working code:

- The program can measure finished product (size, complexity, quality…)
  - Rather than estimates of the finished product being carried (and revised) across the program timeline, we can know *actual* values for incrementally completed work

- The program can focus on ‘concept-to-capability’ cycle
  - Hidden tradeoffs can compromise design time, or squeeze testing schedules in a waterfall lifecycle – because they are not necessarily visible until later.
  - Cycle time measures in agile lifecycles can show the entire value stream within each incremental delivery.

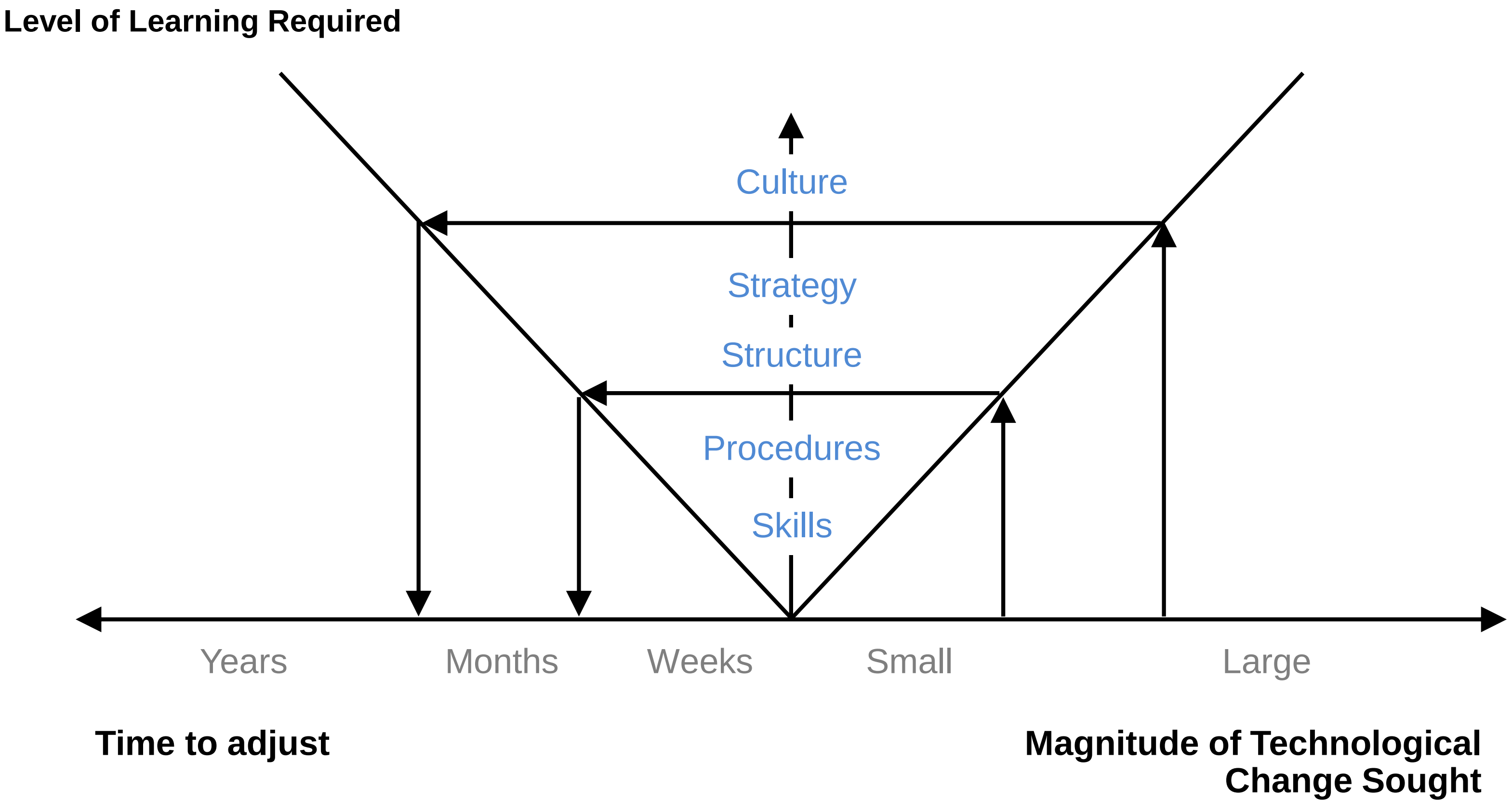
- Overall capacity can be understood earlier
  - Rather than measuring the productivity of individual disciplines, overall program capacity to achieve the desired schedule can be estimated

Categories of oversight metrics: Ask new questions

Category	Description
Flow	Flow measures come out of the lean engineering and management environment. They focus on understanding the "idea to realization" cycle time. Flow measures for senior oversight focus on the development organization's ability to consistently meet timelines for deployment of IT functions according to a roadmap. These are cycles measured in weeks and months, rather than quarterly or annual cycles seen traditionally.
Engagement	Engagement measures help oversight organizations understand the level of collaboration that has been achieved. Timely involvement of stakeholders from the workflow supported by the IT system results in a deeper understanding of intended usage. Evolution of the workflow to better utilize technology results from engagement with the correct decision makers.
Quality	Quality measures at senior oversight levels have less to do with software defect rates than they do with the quality of the services supported by the IT systems. For example, improvements in wait times for key services, or percentage of "made it through in one pass" attempts to use a service are potential quality measures. These measures, in turn, drive the priorities for quality measures among software teams.
Risk	Risk measures for senior oversight can focus on the development organizations' performance in managing threats to their success, more than those threats themselves. When using Agile methods, confidently asserting the expected success of a program is no longer based on the comprehensive- ness of up-front specification documents. Therefore, an oversight approach for Agile cannot rely on review and approval of such projective documents as the primary mode of risk identification. The short and steady cadence of Agile promotes rapid learning.

