SAFe 4.0 for Teams

Day 1

1. Introducing the Scaled Agile Framework
2. Building an Agile Team
3. Planning the Iteration

Day 2

4. Executing the Iteration
5. Executing the PI

Logistics

- Class times
- Breaks
- Lunch
- Restrooms
- Other

At the end of this course you should be able to:

- Apply SAFe to scale Lean and Agile development in your enterprise
- Know your team and its role on the Agile Release Train
- Know all other teams on the train, their roles and the dependencies between the teams
- Plan iterations
- Execute iterations and demo value
- Plan program increments
- Integrate and work with other teams on the train

Lesson 1

Introducing the Scaled Agile Framework

Day 1

1. Introducing the Scaled Agile Framework
2. Building an Agile Team
3. Planning the Iteration

Day 2

4. Executing the Iteration
5. Executing the PI

Learning objectives

1.1 Have a basic understanding of SAFe

1.2 Embrace SAFe values, mindset, and principles

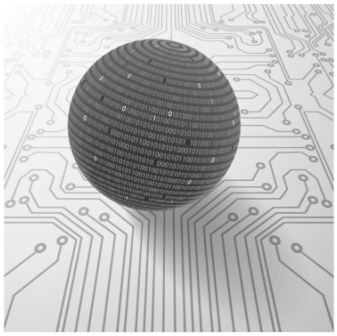
1.3 Understand Scrum, Kanban, and XP

1.1 Have a basic understanding of SAFe

Keeping pace

Our methods must keep pace with an increasingly complex world.

- We've had Moore's Law for hardware, and now software is eating the world
- Our development practices haven't kept pace; Agile shows the greatest promise but was developed for small teams
- We need a new approach, one that harnesses the power of Agile and Lean and applies to the needs of the largest development enterprises



We thought we'd be developing like this:

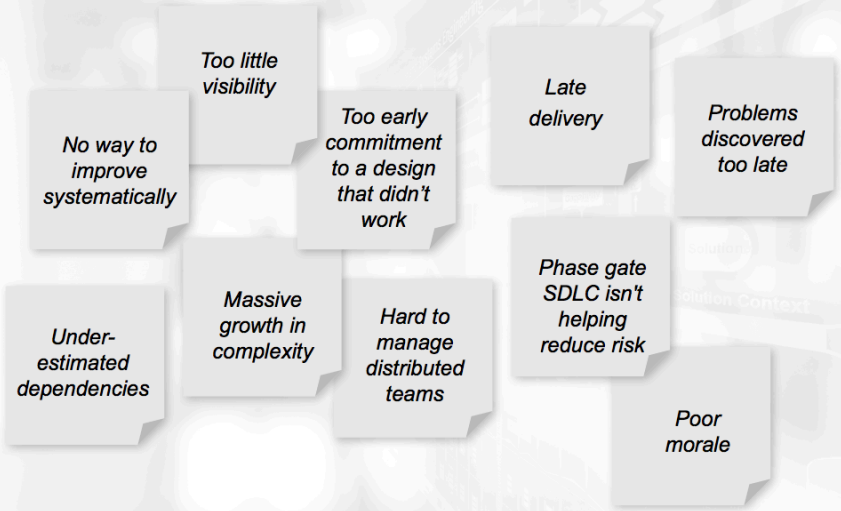
1.9

But sometimes it feels like this:

Library of Congress

1.10

And our retrospectives read like this:



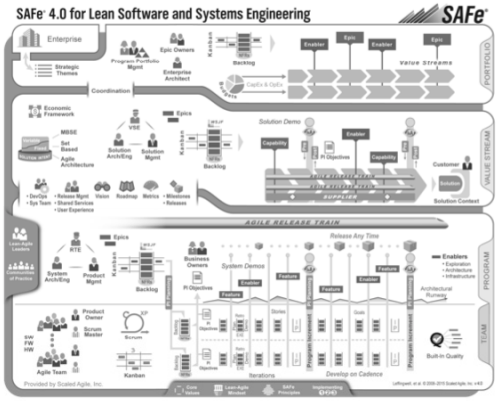
1.11

SAFe is a freely revealed
online knowledge base of
proven, integrated success patterns
for implementing Lean-Agile development
at Enterprise scale

1.12

The Scaled Agile Framework® (SAFe®)

Synchronizes alignment, collaboration, and delivery for large numbers of teams



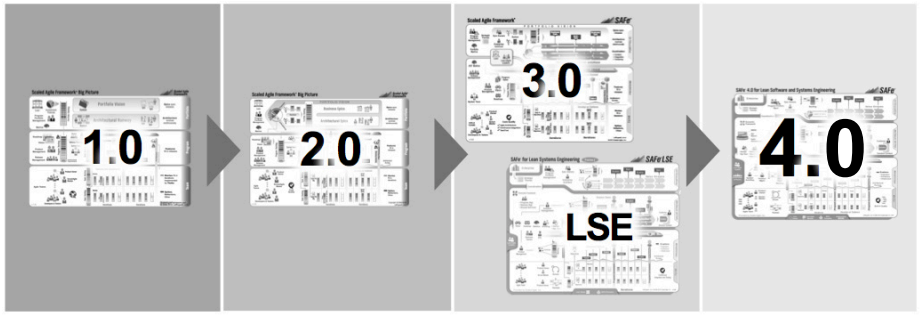
Core Values

1. Built-In Quality
2. Program execution
3. Alignment
4. Transparency

Roots past, present, and future

Field experience at Enterprise scale

2011 Now…



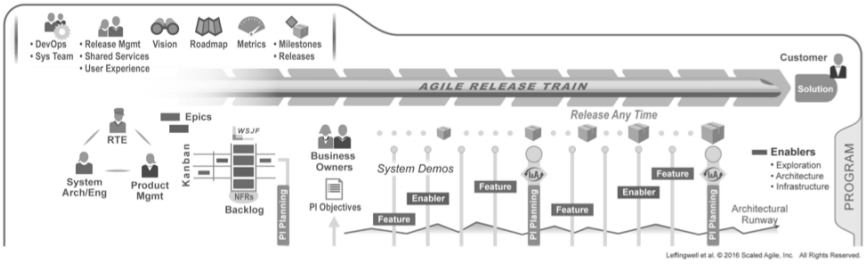
Agile development | Lean product development | Systems thinking

Nothing beats an Agile Team

- Empowered, self-organizing, self-managing, cross-functional team
- Delivers a valuable, tested, working system every two weeks
- Uses a team framework that combines the best of Scrum project management, XP-inspired technical practices, and Kanban for flow
- Delivers value via user stories

. . . Except a team of Agile Teams

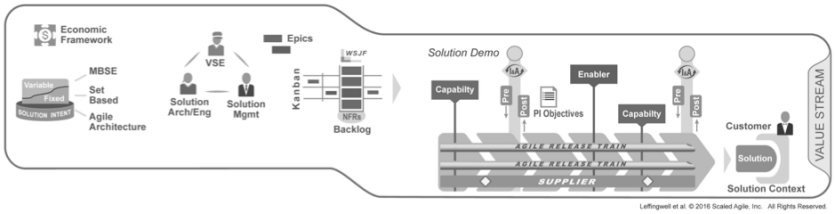
- Self-organizing, self-managing team-of-Agile-Teams
- Delivers working, tested, full-system increments every two weeks
- Operates with Vision, architecture, and UX guidance



- Common Iteration lengths and estimating
- Face-to-face planning for collaboration, alignment, and adaptation
- Value delivery via Features and benefits

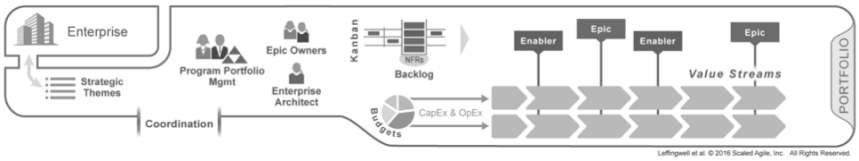
Coordinating large Value Streams

- Coordinates development of large Solutions
- Synchronizes multiple ART Value Streams
- Manages Solution Intent



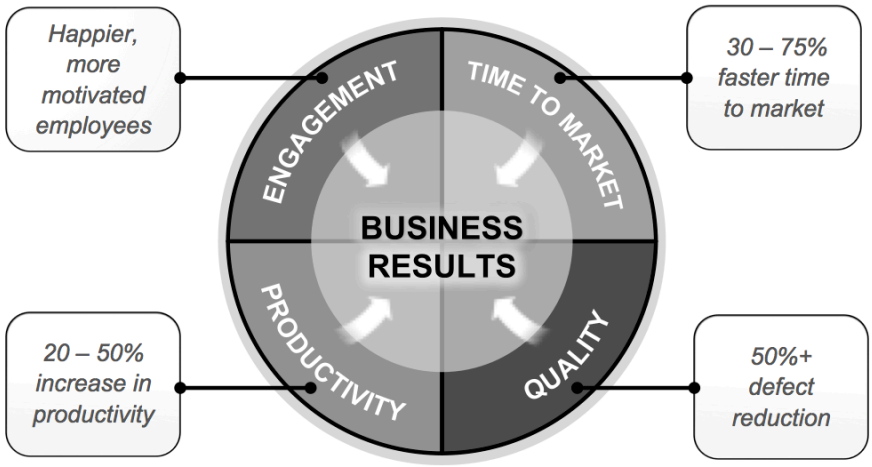
- Integrates Suppliers as partners
- Delivers value via Capabilities

In an Agile Portfolio



- Organized around the flow of value
- Lean-Agile budgeting empowers decision-makers
- Kanban system provides portfolio visibility and WIP limits
- Enterprise architecture guides larger technology decisions
- Objective Metrics support governance and improvement
- Delivers value via Epics

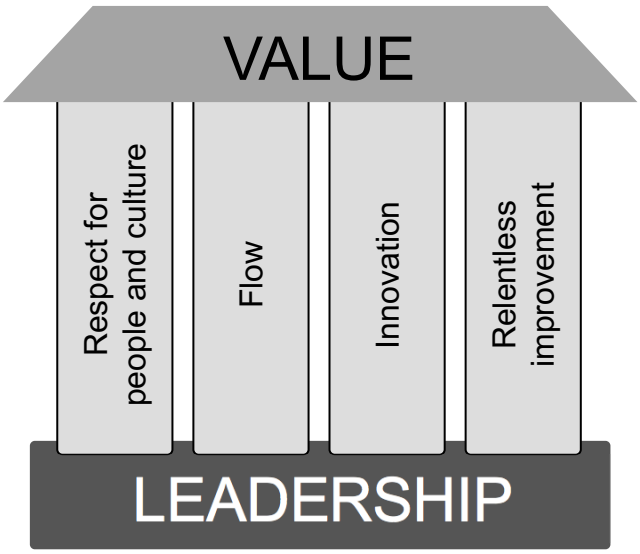
That gets business results



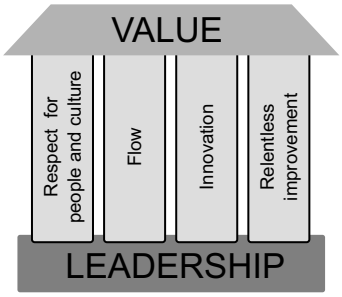
ScaledAgileFramework.com/**case-studies**

1.2 Embrace SAFe values, mindset, and principles

SAFe House of Lean



Purpose

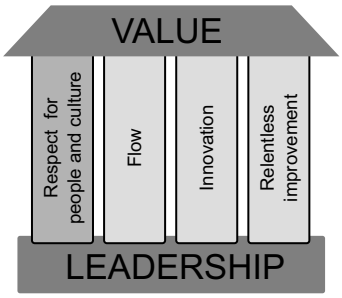


There is only one boss. The customer. And he can fire everybody in the company.
—Sam Walton

Achieve the sustainably shortest lead time with:

- Best quality and value to people and society
- High morale, safety, customer delight

Respect for people and culture

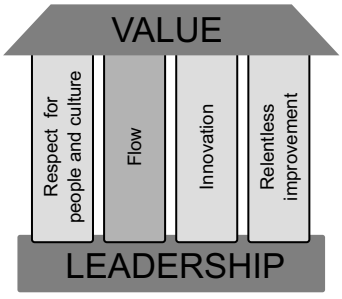


Culture eats strategy for breakfast.

—Peter Drucker

- People do all the work
- Your customer is whoever consumes your work
  - Don't overload them
  - Don't make them wait
  - Don't force them to do wasteful work
  - Don't impose wishful thinking
- Build long-term partnerships based on trust
- Cultural change comes last, not first
- To change the culture, you have to change the organization

Flow

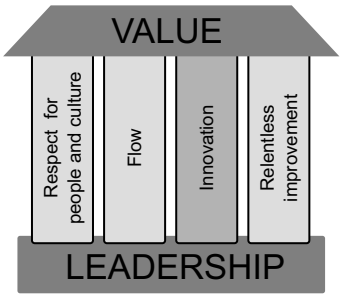


Operating a product development process near full utilization is an economic disaster.

—Don Reinertsen

- Optimize continuous and sustainable throughput of value
- Avoid start-stop-start project delays
- Build quality in; flow depends on it
- Understand, exploit, and manage variability
- Integrate frequently
- Inform decision-making via fast feedback

Innovation



Innovation comes from the producer.

—W. Edwards Deming

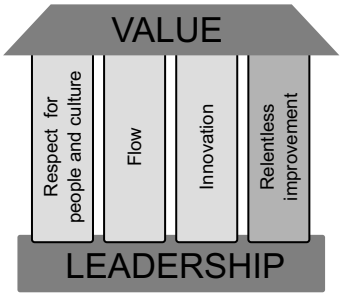
- Producers innovate; customers validate
- Get out of the office (Gemba*)

  No useful improvement was ever invented at a desk.

  — Taiichi Ohno
- Provide time and space for creativity
- Apply innovation accounting
- Pivot without mercy or guilt

** Gemba: The "real place" where the work is actually done.*

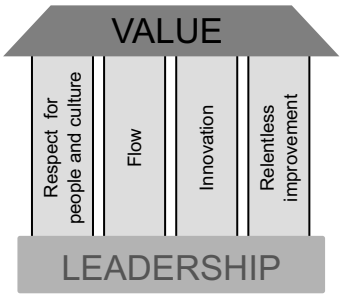
Relentless improvement



Those who adapt the fastest, win.

- A constant sense of danger
- Optimize the whole
- Consider facts carefully, then act quickly
- Apply Lean tools to identify and address root causes
- Reflect at key Milestones; identify and address shortcomings

Leadership



People are already doing their best; the problems are with the system.

Only management can change the system.

—W. Edwards Deming

- Lead the change
- Know the way; emphasize life-long learning
- Develop people
- Inspire and align with mission; minimize constraints
- Decentralize decision-making
- Unlock the intrinsic motivation of knowledge workers

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.

agilemanifesto.org

Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

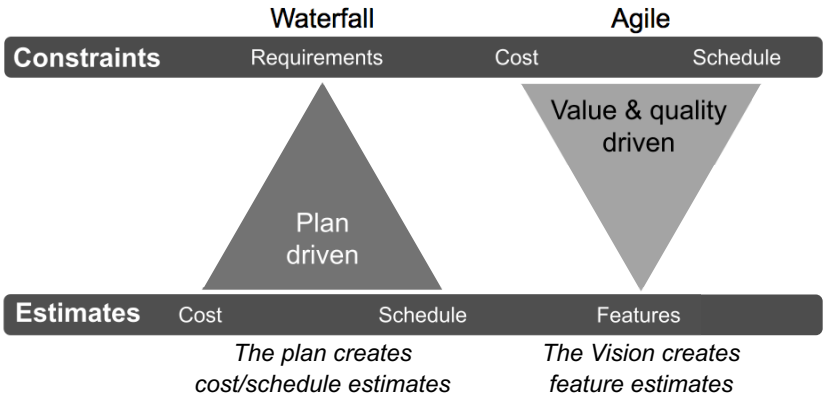
agilemanifesto.org/principles.html

Agile Manifesto

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain 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.

agilemanifesto.org/principles.html

Agile turns development upside-down



- Agile Teams show that *dates* matter and *meet* their commitments
- Business Owners understand how *priorities* matter
- Fix *quality*, not scope

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

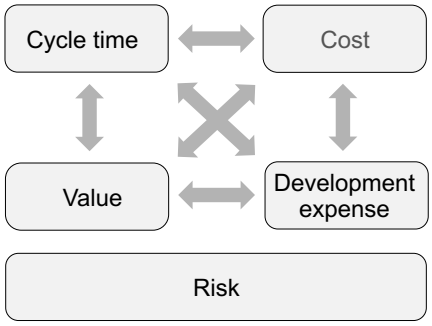
#7 - Apply cadence, synchronize with cross-domain planning

#8 - Unlock the intrinsic motivation of knowledge workers

#9 - Decentralize decision-making

#1 Take an economic view

Many decisions at the team level impact development economics.