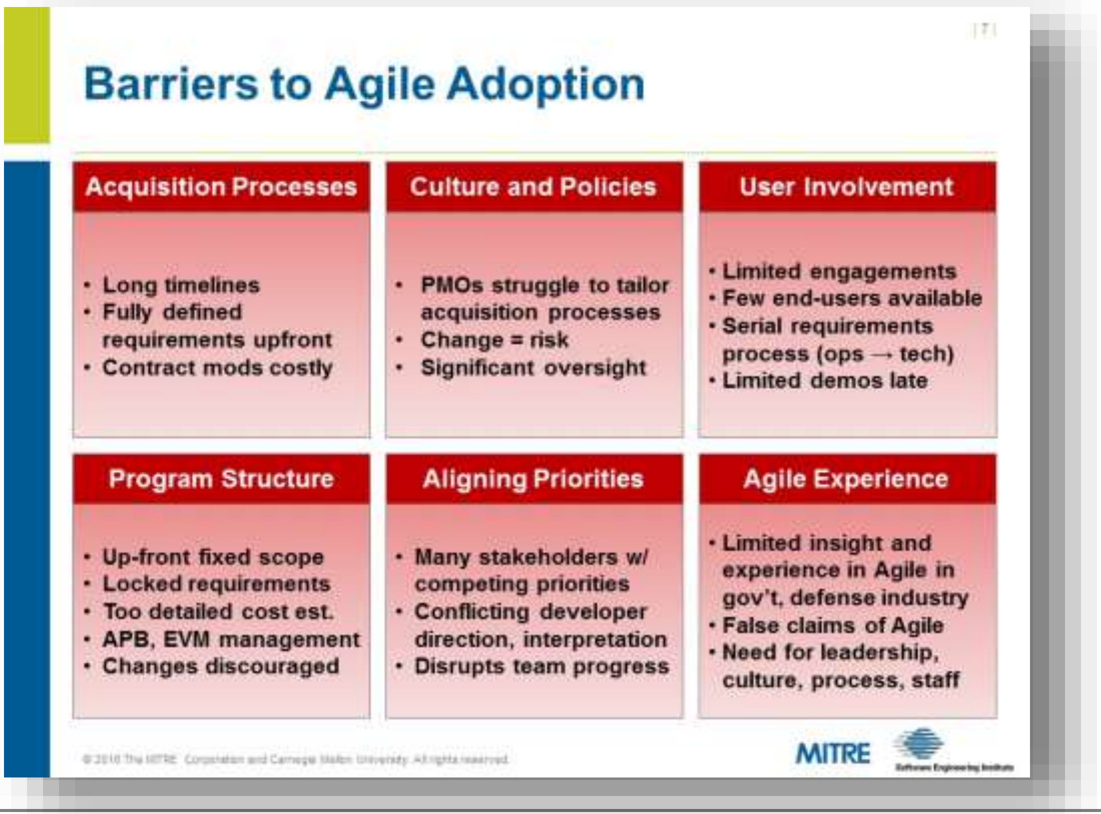
Source: SEI Congressional testimony July 14, 2016 to House Ways and Means Committee.

If this is so great, why isn’t everyone already doing it?



SEI Observations on Agile Adoption Barriers

Which of these do your programs face?



| 7 |

Barriers to Agile Adoption

Acquisition Processes

- Long timelines
- Fully defined requirements upfront
- Contract mods costly

Culture and Policies

- PMOs struggle to tailor acquisition processes
- Change = risk
- Significant oversight

User Involvement

- Limited engagements
- Few end-users available
- Serial requirements process (ops → tech)
- Limited demos late

Program Structure

- Up-front fixed scope
- Locked requirements
- Too detailed cost est.
- APB, EVM management
- Changes discouraged

Aligning Priorities

- Many stakeholders w/ competing priorities
- Conflicting developer direction, interpretation
- Disrupts team progress

Agile Experience

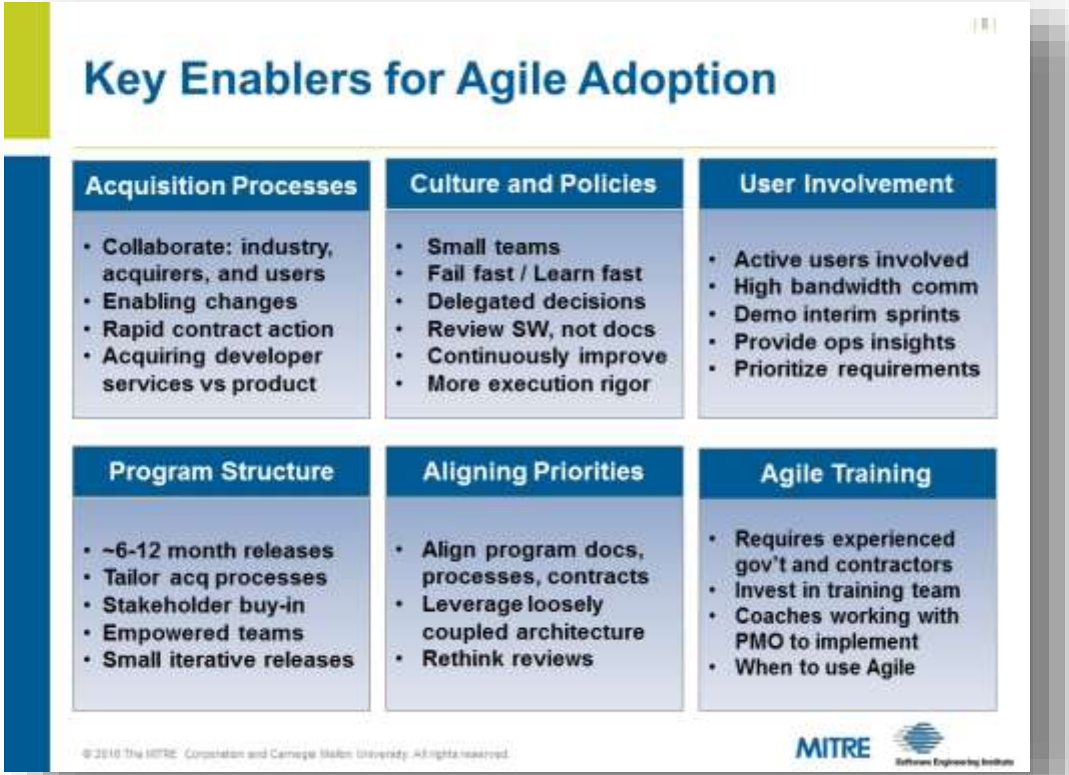
- Limited insight and experience in Agile in gov't, defense industry
- False claims of Agile
- Need for leadership, culture, process, staff

MITRE

Software Engineering Institute

SEI Observations on Key Enablers to Agile Adoption

Which of these do your programs exhibit?



Key Enablers for Agile Adoption

Acquisition Processes

- Collaborate: industry, acquirers, and users
- Enabling changes
- Rapid contract action
- Acquiring developer services vs product

Culture and Policies

- Small teams
- Fail fast / Learn fast
- Delegated decisions
- Review SW, not docs
- Continuously improve
- More execution rigor

User Involvement

- Active users involved
- High bandwidth comm
- Demo interim sprints
- Provide ops insights
- Prioritize requirements

Program Structure

- ~6-12 month releases
- Tailor acq processes
- Stakeholder buy-in
- Empowered teams
- Small iterative releases

Aligning Priorities

- Align program docs, processes, contracts
- Leverage loosely coupled architecture
- Rethink reviews

Agile Training

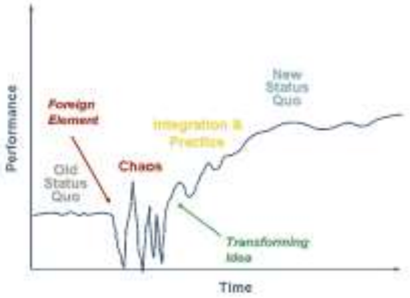
- Requires experienced gov't and contractors
- Invest in training team
- Coaches working with PMO to implement
- When to use Agile

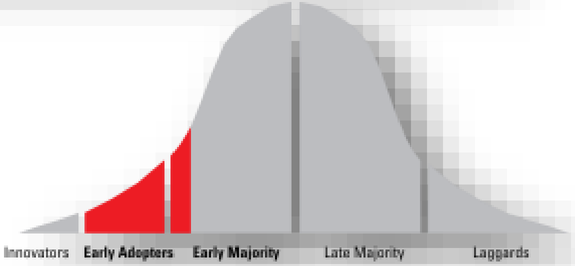
MITRE

Software Engineering Institute

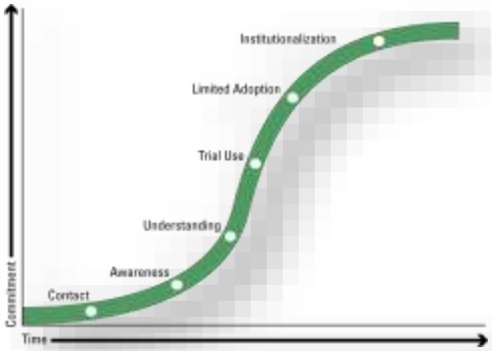
“Traditional” Adoption Tools and Methods Work Well with Agile Adoption

Understand the Change Cycle and Your Adoption Population



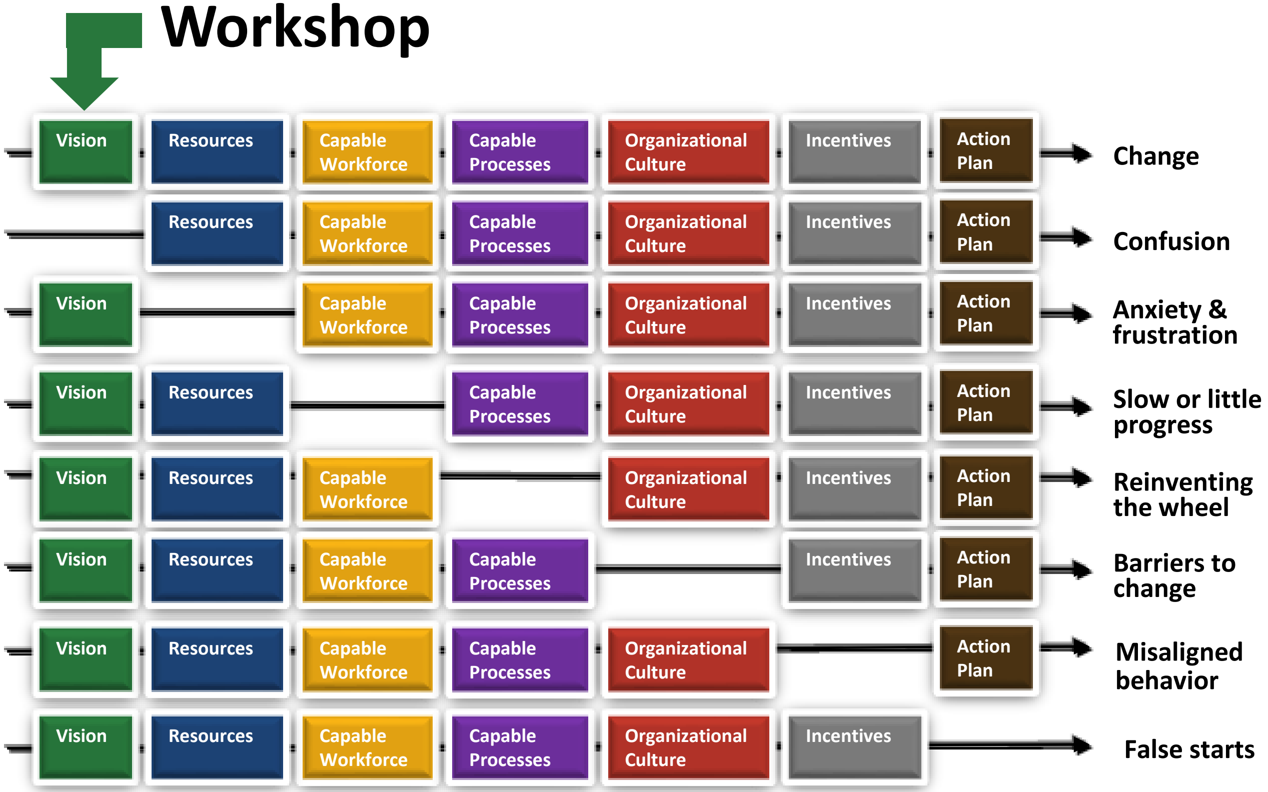


Prepare for Both Communication and Implementation Support Mechanisms that are Needed