Understanding economics requires understanding of the interaction amongst multiple variables.

—Don Reinertsen, Principles of Product Development Flow

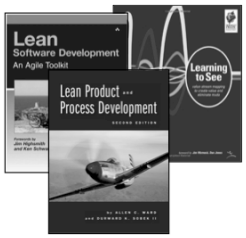
- Validate key decisions based on how it impacts the five variables
- Team makes thousands of decisions during the PI
- Effective team process leverages more economically feasible ways of product development
- Taking economic view does not always require knowing "dollarized value" but is rather a general thinking tool

#2 Apply Systems Thinking

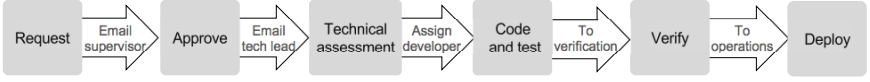
All we are doing is looking at the timeline, from when the customer gives us an order to when we collect the cash. And we are reducing the timeline by reducing the non-value-added wastes. *—Taiichi Ohno*

Understand the full Value Stream:

- Most inefficiencies and impediments in your process will surface themselves as *delays* in value delivery
- Consider all steps as part of the Value Stream, including definition, analysis, validation, and delivery
- Customers and Suppliers are part of your Value Stream
- Establish a culture of continuous improvement of the *full* Value Stream



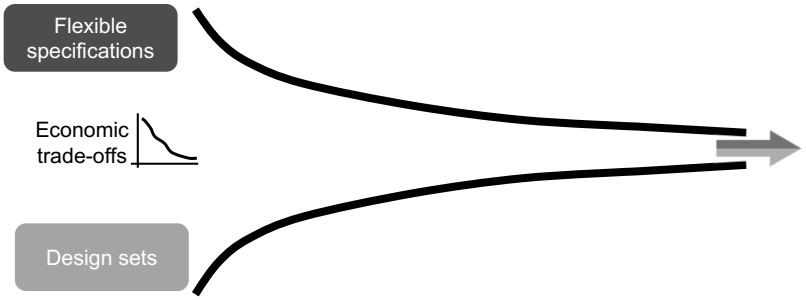
Poppendieck, Ward et al,
Rother, Shook, Womack



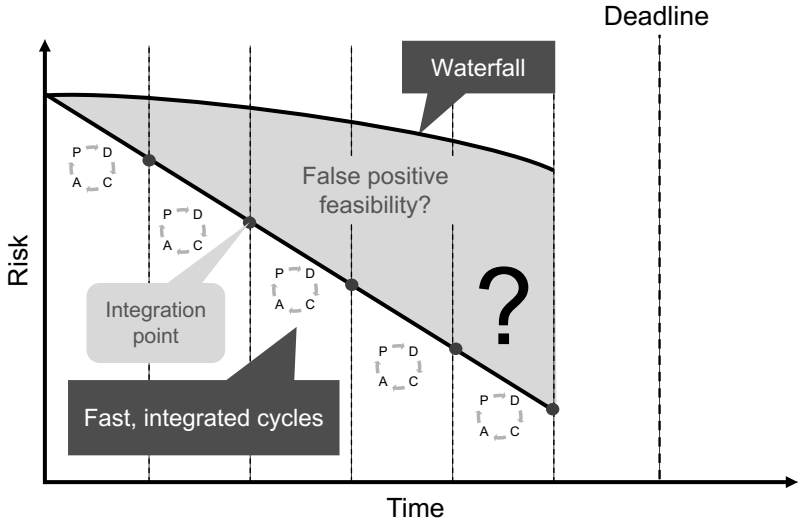
#3 Assume variability; preserve options

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

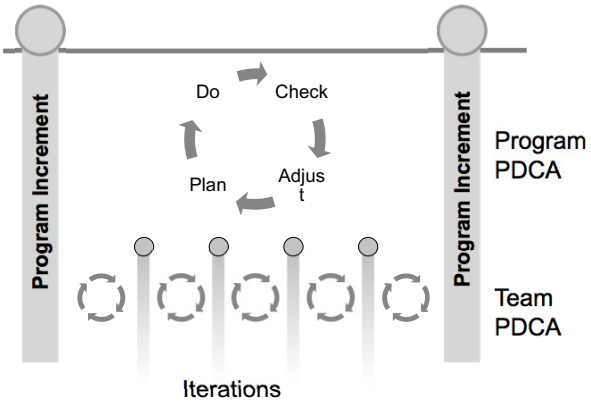
- You cannot possibly know everything at the start
- Requirements must be flexible to make economic design choices
- Designs must be flexible to support changing requirements
- Preservation of options improves economic results



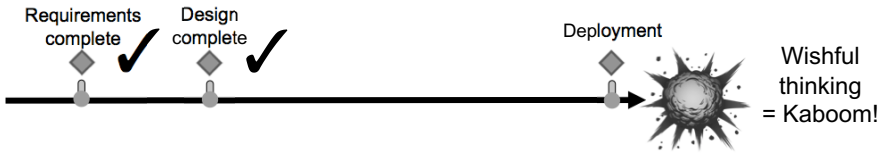
#4 Build incrementally with fast, integrated learning cycles



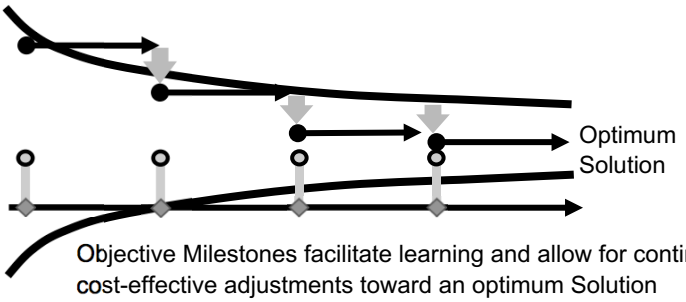
Use Iterations and Program Increments to learn fast



#5 Base milestones on objective evaluation of working systems



- Phase gates force too-early design decisions, encourage false-positive feasibility
- Assume a "point" Solution exists and can be built right the first time



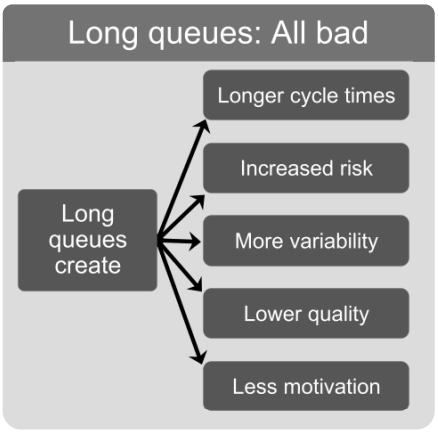
Objective Milestones facilitate learning and allow for continuous, cost-effective adjustments toward an optimum Solution

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

- Understand Little's Law
- Faster processing time decreases wait
- Control wait times by controlling queue lengths

$$W_q = \frac{L_q}{\lambda}$$

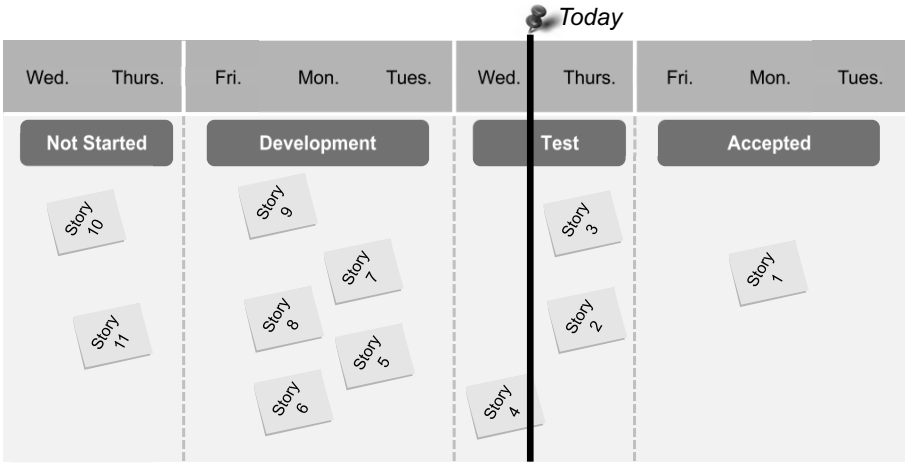
Average wait time = average queue length / average processing rate



Principles of Product Development Flow, Don Reinertsen

Visualize and limit work in process

One team's Big Visible Information Radiator (BVIR)



How is this team doing? How do you know that?

Exercise: Large-batch push

- In your team, choose a four-person group who will process the 10 coins on the table. One additional person is the group timekeeper. Other members are individual timekeepers.
- Each of the four people flips all coins one at a time, recording his own results (heads or tails)
- Then each person passes all coins at the same time to the next person
- The timekeeper records time from the start of the first flip to the completion of the last flip for the group. Each individual timekeeper records time for a single individual.

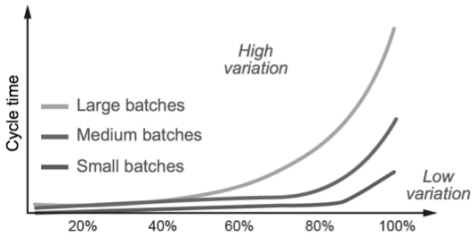
Exercise: Small-batch pull

- Similar four-person process
- Each of the four people flips each coin one at a time and records the result
- People pull coins from the previous person as soon as he is done recording them, and process them immediately
- The timekeeper records the time from the start of the first flip to the completion of the last flip for the group. Each individual timekeeper records time for a single individual.

Reduce batch size

Small batches go through the system faster, with lower variability.

- Large batch sizes increase variability
- High utilization increases variability

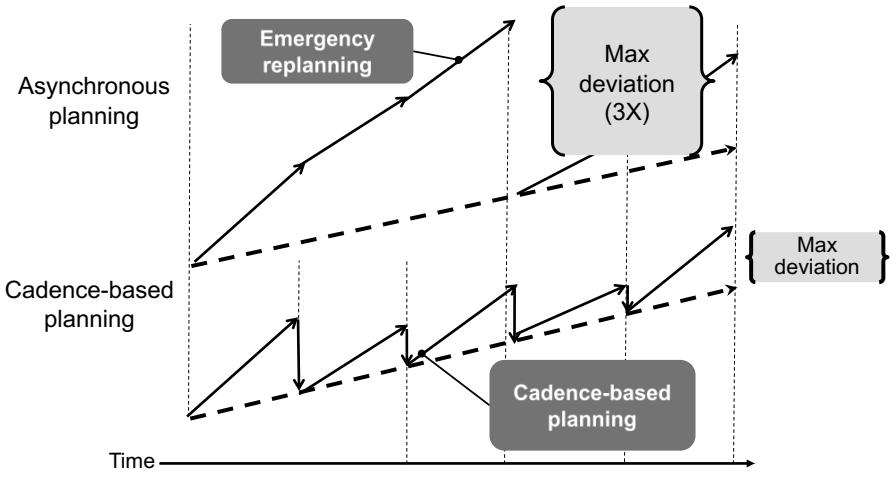


Principles of Product Development Flow, Don Reinertsen
Implementing Lean Software Development, Mary and Tom Poppendieck

- Severe project slippage is the most likely result
- The most important batch is the transport (handoff) batch
- Proximity (collocation) enables small batch size
- Good infrastructure enables small batches

#7 Apply cadence, synchronize with cross-domain planning

Cadence-based planning limits variability to a single interval.



#8 Unlock the intrinsic motivation of knowledge workers

RSA Animate
Drive: The Surprising Truth About What Motivates Us
— Daniel H. Pink

youtube.com/watch?v=u6XAPnuFjJc&feature=youtu.be

#9 Decentralize decision-making

Any inefficiency of decentralization costs less than the value of faster response time.
—Principles of Product Development Flow, Don Reinertsen

- Centralize decisions that:
  - Are infrequent, long lasting, and have significant economies of scale
- Decentralize all others:
  - Frequent decisions
  - Time-critical decisions
  - Decisions that require local information
- Define the economic logic behind a decision; empower individuals and teams to actually make them

1.3 Understand Scrum, Kanban, and XP

1.47

From traditional development to Agile

Instead of a large group

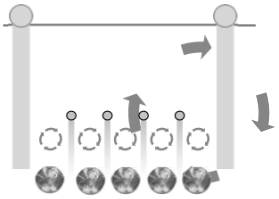
Have small teams working together as a program

Working on all the requirements

Working on small batches of requirements

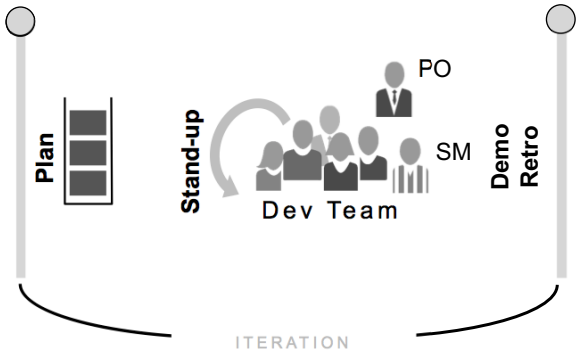
Integrating and delivering value toward the end of development

Delivering value in short timeboxes with frequent integration and improvement cycles



1.48

Scrum in one slide



Three roles (Lesson 2)

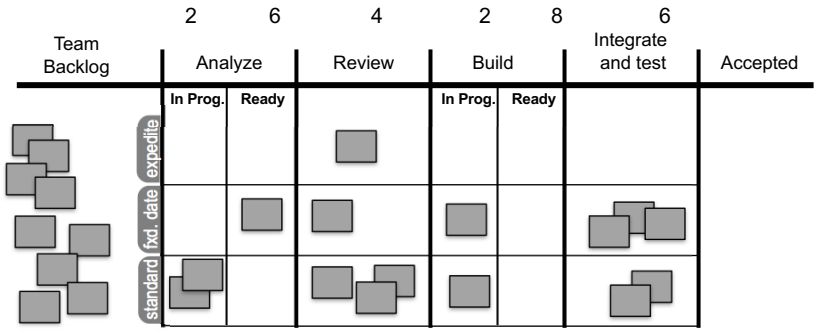
1. Development Team
2. Scrum Master (SM)
3. Product Owner (PO)

Four meetings

1. Iteration Planning (Lesson 3)
2. Daily Stand-up (Lesson 4)
3. Team Demo (Lesson 4)
4. Iteration Retrospective (Lesson 4)

Kanban in one slide

Visualize work flow. Limit work in process. Improve flow.



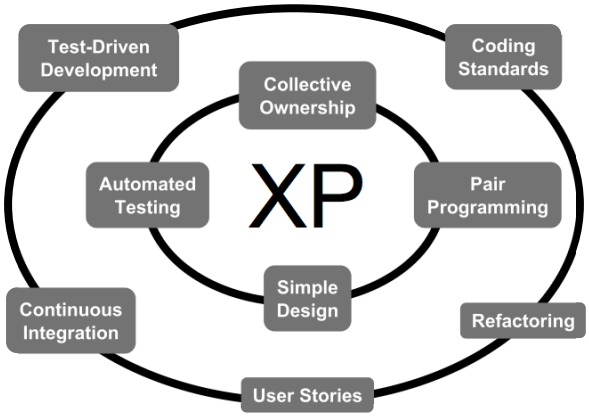
Extreme Programming (XP) in one slide

XP practices drive endemic code quality to unprecedented levels.

Most high-performance teams use Scrum and XP together.

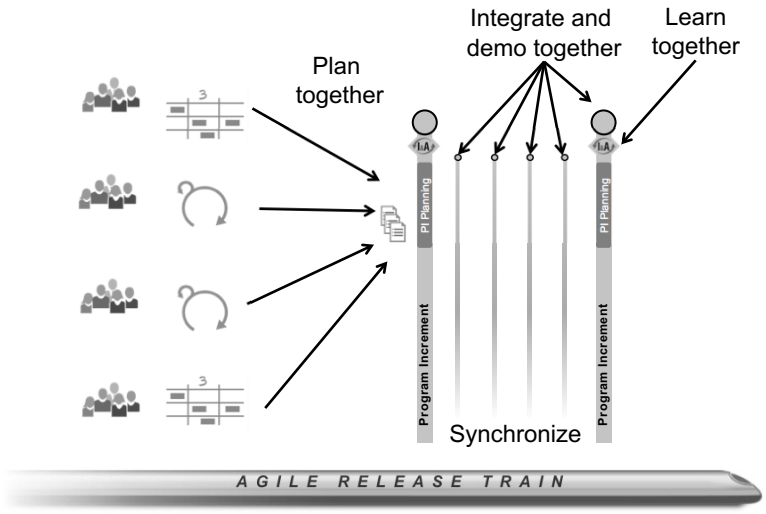
It is hard to get a Scrum with extreme velocity without XP engineering practices.