*Adapted from Daryl R. Conner and Robert W. Patterson, “Building Commitment to Organizational Change,” *Training and Development Journal* (April 1983): 18-30.

Where Leadership, Vision, and Goals Fit into Organizational Improvement



Workshop

Vision	Resources	Capable Workforce	Capable Processes	Organizational Culture	Incentives	Action Plan	Change
	Resources	Capable Workforce	Capable Processes	Organizational Culture	Incentives	Action Plan	Confusion
Vision		Capable Workforce	Capable Processes	Organizational Culture	Incentives	Action Plan	Anxiety & frustration
Vision	Resources		Capable Processes	Organizational Culture	Incentives	Action Plan	Slow or little progress
Vision	Resources	Capable Workforce		Organizational Culture	Incentives	Action Plan	Reinventing the wheel
Vision	Resources	Capable Workforce	Capable Processes		Incentives	Action Plan	Barriers to change
Vision	Resources	Capable Workforce	Capable Processes	Organizational Culture		Action Plan	Misaligned behavior
Vision	Resources	Capable Workforce	Capable Processes	Organizational Culture	Incentives		False starts

Adapted by Buttles (2010) from: Delorise Ambrose, 1987

Attributes of Agile Success in Government Organizations

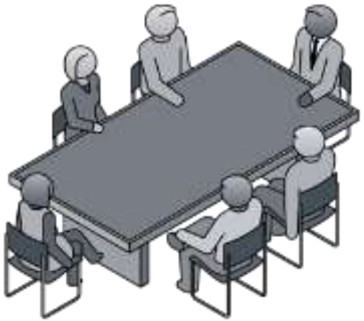
Top Cover

Permission to “fail fast” (learn fast)

Training in Agile

Use of Agile coach

Enough up-front system and software architecture

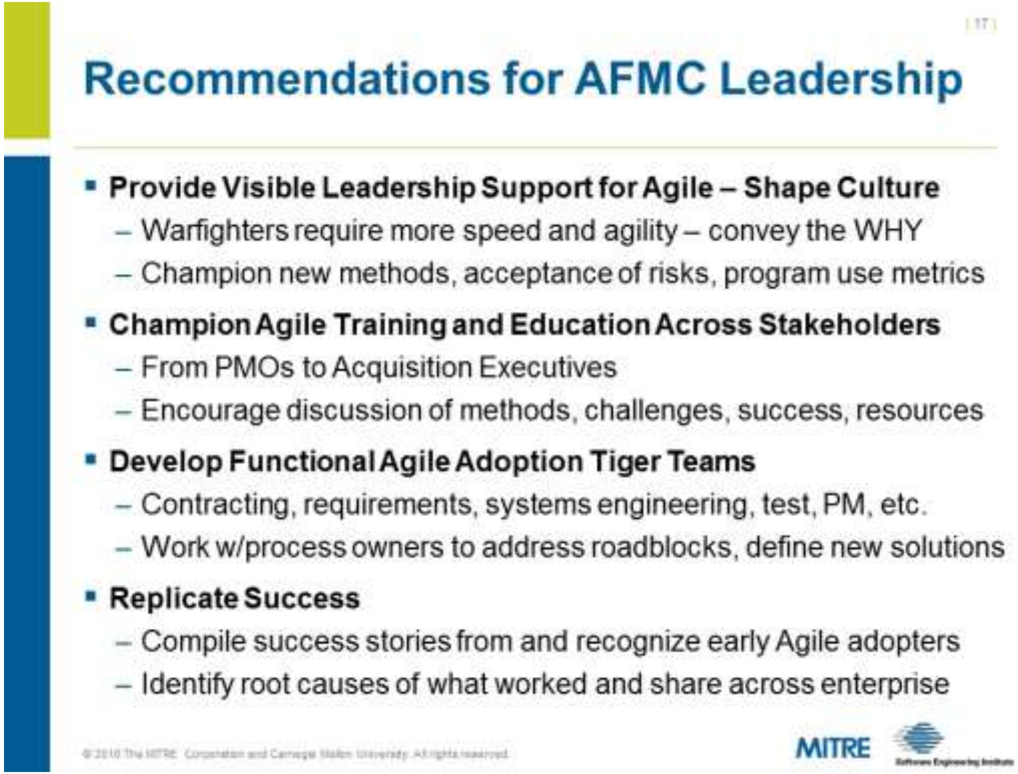


Willing to work collaboratively across government/ contractor boundary

Willing and open to adopt new modes of operation

Dedicated staff

What do leaders have to do to change the environment?



[17]

Recommendations for AFMC Leadership

- **Provide Visible Leadership Support for Agile – Shape Culture**
  - Warfighters require more speed and agility – convey the WHY
  - Champion new methods, acceptance of risks, program use metrics
- **Champion Agile Training and Education Across Stakeholders**
  - From PMOs to Acquisition Executives
  - Encourage discussion of methods, challenges, success, resources
- **Develop Functional Agile Adoption Tiger Teams**
  - Contracting, requirements, systems engineering, test, PM, etc.
  - Work w/process owners to address roadblocks, define new solutions
- **Replicate Success**
  - Compile success stories from and recognize early Agile adopters
  - Identify root causes of what worked and share across enterprise

About Agile: Summary

Agile is an iterative approach to software delivery that builds and delivers software incrementally from the start of the project, instead trying to deliver it all at once near the end.

- Early opportunity for course correction, especially when the environment changes after a program has begun
- Early risk reduction, especially in user-facing areas of the system
- Shorter “idea to realization” cycle resulting in fast user feedback for future increments of functionality

But it’s about more than software engineering to do it right: Needs business/acquisition process support

Oversight: Responsibility for oversight and due diligence doesn’t change; approach to oversight in an Agile setting does. Some examples:

Flow: Predictable delivery volume, deployment speed	**Engagement:** stakeholder involvement
Quality: Defect backlog	**Risk:** Deferred complexity

Contracting: Benefits can’t be realized without contracting approaches that allow for fast learning & pivoting. Some examples:

Supply contracts	Blanket contracts w/pre-qualified contractors/IDIQ pools
Service contracts	Commercial item contracts for development services (FAR 13.5)

The FAR/DFARS encourage bold innovation – the culture has a long way to go

About Agile: Summary (contd.)

(adapted from SEI Testimony to House Ways and Means Social Security Subcommittee)

Agile will not solve all the complex problems associated with software-dominant systems acquisition and sustainment efforts

- But it has contributed significantly to successful efforts (both in IT and weapons systems)

Benefits from using Agile methods only manifest when the developer and acquisition efforts are aligned

Government obligations in oversight must change when Agile is the focus of development

- SEI has observed negative consequences in organizations that do not address these changes.

Changing the oversight approach in Agile settings means asking different questions on a new cadence

- Leads to different measurement and reporting approaches as well.

A focused government workforce development effort is required to enable the knowledge, skills, and abilities needed for effective oversight and interaction in Agile settings.

[1] July 14, 2016, Link: http://waysandmeans.house.gov/event/hearing-modernizing-social-securitys-information-technology-infrastructure/

Contact Information

Will Hayes
Principal Engineer
Software Engineering Institute
Email: wh@sei.cmu.edu
Phone: +1 412 268-6398

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

Customer Relations

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

Web www.sei.cmu.edu/go/agile

BACKUP MATERIALS

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

A Word about Sample RFP Language

No “iconic” RFP language for encouraging Agile development practices exists

- Lots of factors go into what language would be appropriate
- DCMA is considering changes to their policies related to audit points, etc, which could point to some new language—not expected for another year
- NDIA System Engineering Agile working group developed a Special Report on this topic:

RFP Patterns and Techniques for Successful Agile Contracting

http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484056

Useful Interpretation of Agile Principles for Government Settings (1/3)

Agile Principle	Useful Interpretations in Government Settings
The highest priority is to satisfy the customer through early and continuous delivery of valuable software.	In government, the “customer” is not always the end user. The customer includes people who pay for; people who use; people who maintain; as well as others. These stakeholders often have conflicting needs that must be reconciled
Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.	Rather than saying “competitive” advantage, we usually say “operational” advantage. This principle causes culture clash with the “all requirements up front” perspective of many large, traditional approaches.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.	What it means to “deliver” an increment of software may well depend on context. With large embedded systems, we are sometimes looking at a release into a testing lab. Also, for some systems, the operational users are not able to accept all: “deliveries” on the development cadence – because there are accompanying changes in the workflow supported by the software that require updates.
Business people and developers must work together daily throughout the project.	In government settings, we interpret “business” people to be end users and operators, as well as the other types of stakeholders mentioned in Principle 1, since in many government settings, the business people are interpreted as the contracts and finance group.

Source: SEI Congressional testimony July 14, 2016 to House Ways and Means Committee.

Useful Interpretation of Agile Principles for Government Settings (2/3)

Agile Principle	Useful Interpretations in Government Settings
Build projects around motivated individuals. Give them environment and support they need, and trust them to get the job done.	A frequent challenge in government is to provide a suitable technical and management environment to foster the trust that is inherent in Agile settings. Allowing teams to stay intact and focused on a single work stream is another challenge.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	In today's world, even in commercial settings, this is often interpreted as "high bandwidth" rather than only face-to-face. Telepresence via video or screen-sharing allows more distributed work groups than in the past.
Working software is the primary measure of progress.	Our typical government system development approaches use *surrogates* for software – documents that project the needed requirements and design – *rather than the software itself*, as measures of progress. Going to small batches in short increments allows this principle to be enacted, even in government setting, although delivery may well to be a test environment or some internal group other than users themselves.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	This principle is a caution against seeing agility just as "do it faster." Note that this principle includes stakeholders outside of the development team as part of the pacing.

Source: SEI Congressional testimony July 14, 2016 to House Ways and Means Committee.

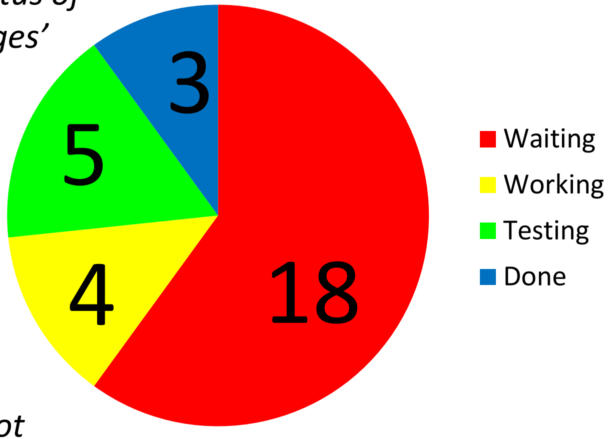
Useful Interpretation of Agile Principles for Government Settings (3/3)

Agile Principle	Useful Interpretations in Government Settings
Continuous attention to technical excellence and good design enhances agility	This is a principle that often is cited as already being compatible with traditional government development.
Simplicity– the art of maximizing the amount of work not done– is essential.	One issue with this principle in government setting is that our contracts are often written to penalize the development organization if they don't produce a product that reflects 100% of the requirements. This principle recognizes that not all requirements we think are needed at the onset of a project will necessarily turn out to be things that should be included in the product.
The best architectures, requirements, and designs emerge from self-organizing teams.	Note that the principle does not suggest that the development team is necessarily the correct team for requirements and architecture. It is however, encouraging teams focused in these areas to be allows some autonomy to organize their work. Another complication in many government settings is that we are often re-architecting and re-designing existing systems.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	This principle is an attempt to ensure that “lessons learned” are actually learned and applied rather than just being “lessons written”

Source: SEI Congressional testimony July 14, 2016 to House Ways and Means Committee.