—Jeff Sutherland, co-creator of Scrum



Adapted from xprogramming.com

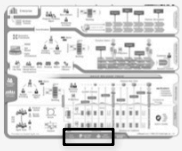
Teams in SAFe are part of an Agile Release Train



Lesson summary

In this lesson, you:

- Learned what SAFe is and its four levels
- Explored the house of Lean, the Agile Manifesto, and SAFe Principles as the foundations of Lean-Agile practices
- Touched upon what Scrum, Kanban and XP are



Suggested Scaled Agile Framework reading:

- *"Core Values," "Lean-Agile Mindset, and "SAFe Principles" articles*

1.53

Lesson 2
Building an Agile Team

Day 1

1. Introducing the Scaled Agile Framework
2. Building an Agile Team
3. Planning the Iteration

Day 2

4. Executing the Iteration
5. Executing the PI

Learning objectives

2.1 Build your team

2.2 Explore the Scrum Master and Product Owner roles

2.3 Meet the teams and people on the train

2.1 Build your team

2.3

The Agile Team

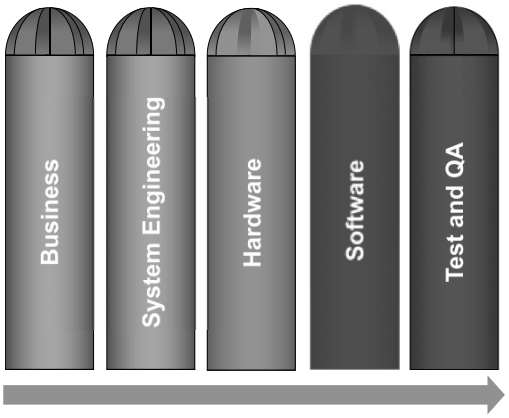
*The 'relay race' approach to product development … may conflict with the goals of maximum speed and flexibility. Instead, a holistic or **'rugby'** approach—where a team tries to go the distance as a unit, passing the ball back and forth—may better serve today's competitive requirements.*

Youtu.be/KWnwl0-aeq0

—Hirotaka Takeuchi and Ikujiro Nonaka, "The New New Product Development Game," *Harvard Business Review,* January 1986

2.4

Value doesn't follow silos

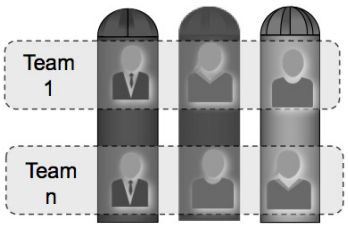


Management challenge:
Connect the silos

- Optimized for vertical communication
- Friction across the silos
- Location via function
- Political boundaries between functions

Build cross-functional Agile Teams

- Cross-functional, self-organizing entities that can define, build, and test a feature or component
- Optimized for communication and delivery of value
- Deliver value every two weeks



The power of “ba”

We, the work, and the knowledge are all one

- Dynamic interaction of individuals and the organization in the form of a self-organizing team; the fuel of “ba” is its self-organizing nature
- Team members create new points of view and resolve contradictions through dialogue
- “Ba” is energized with intentions, vision, interest, and mission
- Leaders provide autonomy, variety, trust, and commitment
- There is creative chaos via demanding performance goals
- The team is challenged to question every norm of development
- Equal access to information at all levels is critical

Nishida, 1921, 1970; *Hitotsubashi on Knowledge Management*, Takeuchi and Nonaka, 2004

youtu.be/yiKFYTFJ_kw

Teamwork

Exercise: Ball game

- You are one team
- Ball must be touched by each person
- Ball must be passed with air time between any two people
- Ball must return to start point before it is counted complete
- Objective is to process as many balls as possible

Instructions

- You will have 2 minutes to organize and begin your first Iteration
- You will do 5 iterations:
  - Each one is 2 minutes, followed by a 1 minute retrospective
- To get credit, you must provide an estimate for the number of balls you think you can process before each iteration

Collocated teams enhance productivity

Collocation:

- Critical for the Agile Team to be effective
- Recommended for programs to have efficient product development flow
- Distributed development must be compensated with efficient remote interaction (video-conferencing, sharing and collaboration tools, Agile project management tools, etc.)

Team working area

Shared team area

Agile Teams power the train

Development Team	▸ Creates and refines user stories and acceptance criteria ▸ Defines/builds/tests/delivers Stories ▸ Develops and commits to Team PI Objectives and Iteration plans
Product Owner	▸ Defines and accepts Stories ▸ Acts as the Customer for developer questions ▸ Works with Product Management to plan Releases ▸ A team has only one Product Owner, who may be dedicated to one or two teams
Scrum Master	▸ Runs team meetings, enforces Agile behavior ▸ Removes impediments, protects the team from outside influence ▸ Attends Scrum of Scrum meetings ▸ May be a part-time role for a team member (25 – 50%), or a single Scrum Master may be shared across 2 – 3 teams

System Team

The System Team provides processes and tools to integrate and evaluate assets early and often

- Builds the development infrastructure and manages environments
- Assists with test automation strategies and adoption
- Provides/supports full system integration
- Performs end-to-end system and performance testing
- Stages and supports the System Demos

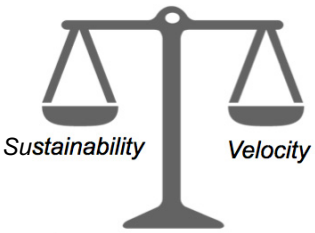
Organizing teams around value

Organize for the larger purpose

- Maximize velocity by minimizing dependencies and handoffs, while sustaining architectural robustness and system qualities

A team can be organized around

- Features
- Components



Far less desirable

- Architectural layer
  - Platform, middleware, UI, DB, business logic, etc.
- Other
  - Programming language, spoken language, technology, location

Finding the right trade-off

Most large programs have a mix

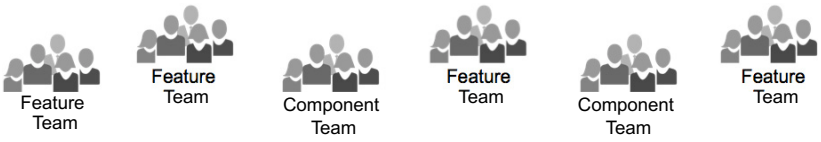
Lean toward Feature Teams:

- Fastest velocity
- Minimize dependencies
- Develop T-shaped skills

Use Component Teams when:

- High reuse, high technical specialization, critical NFRs
- Creating each component as a "potentially replaceable part of the system, with well-defined interfaces"

Generally avoid organizing around architectural layers, as they create team coupling and don't provide a technical separation of concerns



Exercise: Build your team

Full ART Only

As a **team**, we want to understand:

- What are our responsibilities and skill sets?
- Define our team name. Team names should not be the names of components, subsystems, or feature areas.

Instructions

- Discuss your role as a feature or component team
- Discuss what you as a team are responsible for and what other things you can do
- Choose a name for your team
- Prepare a short presentation of your team (name, role on the train, special skills on the team that other teams should know about)

Public Only

Exercise: Build your team

After class go back to your team and discuss:

- What is our team name and why was it chosen?
- Are we a feature or a component team?

2.2 Explore the Scrum Master and Product Owner roles

Roles: Scrum Master

- Coaches the team
- Ensures that the team follows Agile principles and practices
- Facilitates processes and meetings
- Removes impediments and barriers
- Protects the team from external forces

The Scrum Master in the Enterprise

- Coordinates with other Scrum Masters, the System Team, and shared resources in the ART PI Planning meetings
- Works with the above teams throughout each Iteration and PI
- Coordinates with other Scrum Masters and the Release Train Engineer in Scrum of Scrums
- Fosters normalized estimating within the team
- Helps teams operate under architectural and portfolio governance, System Level integration, and System Demos
- Fosters adoption of Agile technical practices

Roles: Product Owner

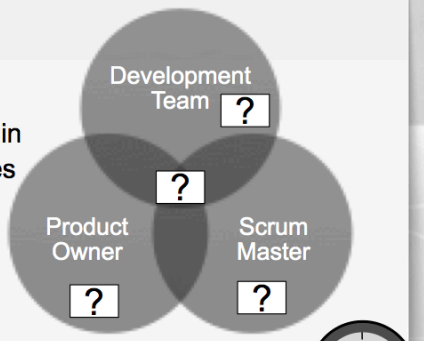
- Member of the Agile Team
- A single voice for the Customer and stakeholders
- Owns and manages the Team Backlog
- Defines and accepts requirements
- Makes the hard calls on scope and content

The Product Owner in the Enterprise

- Establishes the sequence of backlog items based on program priorities, events, and dependencies with other teams
- Operates as part of an extended Product Management Team, usually reporting via a “fat dotted line” to Product Management
- Understands how the Enterprise Backlog Model operates with Epics, Capabilities, Features, and Stories
- Uses PI Objectives and Iteration Goals to communicate with management
- Coordinates with other Product Owners, the System Team, and shared services in the PI Planning meetings
- Works with other Product Owners and the Product Management team throughout each Iteration and PI

Scrum roles and responsibilities card game

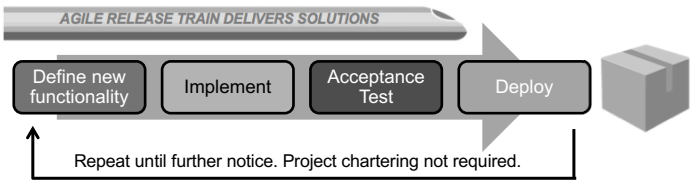
- In your group, draw the following Venn diagram
- Go over the responsibilities sheet in Appendix B and create sticky notes for each, placing them either in a role or at an intersection
- Prepare to discuss your decisions



2.3 Meet the teams and people on the train

The Agile Release Train

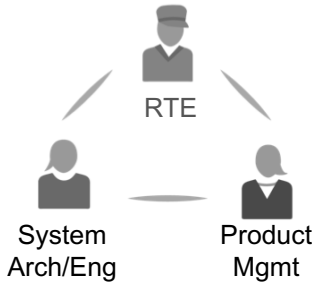
- A virtual organization of 5 – 12 teams (50 – 125+ individuals) that plans, commits, and executes together
- Program Increment (PI) is a fixed timebox; default is 10 weeks
- Synchronized Iterations and PIs
- Aligned to a common mission via a single Program Backlog
- Operates under architectural and UX guidance
- Frequently produces valuable and evaluable system-level Solutions



The ART roles

Three primary roles help ensure successful execution of the Vision and Roadmap initiatives at the Program Level:

1. The Release Train Engineer is a servant leader who facilitates and guides the work of the ART. He acts like a chief Scrum Master.
2. Product Management is the main content authority guiding the train. They own and prioritize the Program Backlog.
3. The System Architect/Engineer has the technical responsibility for the overall architectural and engineering design of the system. She provides architectural and technical guidance to the teams on the train.



Full ART Only

Exercise: Know the people on the train

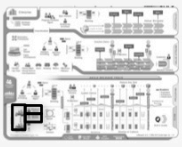
The RTE presents himself and the main players on the train:

- Product Management
- System Architect/Engineering
- UX
- DevOps
- Shared Services
- Each team presents itself (name, area of responsibility, special skills)

Lesson summary

In this lesson, you:

- Learned who your team is and its role on the train
- Explored the roles of the Scrum Master and the Product Owner and how they fit into your context
- Met the people and teams on the train and learned their role in it



Suggested Scaled Agile Framework reading:

- *"Agile Teams" article*
- *"Scrum Master" article*
- *"Product Owner" article*

Lesson 3
Planning the Iteration

Day 1

1. Introducing the Scaled Agile Framework
2. Building an Agile Team
3. Planning the Iteration

Day 2

4. Executing the Iteration
5. Executing the PI

Learning objectives

3.1 Prepare the Backlog

3.2 Plan the Iteration

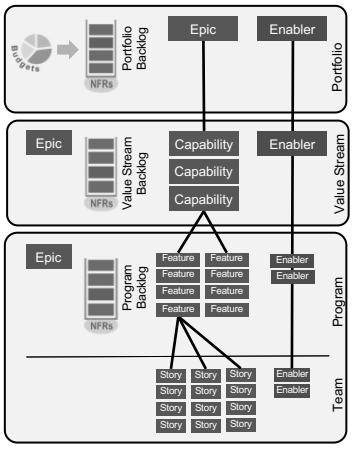
3.1 Prepare the Backlog

3.3

Define Solution Features for the Program Backlog

Features are services that fulfill user needs.

- "Feature" is an industry-standard term familiar to marketing and Product Management
- Expressed as a phrase, "value" is expressed in terms of benefits
- Identified, prioritized, estimated, and maintained in the Program Backlog



3.4

Features have benefits and acceptance criteria

- Benefits justify Feature implementation cost, provide business perspective when making scope decisions
- Business benefits impact economic prioritization of the Feature
- Acceptance criteria are typically defined during Program Backlog refinement
- Reflect functional and Nonfunctional Requirements

SSO example:

Multifactor authentication

Business benefit

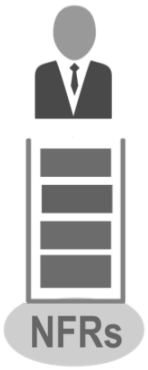
Enhance user security via both password and a device.

Acceptance criteria

1. USB tokens as a first layer
2. Password authentication as a second layer
3. Multiple tokens on a single device
4. User activity log reflecting both authentication factors

The Team Backlog

- Contains all the work the team needs to work on
- Created by the Product Owner and the team
- Prioritized by the Product Owner
- Contains User and Enabler Stories
  - User stories provide Customers with value
  - Enabler Stories build the infrastructure and architectures that makes user stories possible
- Stories in the backlog are prioritized
- Stories for the next Iteration are more detailed than Stories for later Iterations
- Nonfunctional Requirements are a constraint on the backlog



User stories

- Containers for user or Customer value
- Written using the following template:

As a <user role> I can <activity> so that <business value>

- **User role** is the description of the person doing the action
- **Activity** is what they can do with the system
- **Business value** is why they want to do the activity

As a driver, I can limit the amount of money before I fuel so that I can control my expenditure

As a driver, I can get a receipt after fueling so that I can expense the purchase

As the Finance Department, we can print receipts only for drivers who request them so that we can save on paper

(Roles can be people, devices, or systems)

User Story guidelines — The 3 Cs

Card

Written on a card or in the tool and may annotate with notes

As a spouse, I want a clean garage so that I can park my car and not trip on my way to the door

Conversation

The details are in a conversation with the Product Owner

What about the bikes?

Oh yeah, hang the bikes

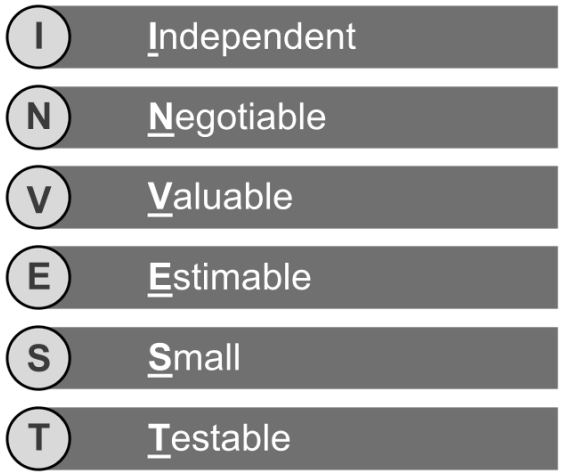
Confirmation

Acceptance criteria confirm the story correctness

- Tools have been put away
- Items on the floor have been returned to the proper shelf
- Bikes have been hung

Source: 3 Cs coined by Ron Jeffries

INVEST in a good Story



Enabler Stories

- Enabler Stories build the groundwork for future user stories
- Three types of Enabler Stories:
  1. Infrastructure – Build development and testing frameworks that enable a faster and more efficient development process
  2. Architecture – Build the Architectural Runway, which enables smoother and faster development
  3. Exploration – Build understanding of what is needed by the customer, to understand prospective solutions and evaluate alternatives

Splitting Features and Stories

Techniques for splitting Features and Stories to fit within their boundaries (PI and Iteration, respectively)

1. Work flow steps
2. Business rule variations
3. Major effort
4. Simple/complex
5. Variations in data
6. Data methods
7. Defer system qualities
8. Operations
9. Use-case scenarios
10. Break out a spike

Exercise: Break Features into Stories

Work with your team to break Features from the Program Backlog into Stories that are small enough to fit into an Iteration

Instructions

- Break a Feature into 5 – 10 Stories
- Make sure the Stories retain a business value
- Try to create Stories that are less than a week in size
- Identify spikes as needed

Acceptance criteria

- Have at least 5 Stories from your Features

Acceptance criteria

- Acceptance criteria provide the details of the Story from a testing point of view
- Acceptance criteria are created by the Team and the PO

As a driver, I can limit the amount of money before I fuel so that I can control my expenditure.

Acceptance criteria

1. The fueling process stops automatically on the exact value
2. I can stop fueling before the limit has been reached and will only be charged for the amount fueled

As a driver, I can get a receipt after fueling so that I can expense the purchase.

Acceptance criteria

1. Receipt includes: Amount fueled, Amount Paid, Tax, Vehicle number, Date, Time

Exercise: Write acceptance criteria

- Write acceptance criteria for the Stories you have identified
- Make sure the acceptance criteria are testable
- The Product Owner needs to approve the acceptance criteria

Estimate Stories with relative Story points

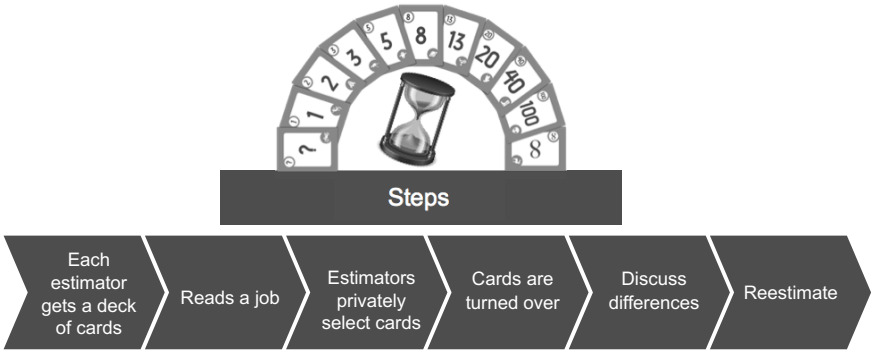
- A Story point is a singular number that represents:
  - Volume: How much is there?
  - Complexity: How hard is it?
  - Knowledge: What do we know?
  - Uncertainty: What's not known?
- Story points are relative; they are not connected to any specific unit of measure
- Compare with other Stories (an 8-point Story should take 4 times longer than a 2-point Story)

How *big* is it?

?

Apply Estimating Poker for fast, relative estimating

- Estimating Poker combines expert opinion, analogy, and disaggregation for quick but reliable estimates
- All team members participate



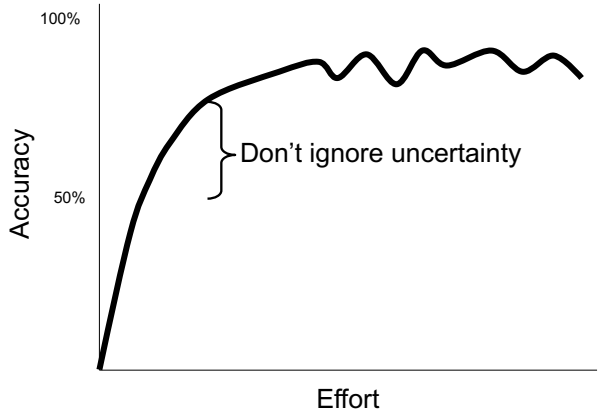
Mike Cohn, *Agile Estimating and Planning*, 2005

Estimation is a whole-team exercise

- Increases accuracy by including all perspectives
- Builds understanding
- Creates shared commitment
- Estimation performed by a manager, architect, or select group negates these benefits

How much time to spend estimating

- A little effort helps a lot
- A lot of effort only helps a little



Exercise: Estimate Stories

- Estimate the Stories you have identified
- Use Estimating Poker together as a group
- The Scrum Master will facilitate the activity
- The Product Owner doesn't vote but can ask or answer questions

Acceptance criteria

At least three Stories are estimated in Story points

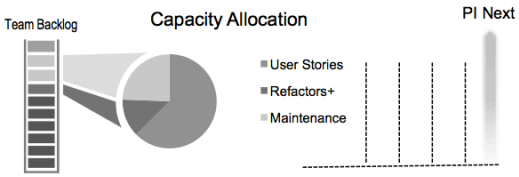
Sequencing Stories

Primary economic prioritization happens at the Program Backlog. Agile Teams sequence work for efficient execution of business priorities.

- The Product Owner and the Team sequence work based on:
  - Story priorities inherited from Program Backlog priorities
  - Events, Milestones, releases, and other commitments made during PI Planning
  - Dependencies with other teams
  - Local priorities
  - Capacity allocations for defects, maintenance, and refactors
- Initial sequencing happens during PI Planning
- Adjustments happen at Iteration boundaries

Capacity allocation for a healthy balance

- By having capacity allocation defined, the Product Owner doesn't need to prioritize unlike things against each other
- Once the capacity allocation is set, the PO and team can prioritize like things against each other



Notes:

1. Helps alleviate velocity degradation due to technical debt
2. Keeps existing Customers happy with bug fixes and enhancements
3. Can change at Iteration or PI boundaries

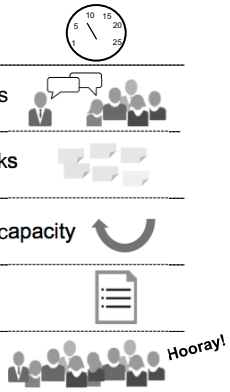
3.2 Plan the Iteration

Plan and commit

Purpose	Define and commit to what will be built in the Iteration
Process	▸ The Product Owner defines *what* ▸ The team defines *how* and *how much* ▸ Four hours max
Result	Iteration Goals and backlog of the team's commitment
Reciprocal commitment	▸ Team commits to delivering specific value ▸ Business commits to leaving priorities unchanged during the Iteration

Iteration Planning flow

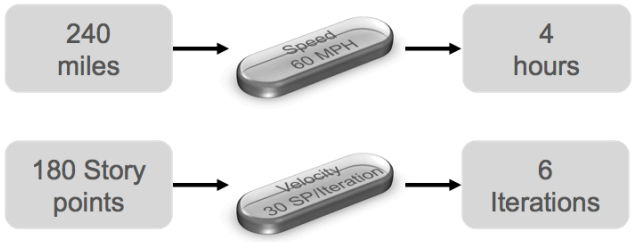
1. The team establishes its velocity
2. The team clarifies and estimates the Stories
3. The team optionally breaks Stories into tasks
4. The process continues while there is more capacity
5. The team synthesizes Iteration Goals
6. Everyone commits



- Timebox: 4 hours
- This meeting is by and for the team
- Subject matter experts (SMEs) may attend as required

Using size to estimate duration

Examples

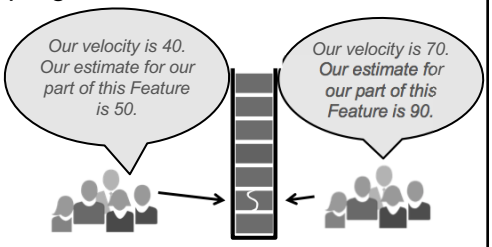


Establish velocity by looking at the average output of the last Iterations.

Establishing velocity before historical data exists

Normalized Story point estimating provides the economic basis for estimating work within and across programs.

1. For every full-time developer and tester on the team, give the team eight points (adjust for part-timers)
2. Subtract one point for every team member vacation day and holiday
3. Find a small Story that would take about a half-day to code and a half-day to test and validate. Call it a **1**.
4. Estimate every other Story relative to that one
5. Never look back (don't worry about recalibrating)



The cost of a Story point is about the same for each of these teams.

Example: Assuming a 6-person team composed of 3 developers, 2 testers, 1 PO, with no vacations, etc.

Estimated velocity = 5 * 8 pts = 40 pts/Iteration

Exercise: Calculate your initial velocity

Use the "Starting fast with capacity-based planning" slide on the previous page to calculate your team's starting velocity.

Instructions

- Calculate your estimated velocity for the next two-week Iteration, which starts after PI Planning
- Use your real availability

Acceptance criteria

- Each team has their estimated velocity for two Iterations

Iteration Goals

Iteration Goals provide clarity, commitment, and management information

They serve three purposes:

1. Align team members to a common purpose
2. Align Program Teams to common PI Objectives. Manage dependencies.
3. Provide continuous management information

Iteration Goals example

1. Finalize and push last-name search and first-name morphology
2. Index 80% of remaining data
3. Other Stories:
   - Establish search replication validation protocol
   - Refactor artifact dictionary schema

Story analysis and estimation

- The Product Owner presents Stories in order of priority
- Each Story
  - Is discussed and analyzed by the team
  - Has its acceptance criteria refined
  - Is estimated
- The process continues until the estimation of the Stories has reached the velocity of the team

Exercise: Plan your first Iteration

Use the team's initial velocity and the estimated Stories to allocate Stories to the first Iteration

Instructions

- Plan which Stories fit into the first Iteration
- Estimate more Stories as needed to fill the first Iteration

Acceptance criteria

- Each team has a list of Stories that they can deliver by the end of the first Iteration

Commit to the Iteration Goals

A Team meets its commitment:

By doing everything they said they would do

- or -

In the event that it is not feasible, they must immediately raise a red flag

Commitment

Too much holding to a commitment can lead to burnout, inflexibility, and quality problems

Adaptability

Too little commitment can lead to unpredictability and lack of focus on results

Team commitments are not just to the work. They are committed to other teams, the program, and the stakeholders.

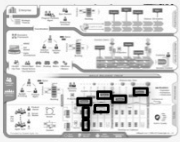
Iteration planning for Kanban teams

- Some teams have a more responsive nature to their work, such as maintenance teams, System Teams, Dev Ops
- These teams find less value in trying to plan the Iteration in detail
- Kanban teams still publish Iteration Goals, which consist of the known parts of their work
- They commit to the goals as well as to a cycle time SLA for incoming work based on their known historical data

Lesson summary

In this lesson, you:

- Learned what features and stories are
- Broke features into stories
- Written stories with acceptance criteria
- Estimated stories using relative estimates with story points
- Practiced planning an iteration



Suggested Scaled Agile Framework reading:

- *"Stories" article*
- *"Features and Capabilities" article*
- *"Iteration Planning" article*

3.33

Lesson 4
Executing the Iteration

Day 1

1. Introducing the Scaled Agile Framework
2. Building an Agile Team
3. Planning the Iteration

Day 2

4. Executing the Iteration
5. Executing the PI

Learning objectives

4.1 Visualize the flow of work

4.2 Measure the flow of value

4.3 Build quality in

4.4 Control flow with meetings

4.5 Demo value

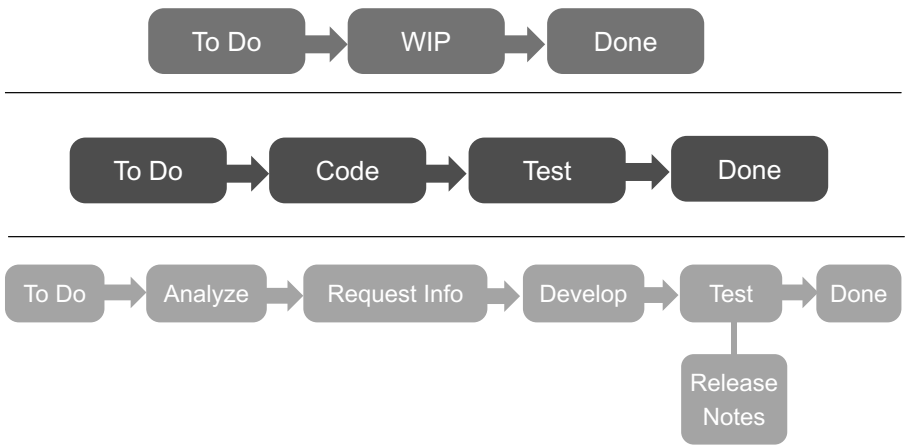
4.6 Retrospect and improve

4.1 Visualize the flow of work

4.3

Visualize the flow of work

- What is the flow of work for your team?
- What are the steps it takes to get a Story to Done?



4.4

Setting WIP limits

WIP limits can apply to a single step or to multiple steps. Some steps have no WIP limits, while others serve as buffers and have minimal as well as maximal WIP.

		5	4-7		3	
	To Do	*Analyze*	*Ready*	*Develop*	*Test*	*Done*
					Release Notes	

Exercise: Build your initial board

Consider the previous three slides and build an initial board for your team.

Instructions

- Define the steps you need to turn Stories into value
- Build the structure of the board
- Assign initial WIP limits

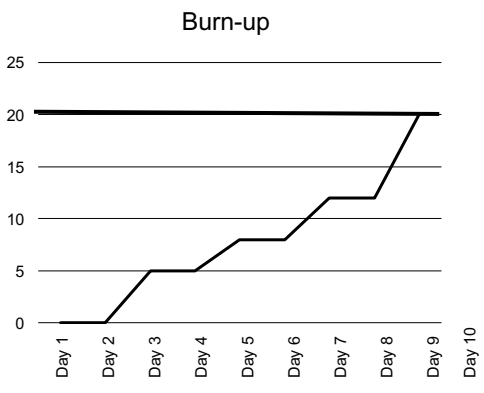
Acceptance criteria

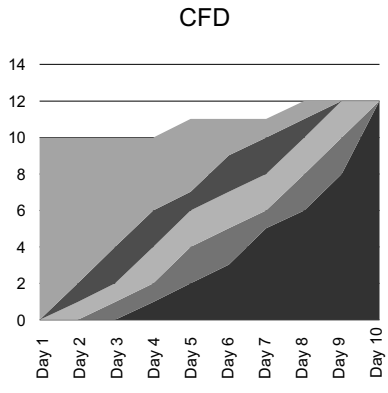
- Each team has a board with WIP limits

4.2 Measure the flow of value

4.7

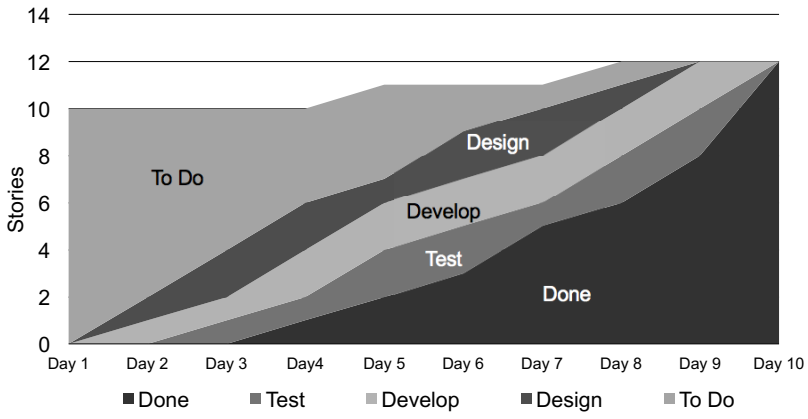
Track status with burn-up charts and CFDs



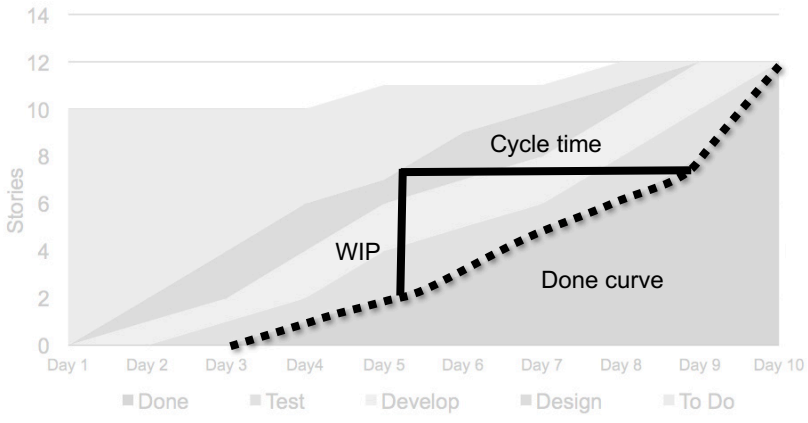


4.8

Understand Cumulative Flow Diagrams (CFD)



What can you learn from a CFD?



4.3 Build quality in

4.11

Built-In Quality

"You can't scale crappy code" (or hardware, or anything else).

Building quality in:

- Ensures that every increment of the Solution reflects quality standards
- Is required for high, sustainable development velocity
- Software quality practices (most inspired by XP) include Continuous Integration, Test-First, refactoring, pair work, collective ownership, and more
- Hardware quality is supported by exploratory, early Iterations; frequent system-level integration; design verification; MBSE; and Set-Based Design

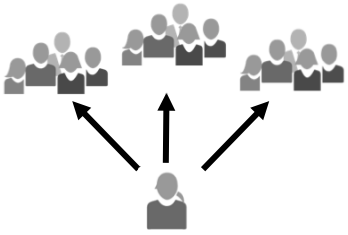
Built-In Quality

4.12

Emergent design and intentional architecture

Every team deserves to see the bigger picture. Every team is empowered to design their part.