Constructing a Cumulative Flow Diagram$_1$

Here we have a Pie Chart showing the status of 30 'work packages'

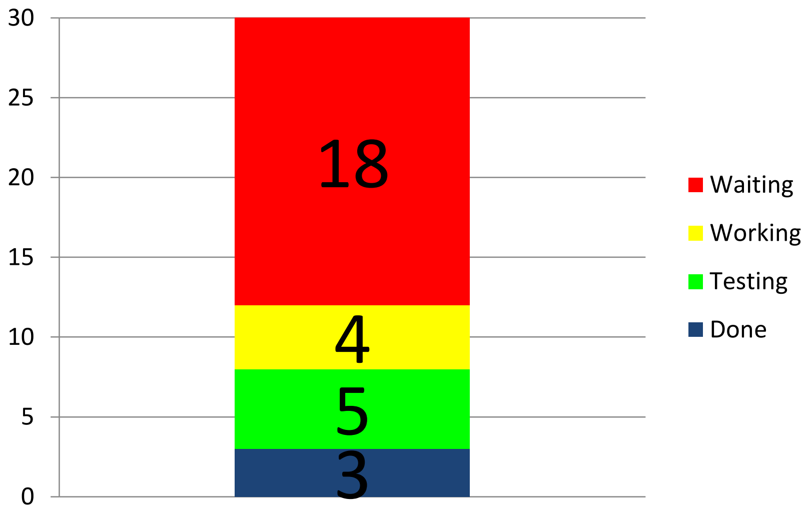


This is a snapshot for a single point in time.

Constructing a Cumulative Flow Diagram$_2$

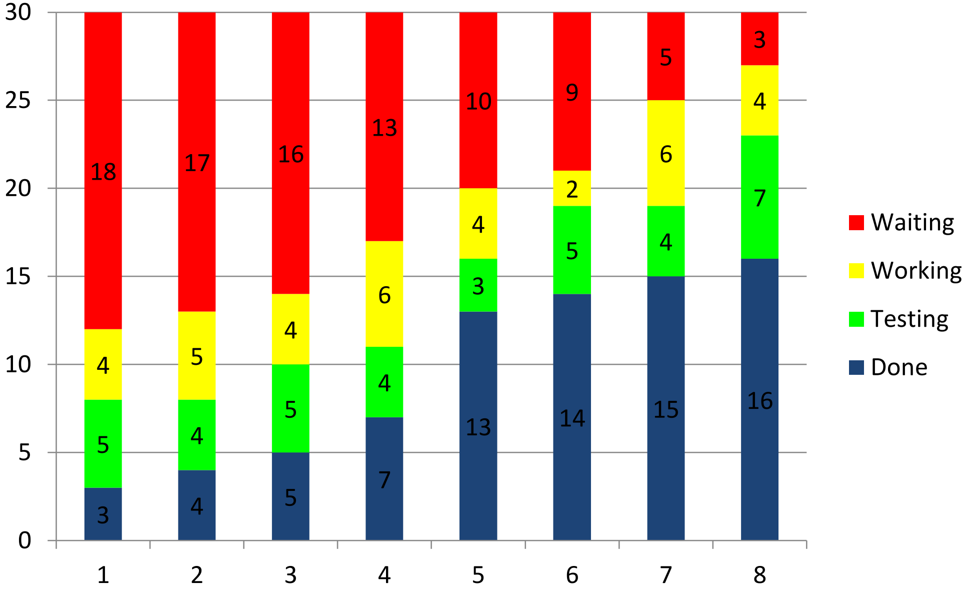
Same data, but presented in a stacked column chart

For a single point in time.



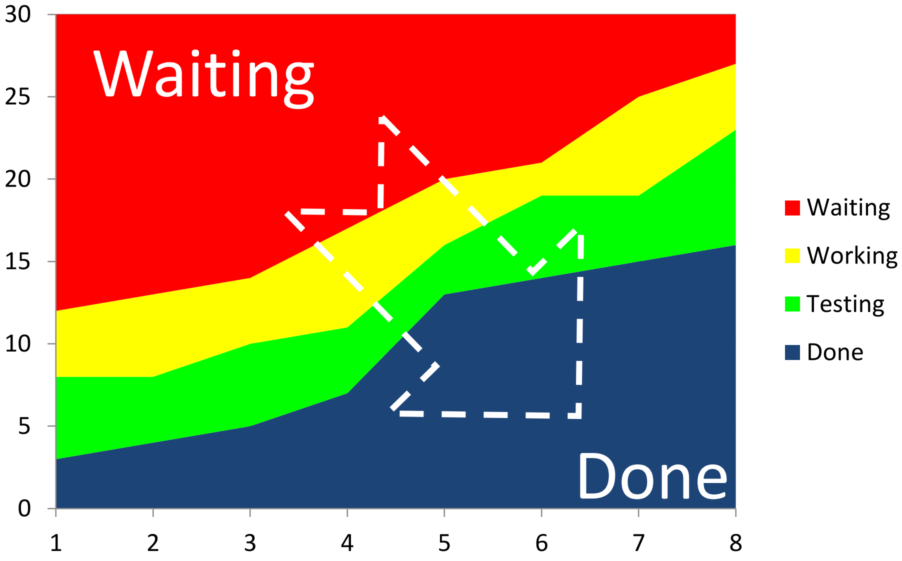
Constructing a Cumulative Flow Diagram$_3$