- Emergent design – Teams grow the system design as user stories require
- Intentional architecture – Fosters team alignment and defines the Architectural Runway

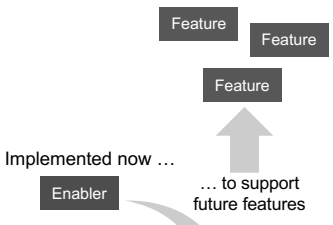


A balance between emergent design and intentional architecture is required for speed of development and maintainability.

Architectural Runway

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

Example: A new, fuzzy search algorithm will enable a variety of future features that can accept potentially erroneous user input



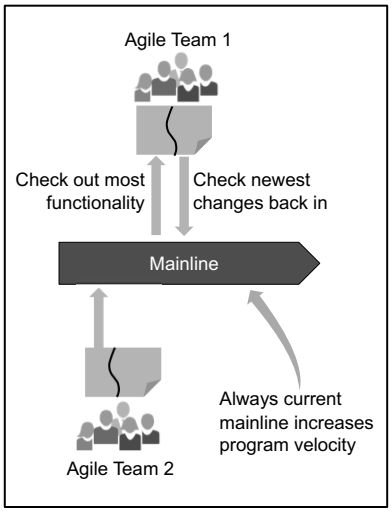
- Enablers build up the runway
- Features consume it
- Architectural Runway must be continuously maintained
- Use capacity allocation (a percentage of the train's overall capacity in a PI) for Enablers that extend the runway

Architectural Runway

Continuous system integration

Teams continuously integrate assets (leaving as little as possible to the System Team).

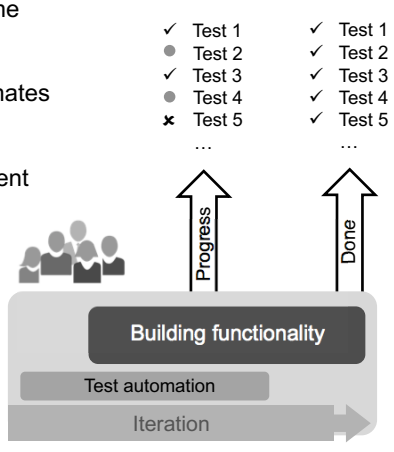
- Integrate every vertical slice of a user story
- Avoid physical branching for software
- Frequently integrate hardware branches
- Use development by intention in case of inter-team dependencies
  - Define interfaces and integrate first, then add functionality



Test first: Automate now!

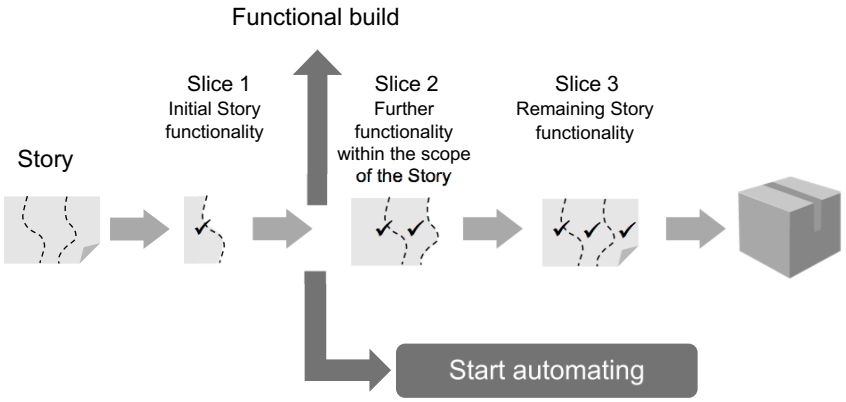
Otherwise velocity is bottlenecked, quality is speculative, and scaling is impossible.

- Automated tests are implemented in the same Iteration as the functionality
- The team that builds functionality also automates the tests
- Create an isolated automated test environment
- Actively maintain test data under version control
- Passing vs. not-yet-passing and broken automated tests are the *real* Iteration progress indicator



Transition to early automation

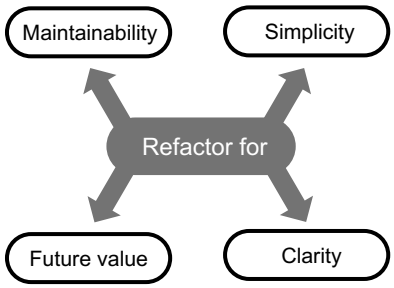
Start automating as soon as the initial functionality is in place.



Refactoring

Refactoring allows teams to maintain high velocity.

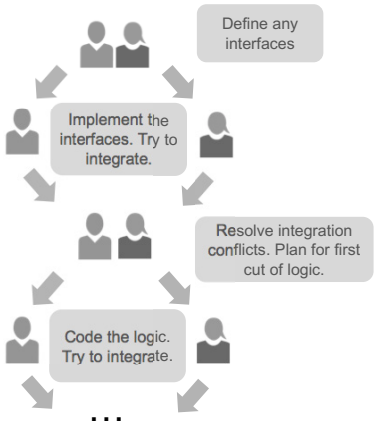
- It is impossible to predict requirements or design in detail
- Refactoring allows teams to quickly correct the course of action
- Emergent design is impossible without continuous refactoring
- Most user stories will include some refactoring effort
- If technical debt is big – teams track and implement as separate backlog items – then refactor



Pair work

Pair work improves system quality, design decisions, knowledge sharing, and team velocity.

- Pair work is …
  - Broader and less constraining than pair programming
  - A collaborative effort of any two team members: dev/dev, dev/PO, dev/tester, etc.
- Team members spend 20% to 80% time pairing
- Spontaneous pairing, and purposeful rotation over time

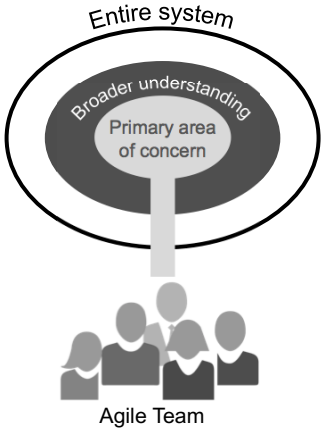


Example user story implementation flow

Collective ownership

Collective ownership addresses bottlenecks, increases velocity, and encourages shared contribution.

- Collective ownership fosters Feature orientation
- Collective test ownership is even more important—it facilitates shared understanding of system behavior
- Collective ownership is supported by:
  - Design simplicity
  - Communities of Practice
  - Pair work
  - Joint specification and design workshops
  - Frequent integration of the entire system
  - Standards



Agile Team

Teams gain a broader understanding of the system

Capture with models, not documents

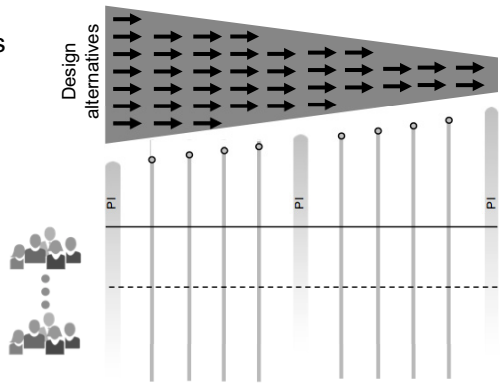
- Models are more adaptable than documents
  - Single changes propagate to all uses
  - Everyone can contribute—all domains, team members
  - Faster, and automated, verification and validation
- Models are more powerful than documents
  - Increases rigor, consistency, accuracy
  - Enables earlier verification and validation
  - Facilitates more strategic reuse
  - Automates document generation (compliance)
  - Provides traceability

MBSE
Model-Based Systems Engineering

Set-Based Design

Continuously convert uncertainty to knowledge.

- Emphasizes design *discovery* over design *creation*
- Concurrently explores multiple designs options to find an optimum, rather than the first, Solution
- Understands design trade-offs before detailing specifications



4.4 Control flow with meetings

4.23

Leading the daily stand-up (DSU)

The DSU is the key to team synchronization and self-organization.

The DSU (or daily Scrum) is not a daily status meeting for management.

It is used to:

- Share information about progress
- Coordinate activities
- Raise blocking issues

Time is whenever is most convenient for the team.

- Every day at the same time in front of the team board
- Timebox of 15 minutes
- Not a problem-solving session
- Update the board

4.24

Daily stand-up patterns

Basic Scrum pattern meeting agenda

Each person answers:

1. What did I do yesterday to advance the Iteration Goals?
2. What will I do today to advance the Iteration Goals?
3. Are there any impediments that will prevent the team from meeting the Iteration Goals?

The meet-after agenda

1. Review topics the Scrum Master wrote on the meet-after board
2. Involved parties discuss, uninvolved people leave

Exercise: Simulate a daily stand-up meeting

- Let's simulate a DSU in front of the class
- 4 – 5 volunteers
- Instructor will play the Scrum Master role

Full ART Only

Exercise: Agree on location and timing for the DSU

- As a team, we want to decide on the place and time for our DSU
- Agree on a time that works for everyone
- Take remote people into account

The backlog refinement session

The backlog refinement session is a preview and elaboration of upcoming Stories.

- Helps the team “sleep” on new Stories prior to the Iteration Planning
- Provides enough time to identify and resolve dependencies and issues that could impact the next Iteration
- The team can improve Stories, add acceptance criteria, and point out missing information to the PO
- Most of the focus is on the next Iteration, but it allows time to discuss future Iterations and even Features for the next PI

4.5 Demo value

4.29

The Team Demo

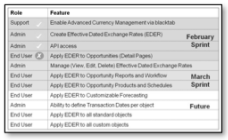
- Provides the true measure of progress by showing working software functionality, hardware components, etc.
- Preparation for the Team Demo starts with planning
- Teams demonstrate every Story, spike, refactor, and NFR
- Attendees are the team and its stakeholders
- Ad hoc demos — don't wait for the Team Demo; demo Stories when they are done

Note: This is the Team Demo, not the System Demo

4.30

Team Demo guidelines

- Timebox: 1 – 2 hours
- Story demo preparation should be limited to 1 – 2 hours. Minimize PowerPoint. Work from the repository of Stories.
- If a major stakeholder cannot attend, the Product Owner should follow up individually.



Tooling is often used to facilitate the demo

Sample Team Iteration Demo Agenda

1. Review business context and Iteration Goals
2. Demo and feedback of each Story, spike, refactor, and NFR
3. Discussion of Stories not completed and why
4. Current risks and impediments
5. Revised PI burn-down chart, changes to the release plan (if any), and what is coming next

The Team Demo: Two views

The Team Demo provides two views into the program:

1. How we did on the Iteration
   - Did we meet the goals?
   - Story by Story review

2. How we're doing on the PI
   - Review of PI objectives
   - Review remaining PI scope and reprioritize if necessary

SAFe Definition of Done

Team Increment	System Increment	Solution Increment	Release
• Stories satisfy acceptance criteria • Acceptance tests passed (automated where practical) • Unit and component tests coded, passed, and included in the BVT • Cumulative unit tests passed • Assets under version control • Engineering standards followed • NFRs met • No must-fix defects • Stories accepted by Product Owner	• Stories completed by all teams in the ART and integrated • Completed Features meet acceptance criteria • NFRs met • No must-fix defects • Verification and validation of key scenarios • Included in build definition and deployment process • Increment demonstrated, feedback achieved • Accepted by Product Management	• Capabilities completed by all trains and have met acceptance criteria • Deployed/installed in the staging environment • NFRs met • System end-to-end integration, verification, and validation done • No must-fix defects • Included in build definition and deployment/transition process • Documentation updated • Solution demonstrated, feedback achieved • Accepted by Solution Management	• All Capabilities done and acceptance criteria met • End-to-end integration and Solution V&V done • Regression testing done • NFRs met • No must-fix defects • Release documentation complete • All standards met • Approved by Solution and Release Management

Exercise: Define your Definition of Done

Consider the team Increment Definition of Done in the previous slide, then create your own Definition of Done for Stories.

Instructions

- As a team, build a definition of what it means to you to finish a Story. Be ready to present.

4.6 Retrospect and improve

4.35

Iteration Retrospective

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

- 30 – 60 minutes
- Pick 1 – 2 things that can be done better, target for next Iteration
- Enter improvement items into the Team Backlog

Sample Agenda

Part 1: Quantitative

1. Review the improvement backlog items targeted for this Iteration. Were they all accomplished?
2. Did the team meet the goals (yes/no)?
3. Collect and review the agreed-to Iteration print metrics.

Part 2: Qualitative

1. What went well?
2. What didn't?
3. What we can do better next time?

4.36

Iteration Metrics

Functionality	Iteration 1	Iteration 2
# Stories (loaded at beginning of Iteration)		
# accepted Stories (defined, built, tested, and accepted)		
% accepted		
# not accepted (not achieved within the Iteration)		
# pushed to next Iteration (rescheduled in next Iteration)		
# not accepted: deferred to later date		
# not accepted: deleted from backlog		
# added (during Iteration; should typically be 0)		

Quality and test automation
% SC with test available/test automated
Defect count at start of Iteration
Defect count at end of Iteration
 # new test cases
 # new test cases automated
 # new manual test cases
Total automated tests
Total manual tests
% tests automated
Unit test coverage percentage

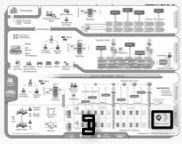
Exercise: Run a short retrospective

- As a group, let's retrospect the course so far.
- What went well?
- What didn't go so well?
- What can we improve for the next time we run this course?

Lesson summary

In this lesson, you:

- Defined the initial flow of your team, built your team board and your WIP limits
- Learned how to measure flow
- Recognized techniques to build quality into development process
- Explored how to demo value to team stakeholders
- Practiced running retrospectives



Suggested Scaled Agile Framework reading:

- *"Built-in Quality" article*
- *"Iteration Execution" article*
- *"Team Demo" article*
- *"Iteration Retrospective" article*

4.39

Lesson 5
Executing the PI

Day 1

1. Introducing the Scaled Agile Framework
2. Building an Agile Team
3. Planning the Iteration

Day 2

4. Executing the Iteration
5. Executing the PI

Learning objectives

5.1 Plan together

5.2 Integrate and demo together

5.3 Learn together

5.1 Plan together

PI Planning

Cadence-based PI Planning meetings are the pacemaker of the Agile Enterprise.

- Two days every 8 – 12 weeks (10 weeks typical)
- Everyone attends in person if at all possible
- Product Management owns Feature priorities
- Development teams own story-planning and high-level estimates
- Architects/Engineering and UX work as intermediaries for governance, interfaces, and dependencies

The goal of PI Planning

Alignment to a common mission!

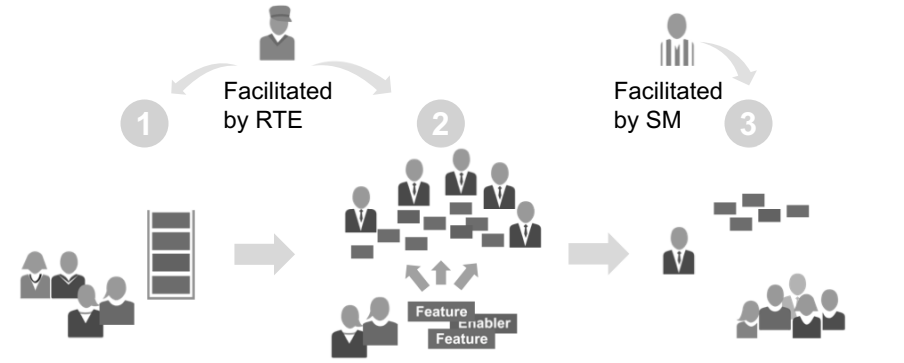
We are here to gain alignment and commitment around a clear set of prioritized objectives. I will now review the agenda for the next two days of the PI Planning event.

Day 1 agenda

8:00–9:00	Business context		State of the business and upcoming objectives
9:00–10:30	Product/Solution Vision		Vision and prioritized Features
10:30–11:30	Architecture Vision and development practices		▸ Architecture, common frameworks, etc. ▸ Agile tooling, engineering practices, etc.
11:30–1:00	Planning context and lunch		Facilitator explains planning process
1:00–4:00	Team breakouts	1 2 3 4	▸ Teams develop draft plans and identify risks and impediments ▸ Architects and Product Managers circulate
4:00–5:00	Draft plan review		Teams present draft plans, risks, and impediments
5:00–6:00	Management review and problem-solving		Adjustments made based on challenges, risks, and impediments

New PI content should better not be a surprise

Upfront presentation of content to the teams solves a lot of problems later, during PI Planning.

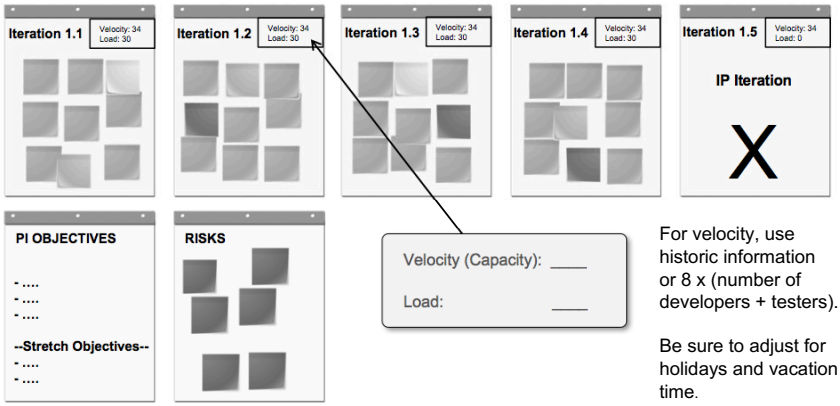


Product Management and other stakeholders refine Program Backlog

Features are roughly split into story-like chunks with POs

Each PO presents initial stories to the team, looks for feedback, big unknowns, etc.

Team breakout #1



For velocity, use historic information or 8 x (number of developers + testers).

Be sure to adjust for holidays and vacation time.

Color coding gives visibility into investments

Green	= User stories	Yellow	= Exploration Enablers	Red/ Pink	= Risks and dependencies
Orange	= Infrastructure/ Enablers	Purple	= Maintenance	Re Pink	= Addressed risks and dependencies

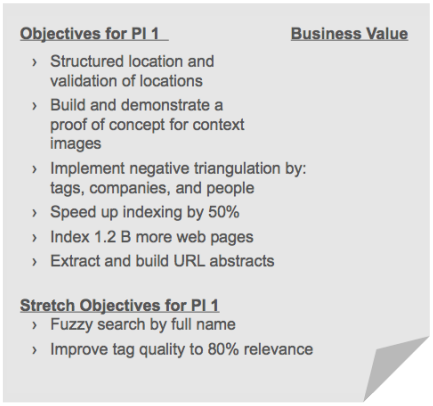
Align to a mission with PI Objectives

Objectives are business summaries of what each team intends to deliver in the upcoming PI.

They often map directly to the features in the backlog.... But not always.

For example:

- Aggregation of a set of Features, stated in more concise terms
- A Milestone like a trade show
- An Enabler Feature needed to support the implementation
- A major refactoring



Stretch objectives

Stretch objectives provide a reliability guard band.

Stretch objectives **do** count in velocity/capacity:

- They are planned, not extra things teams do “just in case you have time”
- But, they are **not** included in the commitment, thereby making the commitment more reliable
- If a team has low confidence in meeting a PI Objective, encourage them to move it to stretch
- If an item has many unknowns, consider moving it to stretch, and put in early spikes

SMART Team PI Objectives

Teams should write their PI Objectives in the "SMART" format.

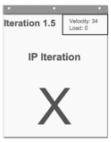
- **S**pecific States the intended outcome as simply, concisely, and explicitly as possible (Hint: Try starting with an action verb)
- **M**easurable It should be clear what a team needs to do to achieve the objective. The measures may be descriptive, yes/no, quantitative, or provide a range.
- **A**chievable Achieving the objective should be within the team's control and influence
- **R**ealistic Recognize factors that cannot be controlled. (Hint: Avoid making "happy path" assumptions)
- **T**ime-bound The time period for achievement must be within the PI, and therefore all objectives must be scoped appropriately

Team deliverables detail

Iterations

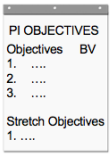
- If a Story has a dependency, put a red sticky on it describing the dependency. Put a check mark through it once the dependency has been addressed.
- If a risk is broader in nature, put it on the risk sheet
- If needed, allocate a percentage of capacity for unplanned activities like maintenance and production support

IP



- The last Iteration will be used for Innovation and Planning (IP)
- You should have a velocity but not a load on the IP Iteration, since it should not contain any user value stories

Objectives

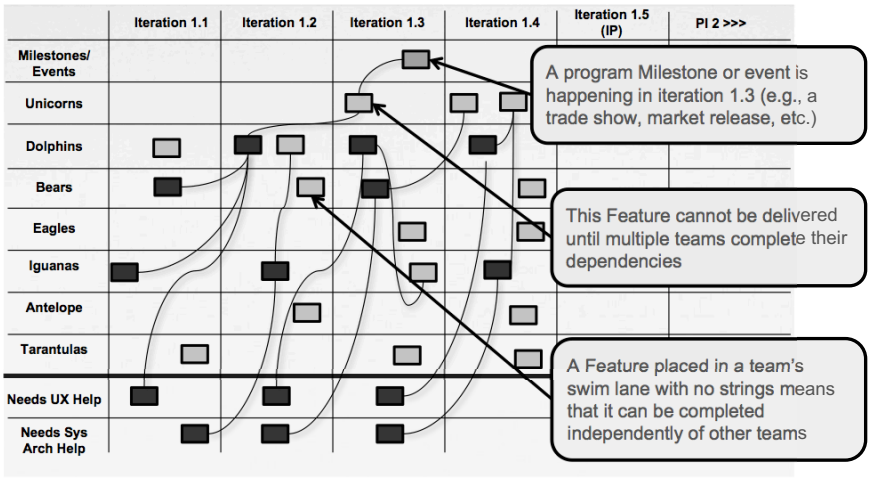


- PI Objectives should be written as "SMART" objectives
- Objectives are assigned business value during the second team breakout
- Stories supporting stretch objectives are included in the load calculation

Risks

- Program risks are those that need to be escalated to the program level. They will be captured and "ROAMed" after the final plan review.
- Team risks are those under the team's control. They won't be presented.

Program board: Feature delivery, dependencies, and Milestones



Blue = Features

Red/ Pink = Significant dependency

Orange = Milestone/ event

Red String = A dependency requiring Stories or other dependencies to be completed before the Feature can be completed

5.13

Exercise: Identify dependencies

- In your team, review the Stories you created from your Features and identify dependencies with other teams
- If you are sitting with your whole ART:
  - Which teams are you likely to be dependent on regularly?
  - Which teams do you think will regularly dependend on you?

Acceptance criteria

- A list of dependencies for your Stories or Features
- Be ready to present the teams you depend on and that depend on you

5.14

Management review and problem-solving

At Day 1 end, management meets to make adjustments to scope and objectives based on the day's planning.

Common questions during the manager's review:

- What did we just learn?
- Where do we need to adjust Vision? Scope? Resources?
- Where are the bottlenecks?
- What Features must be de-scoped?
- What decisions must we make between now and tomorrow to address these issues?

Used with permission of Hybris Software

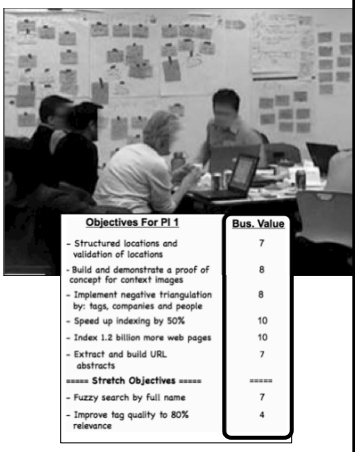
Day 2 agenda

8:00–9:00	Planning adjustments	Planning adjustments made based on previous day's management meeting
9:00–11:00	Team breakouts	▸ Teams develop final plans and refine risks and impediments ▸ Business Owners circulate and assign business value to team objectives
11:00–1:00	Final plan review and lunch	Teams present final plans, risks, and impediments
1:00–2:00	Program risks	Remaining Program Level risks are discussed and ROAMed
2:00–2:15	PI confidence vote	Team and program confidence vote
2:15–???	Plan rework if necessary	If necessary, planning continues until commitment is achieved
After commitment	Planning retrospective and moving forward	▸ Retrospective ▸ Moving forward ▸ Final instructions Presented by RTE

Team breakout #2

Based on new knowledge (and a good night's sleep), teams work to create their final plans.

- In the second team breakout, Business Owners circulate and assign business value to PI Objectives from low (1) to high (10)
- Teams finalize the Program Increment plan
- Teams also consolidate program risks, impediments, and dependencies
- Stretch objectives provide the capacity and guard band needed to increase cadence-based delivery reliability



Addressing program risks

After all plans had been presented, remaining program risks and impediments are discussed and categorized.

ROAMing risks:

- **R**esolved – Has been addressed; no longer a concern
- **O**wned – Someone has taken responsibility
- **A**ccepted – Nothing more can be done. If risk occurs, PI may not yield the planned results.
- **M**itigated – Team has plan to adjust as necessary

Confidence votes: Team and Program

After dependencies are resolved and risks are addressed, a confidence vote is taken at the Team and Program Levels.

"Fist of five" confidence vote

Range of 1 through 5:

1 = No confidence

5 = Very high confidence

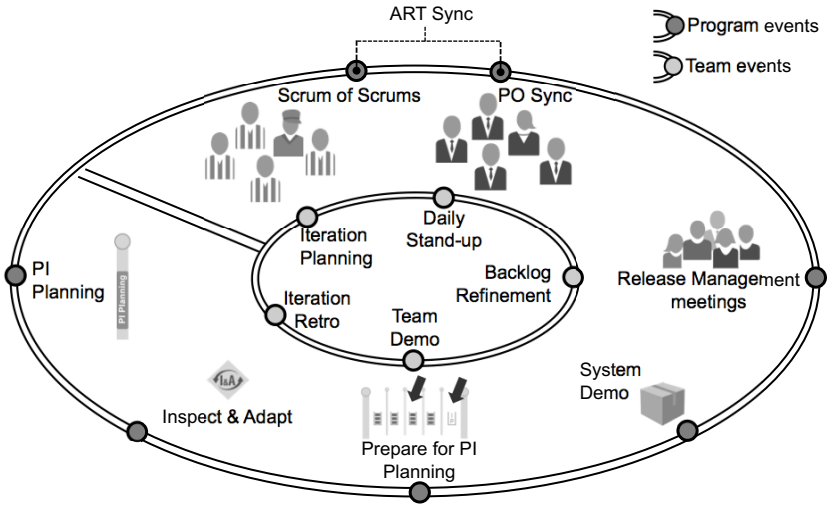
A commitment with two parts:

1. Teams agree to do everything in their power to meet the agreed-to objectives
2. In the event that fact patterns dictate that it is simply not achievable, teams agree to escalate immediately so that corrective action can be taken

5.2 Integrate and demo together

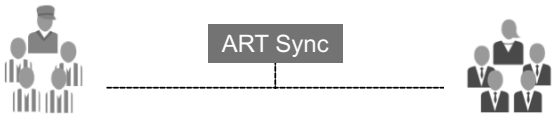
Program execution

Program events create a closed-loop system to keep the train on the tracks.



ART Sync

Programs coordinate dependencies through sync meetings.



Scrum of Scrums

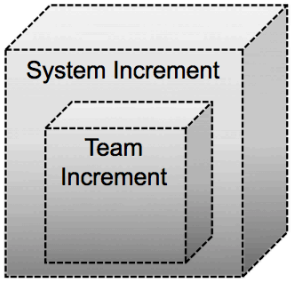
- Visibility into progress and impediments
- Facilitated by RTE
- Participants: Scrum Masters, other select team members, SMEs if necessary
- Weekly or more frequently, 30 – 60 minutes
- Timeboxed, and followed by a "meet after"

PO Sync

- Visibility into progress, scope, and priority adjustments
- Facilitated by RTE or PM
- Participants: PMs, POs, other stakeholders, and SMEs as necessary
- Weekly or more frequently, 30 – 60 minutes
- Timeboxed, and followed by a "meet after"

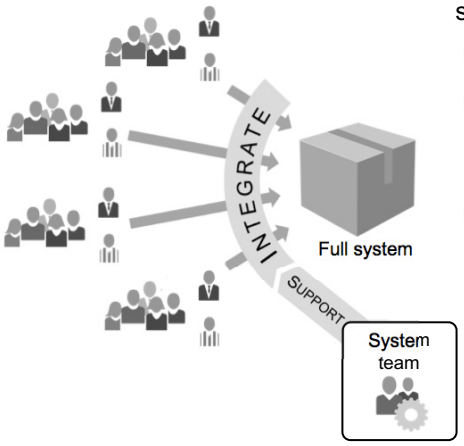
New system increment every two weeks

Every two weeks, teams evaluate the status of the new, integrated system increment.



- Features are functionally complete or "toggled" so as not to disrupt demonstrable functionality
- New Features work together, and with existing functionality
- Architectural Runway work in process is scaffolded and toggled
- System is continually verified via Story and Feature acceptance tests
- All practical NFR testing is done continuously

System Demo every two weeks



Demonstrate the full Solution increment to stakeholders every Iteration

- An integrated Solution demo
- Happens after the teams' demo (may lag by as much as one Iteration, maximum)
- Demo from the staging environment, or the nearest proxy

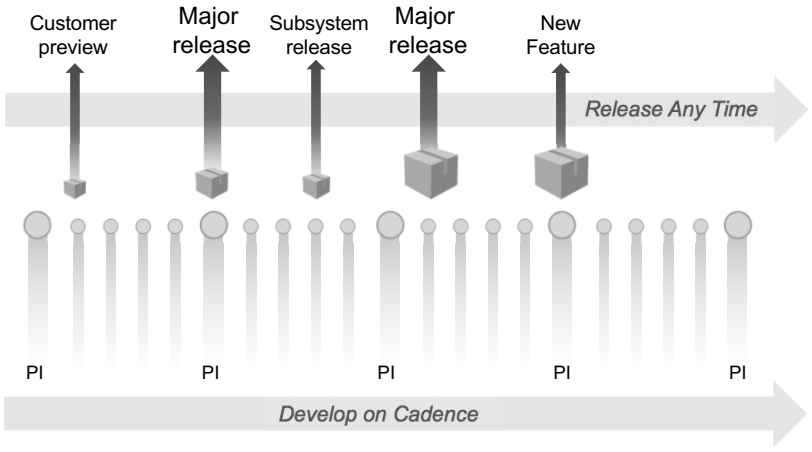
Exercise: System Demo challenges

- What are the challenges to having a new system increment every two weeks?
- Think about various aspects:
  - Environment
  - Culture
  - Tools
  - People

- In your group, prepare a list of 3 – 5 items that would make it hard to implement a System Demo of the integrated system every two weeks
- Be ready to present

Develop on Cadence. Release Any Time.

Development cadence limits variability to a single PI interval. Release is on demand.



Build a deployment pipeline

Real value occurs only when the end users are successfully operating the Solution.

DevOps

Deployment pipeline streamlines delivery:

1. Staging environment emulates production
2. Development and test environments match production to the extent feasible
3. Working system is deployed to staging every Iteration
4. Supporting activities:
   - Everything under version control
   - Ability to automatically build environments
   - Automated the actual deployment process

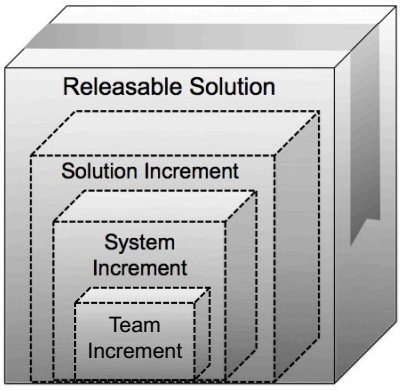
Releasing includes additional activities

System validation:

- User acceptance testing
- Final NFR testing
- Integration testing with other systems
- Regulatory standards and requirements

Documentation:

- Release communications
- End user documentation
- Bill of materials
- Training support personnel
- Installation/deployment instructions
- Legal, regulatory, other approvals
- etc. …



5.3 Learn together

5.29

Innovation and Planning Iteration

Facilitate reliability, Program Increment readiness, planning, and innovation

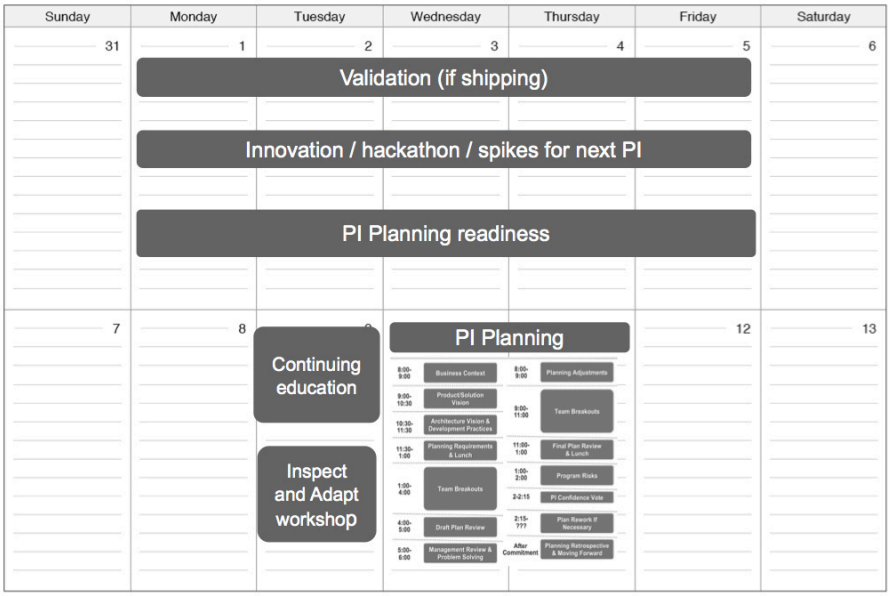
- Innovation: Opportunity for innovation spikes, hackathons, and infrastructure improvements
- Planning: Provides for cadence-based planning
- Estimating guard band for cadence-based delivery

Provide sufficient capacity margin to enable cadence.