… adding the next 7 times



Constructing a Cumulative Flow Diagram$_4$

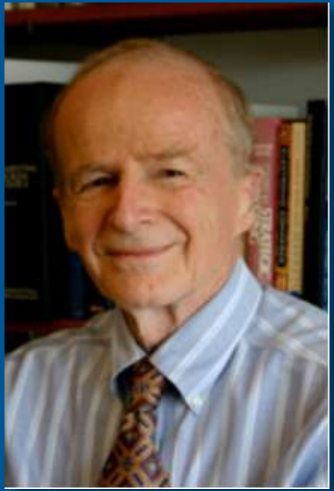
… now we are looking at the flow from “Waiting” to “Done”…
This view starts to show patterns a little easier…



Little’s Law

$$L = \lambda W$$

…the long-term average number L of customers in a stationary system is equal to the long-term average effective arrival rate λ multiplied by the average time W that a customer spends in the system…

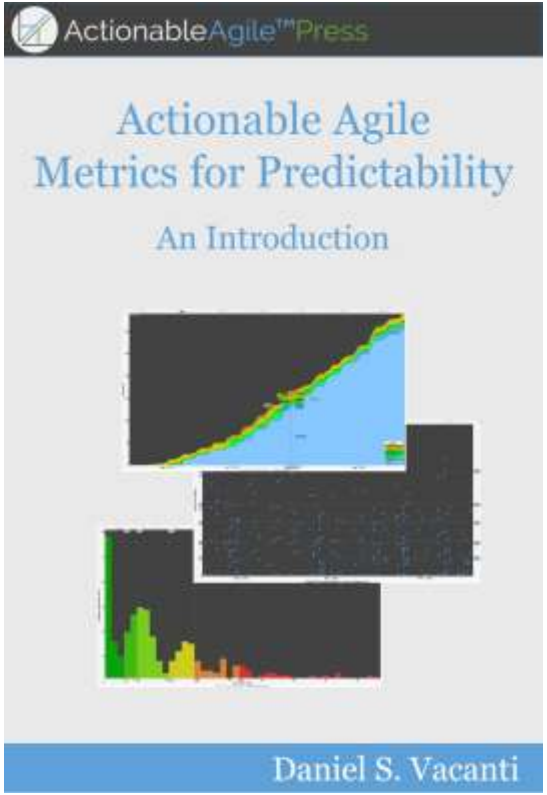


http://mitsloan.mit.edu/faculty-and-research/faculty-directory/detail/?id=41432

Little's Law in Agile Metrics

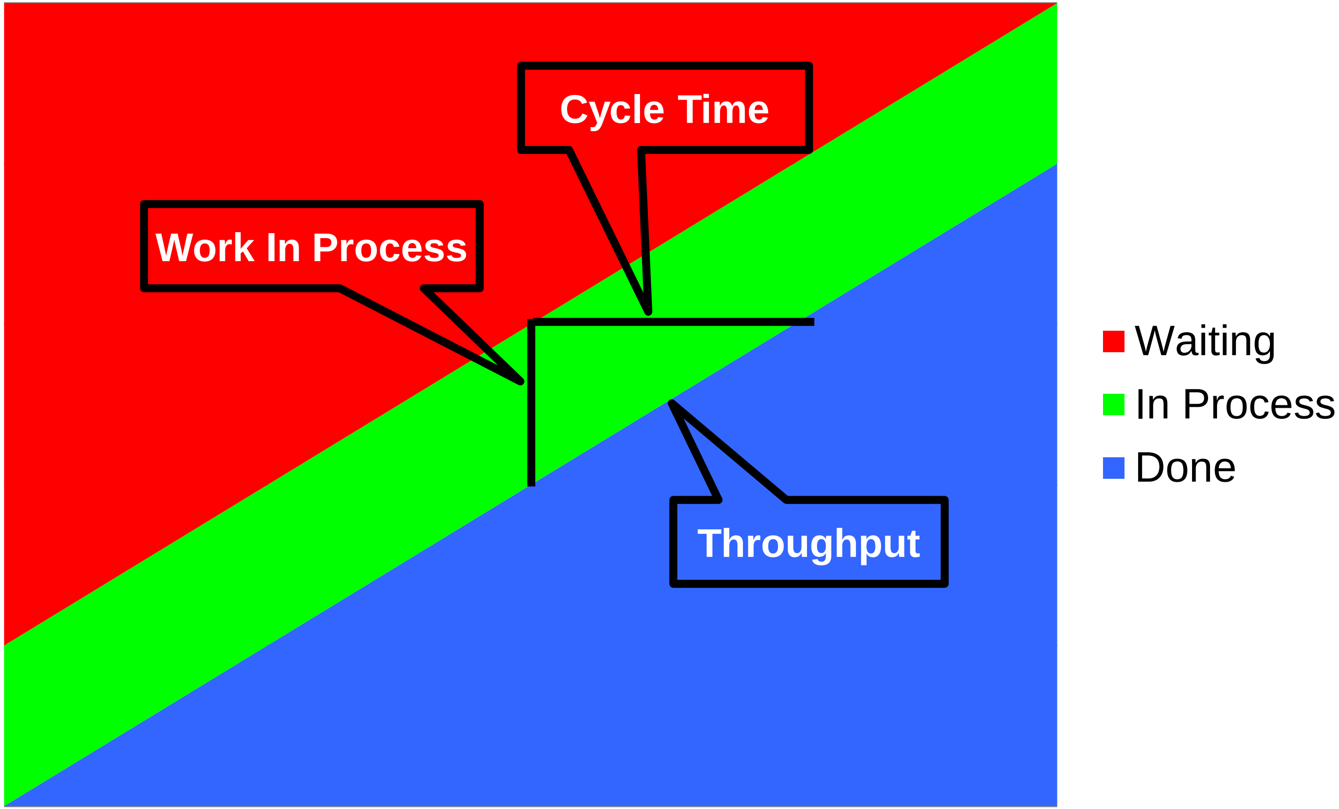
Three Metrics Emphasized*:

1. **Work In Progress** (the number of items that we are working on at any given time),
2. **Cycle Time** (how long it takes each of those items to get through our process), and
3. **Throughput** (how many of those items complete per unit of time).

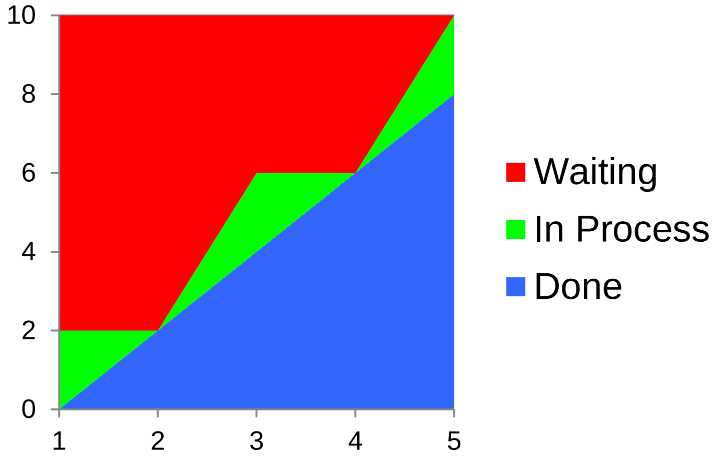


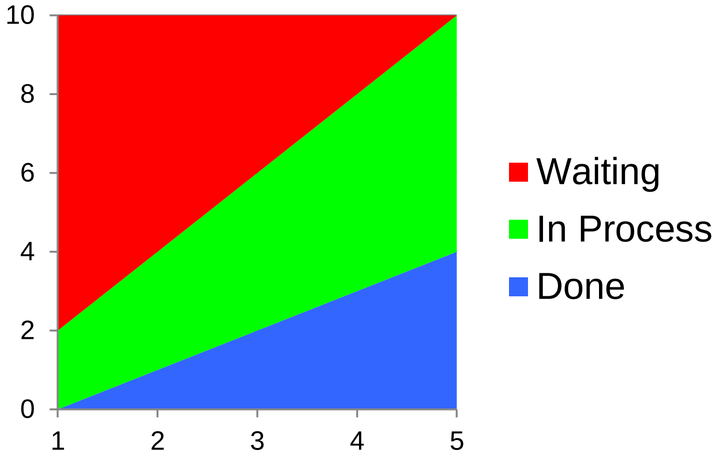
** Excerpted from page 13 of the book depicted on the right.*

Utility of Little’s Law

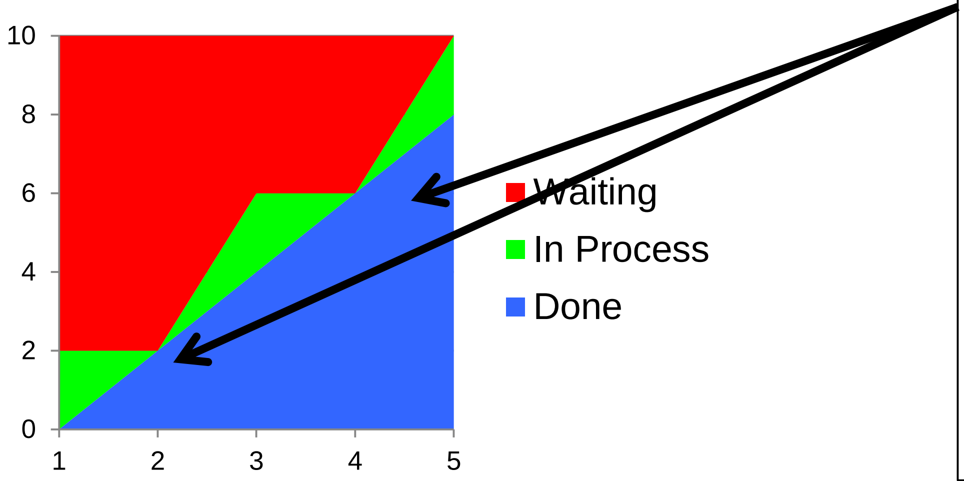


Exercise: What is Going on Here?





Exercise: What *MIGHT BE* Happening$_1$



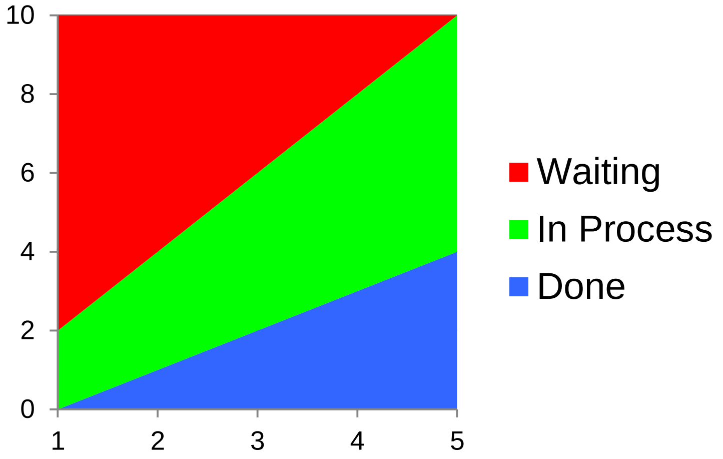
At time 2, and then again at time 4, the number of items “In Process” goes to zero.

- Have we lost the resource(s) performing the work due to rework demands from elsewhere?
- Is this intentional scheduling of work to occur only during time periods 1, 3, and 5?

Exercise: What *MIGHT BE* Happening$_2$

The number of items that are “In Process” is growing over time.

- The rate at which things enter “In Process” is greater than the rate at which things leave “In Process.”
- Are people moving onto new items without completing their work?
- Are new resources being added, who start new work at each time period?
- Are things moving into the “Done” state quickly enough?


🟥 Waiting
🟩 In Process
🟦 Done