—Don Reinertsen, Principles of Product Development Flow

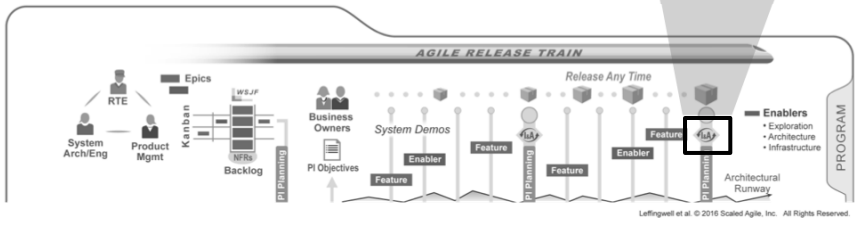
5.30

IP Iteration calendar



Inspect and Adapt

- Three parts:
  1. The PI System Demo
  2. Quantitative measurement
  3. The problem-solving workshop
- Attendees: Teams and stakeholders
- Timebox: 3 – 4 hours per PI



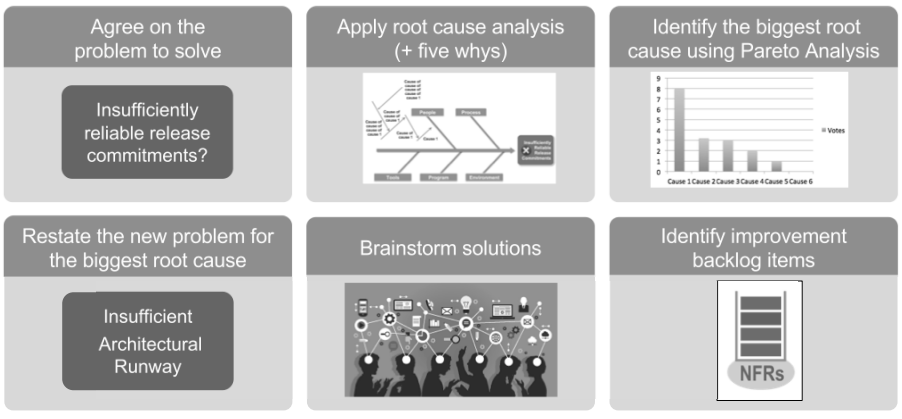
PI System Demo

At the end of the PI, teams demonstrate the current state of the Solution to the appropriate stakeholders.

- Often led by Product Management, POs, and the System Team
- Attended by Business Owners, program stakeholders, Product Management, RTE, Scrum Masters, and teams

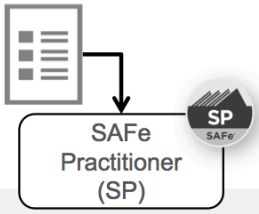
The problem-solving workshop

Teams conduct a short retrospective, then systematically address the larger impediments that are limiting velocity.



Now, let's plan our PI …

… and take the exam.

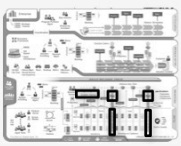


- SAFe Practitioner (SP) certification mark
- Access to new workbook releases
- Optional SP directory listing

Lesson summary

In this lesson, you:

- Learned how to plan and execute a Program Increment as a train
- Identified potential dependencies between teams
- Discussed the importance and challenges of the system demo
- Explored how to improve as a team of teams in the Inspect and Adapt workshop



Suggested Scaled Agile Framework reading:

- *"Program Increment" article*
- *"System Demo" article*
- *"Inspect and Adapt" article*

Appendix A

Writing and Splitting Stories

A.1

A.1 INVEST in a Good Story

A.2

INVEST in a good Story

- **I** Independent
- **N** Negotiable
- **V** Valuable
- **E** Estimable
- **S** Small
- **T** Testable

Stories are Independent

- Write closed Stories
- Slice through the architecture (vertical)
- Write only the delta (the change)
- Remove non-value dependencies (both technical and functional)

Non-value dependencies

As an administrator, I can set the consumer's password security rules so that users are required to create and retain secure passwords, keeping the system secure.

As a consumer, I am required to follow the password security rules set by the administrator so that I can maintain high security for my account.

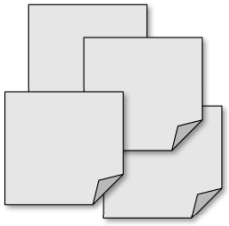
Split in different manner: *setup* and *enforcement* in each Story

As an administrator, I can set the password expiration period so that users are forced to change their passwords periodically.

As an administrator, I can set the password strength characteristics so that users are required to create passwords that are difficult to hack.

Stories are **N**egotiable

- User stories are statements of *intent*, not contracts or detailed requirements
- Too much detail gives impression of false precision or completeness
- Flexibility drives release schedule and goals



Stories are **V**alued by users

- Write Stories in the voice of the Customer
- Write for one user

"Technical" voice and value

Refactor the error logging system

User voice and value

As a consumer, I can receive a consistent and clear error message anywhere in the product so that I know how to address the issue.

As a technical support member, I want the user to receive a consistent and clear message anywhere in the application so they can fix the issue without calling support.

Stories are **E**stimable

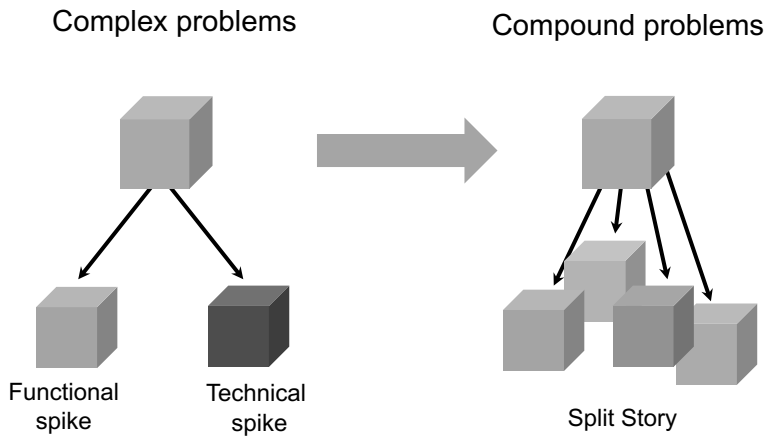
User stories are for planning and tracking

- To measure release progress, each Story needs an estimate of size

Estimating may be difficult because …

- Developers lack the domain knowledge to know what is to be done
- Developers lack the technical knowledge to know how to do something
- The Story is too big or too vague

Stories are **S**mall enough to fit in iterations



Stories are Testable

- Write Stories that are testable
- Include acceptance criteria for each Story

Not testable

As a power generation company salesperson, I want my search results to return quickly so that I can find relevant contacts for the information I am searching.

Testable

As a power generation company salesperson, I want to receive the first page of search results within 3 seconds so that I can find relevant contacts quickly.

A.2 Splitting Features and Stories

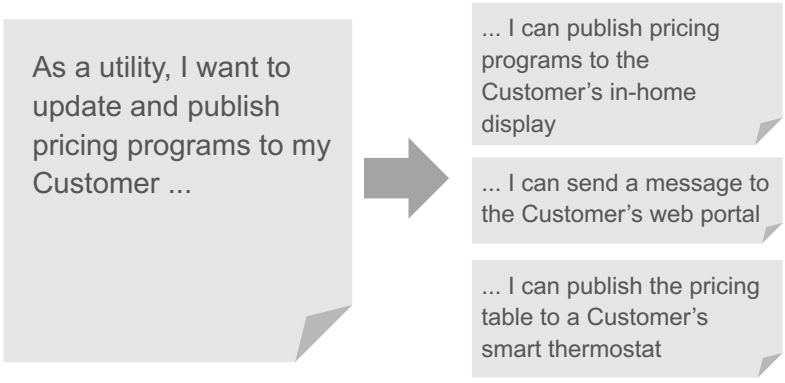
Splitting Features and Stories

Techniques for splitting Features and Stories to fit within their boundaries (PI and Iteration respectively)

1. Work flow steps
2. Business rule variations
3. Major effort
4. Simple/complex
5. Variations in data
6. Data methods
7. Defer system qualities
8. Operations
9. Use-case scenarios
10. Break out a spike

1. Split by work flow steps

Identify specific steps that a user takes to accomplish a work flow, then implement the work flow in increments.



2. Split by business rule variations

Business rule variations often provide a straightforward splitting scheme.

As a utility, I can sort Customers by different demographics ...

... sort by zip code

... sort by home demographics

... sort by energy consumption

3. Split by major effort

Split into several parts, with the first requiring the most effort. More functionality can be added later on.

As a user, I want to be able to select/change my pricing program with my utility through my web portal ...

... I want to use time-of-use pricing

... I want to prepay for my energy

… I want to enroll in critical peak pricing

4. Split by simple/complex

Simplify! What's the simplest version that can possibly work?

As a user, I basically want a fixed price, but I also want to be notified of critical peak pricing events ...

... respond to the time and the duration of the critical peak pricing event

... respond to emergency events

5. Split by variations in data

Variations in data provide additional opportunities, such as those shown in this localization example.

As a utility, I can send messages to Customers ...

Customers who want their messages in Spanish

Customers who want their messages in Arabic

Customers who want their messages in … etc.

6. Split by data methods

Complexity can be in the interface rather than the functionality itself. Split these Stories to build the simplest interface first.

As a user, I can view my energy consumption in various graphs ...

... using bar charts that compare weekly consumption

... in a comparison chart, so I can compare my usage to those who have the same or similar household demographics

7. Split by deferring system qualities

Sometimes functionality isn't that difficult. More effort may be required to make it faster ... or more precise ... or more scalable.

As a user, I want to see real-time consumption from my meter ...

... interpolate data from the last known reading

... display real-time data from the meter

8. Split by operations

Split by type of operation: **C**reate **R**ead **U**pdate **D**elete (CRUD)

As a user, I can manage my account ...

... I can sign up for an account

... I can edit my account settings

... I can cancel my account

… I can add more devices to my account

9. Split by use case scenarios

If use cases are used to represent complex interaction, the Story can be split via the individual scenarios.

As a user, I can enroll in the energy savings program through a retail distributor...

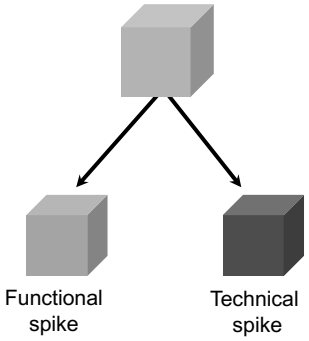
Use Case/Story #1 (happy path): Notify utility that consumer has equipment

Use Case/Story #2: Utility provisions equipment and data, notifies consumer

Use Case/Story #3 (alternate scenario): Handle data validation errors

10. Break out a spike

- A Story or Feature may not be understood well enough to estimate. Build a technical or functional spike to figure it out, then split the Story based on that result.
- Sometimes the team needs to develop a design, or prototype an idea
- Spikes are demonstrable, like any other Story



SCALED AGILE

SAFe 4.0 Scrum Master

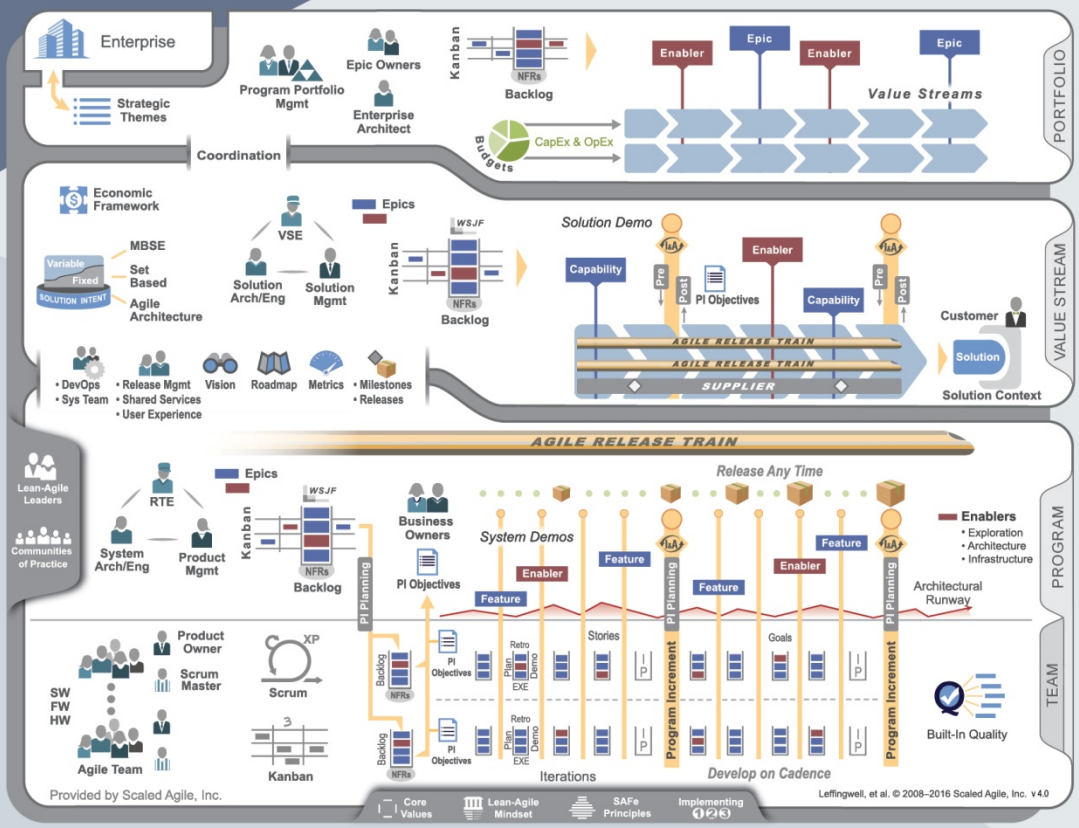
Scrum Roles and Responsibilities Card Game

Attend daily stand up meeting	Create user stories
Prioritize the team backlog	Facilitate team events
Attend the retrospective	Communicate with other teams
Test the system	Improve the team processes
Demo the system	Ensure quality
Develop the system	Represent the customer
Accept requirements	Help remove impediments
Coach the Agile Team	Plan the iteration
Attend backlog refinement	Attend Scrum of Scrums meetings

SAFe® 4.0 Glossary

Scaled Agile Framework® Terms and Definitions

ENGLISH



VERSION 4.0.5

PROVIDED BY

SAFe® | SCALED AGILE®

www.scaledagileframework.com | www.scaledagile.com

Agile Architecture
Agile architecture is a set of values and practices that advances the design and architecture of a system while implementing new business functions.

Agile Release Train
The Agile Release Train (ART) is a long-lived team-of-Agile-teams, which along with other stakeholders, develops and delivers solutions incrementally, using a series of fixed iterations within a program increment timebox. The ART aligns teams to a common business and technology mission.

Agile Teams
Agile teams are a group of three to nine dedicated individual contributors, covering all the roles necessary to define, build, and test a quality increment of value in an iteration.

Architectural Runway
Architectural runway consists of the existing code, components and technical infrastructure necessary to support implementation of high priority, near-term features, without excessive delay and redesign.

Budgets
See Lean-Agile budgeting.

Built-in Quality
Built-in quality practices ensure that each solution element, at every increment, meets appropriate quality standards throughout development.

Business Epic
See epic.

Business Owners
Business Owners are a small group of stakeholders who have the primary technical, fitness for use, governance and Return on Investment (ROI) responsibility for a solution developed by an Agile Release Train (ART). They are key stakeholders on the ART and actively participate in certain events.

Capability
A capability is a high-level solution behavior that typically spans multiple Agile Release Trains (ARTs). They are sized and split as necessary to fit within a single program increment.

CapEx and OpEx
Capital Expenses (CapEx) and Operating Expenses (OpEx) describe Lean-Agile financial practices for tracking capital expenses (CapEx), and operating expenses (OpEx) in a value stream budget. In some cases, CapEx may include capitalized labor associated with development of intangible assets such as software, intellectual property and patents.

Communities of Practice
Communities of Practice (CoPs) are groups of people who have a common interest in a specific technical or business domain. They collaborate regularly to share information, improve their skills and performance and advance the general knowledge of the domain.

Continuous Integration
Continuous integration is a built-in quality practice, where team members constantly integrate and verify their work, using automated build-and-test environments that quickly identify problems and defects.

Coordination
See value stream coordination.

Core Values
Core values define the ideals and beliefs that are key to SAFe's effectiveness. They are: alignment, built-in quality, transparency, and program execution.

Customer
The customer is anyone who consumes the work of a value stream. Customers are an integral part of the agile development process and value stream.

Develop on Cadence
Develop on cadence is a strategy for managing the inherent variability in solution development, by making sure important events and activities on a regular, predictable schedule.

DevOps
DevOps is a mindset, culture, and set of technical practices that foster communication, collaboration, and close cooperation among all the professionals needed to develop, test, deploy, and maintain a solution.

Economic Framework
The economic framework is a set of decision rules aligning everyone to the financial objectives of the mission, defines economic tradeoff parameters, and assures operation within the budget provided by program portfolio management.

Enablers
Enablers further the exploration, infrastructure, and architecture development activities needed to support future business functionality. Enablers arise from the backlog and occur at all levels of the framework where they are described as enabler portfolio epics, enabler capabilities, enabler features, or enabler stories.

Enterprise
The enterprise represents the business entity that has the ultimate strategic, fiduciary, and governance authority for all the development value streams that make up a portfolio.

Enterprise Architect
The Enterprise Architect fosters adaptive design and engineering practices, and drives strategic architectural initiatives for a portfolio. Enterprise Architects facilitate the reuse of ideas, components, and proven patterns across solutions.

Epic
An epic is a significant initiative that typically affects multiple value streams and ARTs. Epics require analysis using a lightweight business case and financial approval before implementation. There are two types of epics: business epics and enabler epics, which may occur at the portfolio, value stream, and program level.

Epic Owners
Epic Owners are responsible for coordinating portfolio epics through the portfolio kanban system. They develop the business case and, when approved, work directly with the key stakeholders on the selected Agile Release Trains (ARTs) to help realize the implementation.

Feature
A feature is a service provided by the system that fulfills stakeholder needs and can be delivered by a single ART. Each feature includes a statement of benefits and acceptance criteria, and is sized to fit within a program increment.

Implementing 1-2-3
Implementing 1-2-3 is a proven success pattern for launching SAFe. It describes the three basic steps: 1) train Lean-Agile change agents, 2) train executives, managers, and leaders, 3) train teams and launch Agile Release Trains (ARTs).

Innovation and Planning Iteration
The Innovation and Planning iteration occurs every PI and serves multiple purposes. It acts as an estimating buffer for meeting PI objectives, as well as providing dedicated time for innovation, continuing education, and PI planning and Inspect and Adapt (I&A) events.

Inspect and Adapt
Inspect and Adapt (I&A) is a significant event, held at the end of each Program Increment (PI) where the current state of the solution is demonstrated and evaluated. Teams then reflect, and identify improvement backlog items via a structured, problem-solving workshop.

Iteration Execution
Iteration execution is how Agile teams manage their work throughout the iteration timebox, resulting in a high-quality, working, tested system increment. Each iteration follows a standard pattern: plan the iteration, commit to a goal, execute, demonstrate the work to the key stakeholders, and hold a retrospective.

Iteration Goals
Iteration goals are high-level summaries of the business and technical goals that the team and Product Owner agree to accomplish in an iteration. They serve as a communication mechanism within the team, and to the team's stakeholders, and help ensure alignment with the PI objectives.

Iteration Planning
Iteration planning is an event at which all team members determine how much of the team backlog they can commit to deliver during an upcoming iteration. The team summarizes the work as a set of committed iteration goals.

Iteration Retrospective
The iteration retrospective is a meeting where the team members discuss the results of the iteration, review their practices and identify ways to improve.

Iterations
Iterations are a standard, fixed-length timebox during which teams deliver incremental value in the form of working, tested software and systems. Iteration lengths may be chosen from one to three weeks, with two weeks being the suggested, and most common, duration.

Lean-Agile Budgeting
Lean-Agile budgeting is a set of practices that fund and empower value streams, while maintaining financial and fitness-for-use governance through objective evaluation of working systems, and dynamic budget adjustments.

Lean-Agile Leaders
Lean-Agile leaders are lifelong learners who are responsible for the successful adoption of SAFe and the results it delivers. They help teams build better systems by learning, exhibiting, teaching and coaching SAFe's Lean-Agile principles and practices.

Lean-Agile Mindset
A Lean-Agile mindset combines the concepts of the Agile Manifesto and Lean thinking, serving as the basis for SAFe principles and practices.

Metrics
Metrics are agreed-upon measures used to evaluate how well the organization is adopting SAFe and progressing toward the portfolio, value stream, program and team level business and technical objectives.

Milestones
Milestones are used to track progress toward a specific goal or event. There are three types of SAFe milestones: fixed-date milestones, program increment milestones and learning milestones.

Model-Based Systems Engineering
Model Based Systems Engineering (MBSE) is a methodology that focuses on the development of various related system models that are used to define and design the system. Models provide an efficient way to explore, update and communicate system aspects to stakeholders while significantly reducing or eliminating dependence on traditional documents.

Nonfunctional Requirements
Non-functional requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs.

Program Increment Planning
Program Increment (PI) planning is a cadence-based, face-to-face planning event that serves as the heartbeat of the Agile Release Train (ART), aligning all the teams on the ART to a common goal.

Portfolio Backlog
The portfolio backlog is a prioritized list of epics that have been approved for implementation through the portfolio kanban system.

Portfolio Business Epic
Portfolio business epics are large business initiatives that often span value streams and require multiple program increments to deliver.

Portfolio Kanban
The portfolio kanban is a method used to visualize and manage the analysis, prioritization and flow of portfolio epics from ideation to implementation and completion.

Portfolio Level
The portfolio level contains the roles, artifacts and processes needed to meet the strategic intent of the portfolio. It defines the value streams and funding for the people and other resources that build the solutions.

Pre- and Post-PI Planning
Pre- and post-PI (Program Increment) planning meetings are used to prepare for, and follow-up, PI planning for multiple Agile Release Trains (ARTs) and suppliers in a large value stream.

Product Management
Product Management is the content authority for the program level and backlog. They are responsible for identifying customer needs, prioritizing features and developing the program vision and roadmap.

Product Owner
The Product Owner is the content authority for the team level. They are responsible for the team backlog, prioritizing and accepting stories, and representing the customer to the Agile team.

Program Backlog
The program backlog is a prioritized list of features intended to address user needs and deliver business benefits. It also includes the enabler features necessary to build the architectural runway.

Program Epics
Program epics are initiatives significant enough to require analysis using a lightweight business case and financial approval before implementation. Their scope is limited to a single Agile Release Train (ART) and may take several Program Increments to develop.

Program Increment
A Program Increment (PI) is a timebox in which an Agile Release Train (ART) delivers incremental value in the form of working, tested software and systems. PIs are typically eight to twelve weeks long, and the most common pattern for a PI is four development iterations, followed by one Innovation and Planning (IP) iteration.

Program Kanban
The program kanban is a method used to visualize and manage the analysis, prioritization and flow of program epics and features from ideation to completion for a single Agile Release Train (ART).

Program Level
The program level contains the roles and activities needed to continuously deliver solutions via an Agile Release Train (ART).

Program Increment Objectives
Program Increment (PI) objectives are an integrated summary of all the teams' PI objectives for an Agile Release Train (ART). They are used to communicate the plan to stakeholders and to measure accomplishments of the ART for a program increment.

Program Portfolio Management
Program Portfolio Management (PPM) is a function that contains the individuals who have the ultimate decision-making authority of a portfolio. It is responsible for strategy, investment funding, common program management elements, and portfolio governance.

Release
A release is the private or public delivery of valuable, working, and fully tested and validated solution increments.

Release Any Time
Release any time is a practice whereby the Agile Release Train (ART) separates the development cadence from the solution release cycle. Within appropriate technical and business governance, ARTs may release whatever is needed, at any time.

Release Management
Release Management is a function that communicates the status of the release to stakeholders, validates that the solution meets the relevant quality and governance criteria, and provides the final authorization for the release.

Release Train Engineer
The Release Train Engineer (RTE) is a servant leader and coach for the Agile Release Train (ART), who facilitates the ART's processes, events, and execution. He or she escalates impediments and helps manage risk, value delivery, and continuous improvement.

Roadmap
The roadmap is a schedule of events and milestones that communicate planned deliverables over a timeline. It includes commitments for the planned PI and offers visibility into the forecasted deliverables of the next few PIs.

SAFe Principles
SAFe principles define the nine core beliefs, fundamental truths and economic premises that drive effective roles, and practices and successful SAFe implementation.

Scrum Master
Scrum Masters are servant leaders and coaches for an Agile team. They help educate the team in Scrum, extreme programming, Kanban and SAFe, and ensure that the process is being followed. They also help remove impediments, and foster an environment for high-performing team dynamics, continuous flow, and relentless improvement.

ScrumXP
ScrumXP is a lightweight process for cross-functional, self-organized teams to deliver value within the context of SAFe. ScrumXP combines the power of Scrum project management practices with extreme programming inspired technical practices.

Set-Based Design
Set-based design is a practice in which the requirements and design options are kept flexible for a longer period of time. Instead of choosing a single "point" solution upfront, set-based design is used to identify design options, and eliminate poorer choices over time, thereby enabling flexibility in the design process.

Shared Services
Shared Services represents the specialty roles, people and services necessary for the success of an Agile Release Train (ART) or value stream, but can't be dedicated to any specific ART full-time (e.g. security specialists, database administrators).

Solution
A solution is a final product, service, or system delivered to the customer or, enables an operational value stream within the organization.

Solution Architect/Engineering
Solution Architect/Engineering represents the individuals and teams who have the technical responsibility for the overall architecture and engineering design of the solution. They help align the value stream and the Agile Release Train (ART) to a common technological and architectural vision.

Solution Context
Solution context identifies critical aspects of the environment for the target solution. It identifies impact on the requirements, usage, installation, operation, and support of the solution itself.

Solution Demo
The solution demo is where the results of all the development efforts from multiple Agile Release Trains (ARTs)—along with the contributions from suppliers—are integrated, evaluated, and made visible to customers and other stakeholders.

Solution Intent
Solution intent represents the repository for storing, managing, and communicating knowledge of the current and intended solution, including traceability between the items where required.

Solution Management
Solution Management is the content authority for the value stream level. They work with customers to understand their needs, create the vision and roadmap, define requirements, and guide work through the value stream kanban.

Spanning Palette
The spanning palette contains various roles and artifacts that may be applicable at any level of the framework. It is used to apply various SAFe elements to a specific team, program, value stream, or portfolio level.

Stories
Stories are short, simple descriptions of a small piece of desired functionality, written in the user's language. Each story supports incremental development by implementing a small, vertical slice of system functionality.

Strategic Themes
Strategic themes are itemized, differentiated business objectives that connect a portfolio to the business strategy of the enterprise. They provide business context for decision-making and they serve as inputs to the vision, budget and backlogs for the portfolio, value stream, and program levels.

Supplier
A supplier is an organization that develops and delivers components and subsystems that help value streams deliver value to their customers.

System Architect/Engineering
System Architect/Engineering represents an individual or small team that defines a common technical and architectural vision for the solution under development. They participate in defining the system and subsystems, interfaces, validate technology assumptions, and evaluate alternatives.

System Demo
The system demo occurs at the end of every iteration, and provides an integrated view of the new features, which have been delivered by all the teams in the Agile Release Train (ART) for the most recent iteration. It provides the ART with an objective measure of progress during a program increment.

System Team
The System Team is a special Agile team that provides assistance in building and using the Agile development environment, including continuous integration and test automation. The System Team integrates assets from Agile teams, performs end-to-end solution testing where necessary, and assisting with deployment. They often facilitate the system demo.

Team Backlog
The team backlog contains user and enabler stories that originate from the program backlog, as well as stories that arise locally from the team's specific context. It represents all the things a team needs to do to advance their portion of the system.

Team Demo
The team demo is used to measure progress and get feedback at the end of each iteration by demonstrating every story, spike, refactor, and new Non-Functional Requirement (NFR) developed in the recent iteration.

Team Kanban
Team kanban is a method that facilitates the flow of value by visualizing Work in Process (WIP) and establishing WIP limits, measuring throughput, and continuously improving the process.

Team Level
The team level contains the roles, activities, and process model for the teams who power the Agile Release Train (ART).

Team PI Objectives
Team PI objectives describe the business and technical goals that an Agile team intends to achieve in the upcoming Program Increment (PI). They summarize and validate business and technical intent, which enhances communication, alignment, and visibility.

User Experience
User experience designers support a consistent user experience across the components and systems of the larger solution, while Agile teams have responsibility for implementing the solution, including the user-facing elements.

Value Stream Backlog
The value stream backlog is the repository for all upcoming capabilities and enablers, each of which can span multiple Agile Release Trains (ARTs) and is used to advance the solution and build the architectural runway.

Value Stream Coordination
Value stream coordination provides guidance for managing dependencies across value streams in a portfolio.

Value Stream Engineer
The Value Stream Engineer (VSE) is a servant leader and coach for a value stream. VSEs facilitate value stream processes, events and execution, as well as driving continuous improvement.

Value Stream Epics
Value stream epics are significant initiatives that require analysis, using a lightweight business case, and financial approval before implementation. Their scope is limited to a to a single value stream and may take several Program Increments (PIs) to develop.

Value Stream Kanban
The value stream kanban is a method used to manage the analysis, prioritization and flow of value stream epics and capabilities, from ideation to completion.

Value Stream Level
The value stream level provides the roles, responsibilities and activities necessary to support those building large and complex solutions, which typically require multiple Agile Release Trains (ARTs), as well as the contributions of suppliers. This optional level is used by organizations that face the largest systems challenges, which require multidisciplinary software and system professionals.

Value Stream PI Objectives
Value stream PI objectives are the business and technical goals identified by value stream stakeholders for a Program Increment (PI). They communicate to stakeholders what the value stream will deliver in the upcoming PI. They are used for large value streams that have multiple Agile Release Trains (ARTs) or suppliers.

Value Streams
Value streams represent the series of steps that an organization uses to build solutions that provide a continuous flow of value to a customer. Value streams are the primary means for understanding business objectives, organizing teams and Agile Release Trains (ARTs), and delivering end value. They are realized by an ART.

Vision

The vision is a description, or future view of the solution to be developed, reflecting customer and stakeholder needs, as well as the proposed features and capabilities. It provides the larger, contextual overview and purpose of the solution under development.

Weighted Shortest Job First

Weighted Shortest Job First (WSJF) is a prioritization model used to sequence “jobs” (e.g., features, capabilities, and epics) so as to produce maximum economic benefit in a flow-based system. WSJF is calculated as the cost of delay divided by job size.

Guide to acronyms and abbreviations

ART	Agile Release Train
BO	Business Owner
BV	Business Value
BVIR	Big Visual Information Radiator
CFD	Cumulative Flow Diagram
CapEx	Capital Expenses
CI	Continuous Integration
CoD	Cost of Delay
CoP	Community of Practice
DoD	Definition of Done
DSU	Daily Stand-up
EA	Enterprise Architect
EO	Epic Owner
FW	Firmware
HW	Hardware
I&A	Inspect and Adapt
IP	Innovation and Planning (iteration)
MBSE	Model-Based Systems Engineering
NFR	Non-functional Requirements
OE	Opportunity Enablement
OpEx	Operating Expenses
PDCA	Plan, Do, Check, Adjust
PI	Program Increment
PM	Product Manage
PM/PO	Product Manager / Product Owner
PO	Product Owner
PPM	Program Portfolio Management
ROAM	Resolved, Owned, Accepted, Mitigated
RR	Risk Reduction
RTE	Release Train Engineer
S4T	SAFe® for Teams
SAFe®	Scaled Agile Framework
SA	SAFe® Agilest
SM	Scrum Master
SMART	Specific, Measurable, Achievable, Realistic, Time-bound
SoS	Scrum of Scrums
SP	SAFe® Practitioner
SPC	SAFe® Program Consultant
SW	Software
UX	User Experience
VS	Value Stream
VSE	Value Stream Engineer
WIP	Work in Process
WSJF	Weighted Shortest Job First
XP	Extreme Programming