Lesson 5

Improving Flow with Kanban and XP

1. Exploring the Scrum Master Role in the SAFe Enterprise
2. Applying SAFe Principles: A Scrum Master's Perspective
3. Exploring Agile and Scrum Anti-Patterns
4. Facilitating Program Execution
5. Improving Flow with Kanban and XP
6. Building High-Performing Teams
7. Improving Program Performance with Inspect and Adapt

SAFe® Course Attending this course gives students access to the SAFe® Advanced Scrum Master exam and related preparation materials.

Learning objectives

5.1 Build your Kanban board

5.2 Measure and optimize flow

5.3 Build quality in

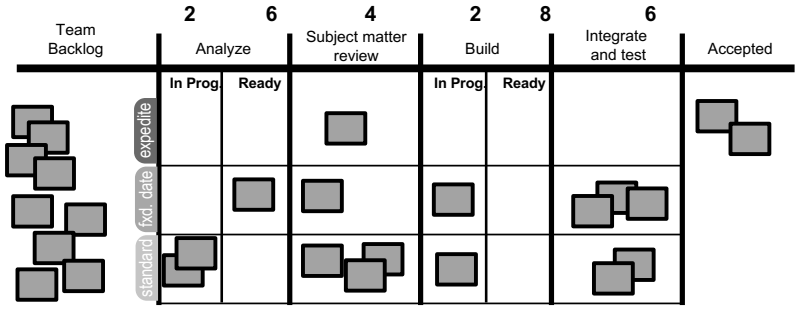
5.4 Foster engineering craftsmanship

5.5 Facilitate collaboration with Architects, System Team, and Operations

5.1 Build your Kanban board

Understanding Kanban

Kanban is a pull system that visualizes work flow and uses WIP limits at different work flow steps to facilitate the flow.



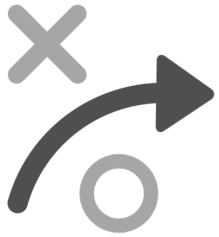
- Kanban relies on empirical data for continuous improvement
- Helps avoid overload by pulling work only when there is excess capacity at a step
- Helps the team reflect their unique work flow
- Makes process policies and decisions visible to the entire team

Overview of the process

1. Identify team workflow steps and interfaces with the rest of the program.
2. Arrange steps. Decide what steps:
   - should be added, removed, or inversed.
   - need to be merged or split.
   - need buffers before or after.
3. Assign initial WIP limits.

---

4. Start operating/adjust.



Step 1: Identify workflow steps/interfaces

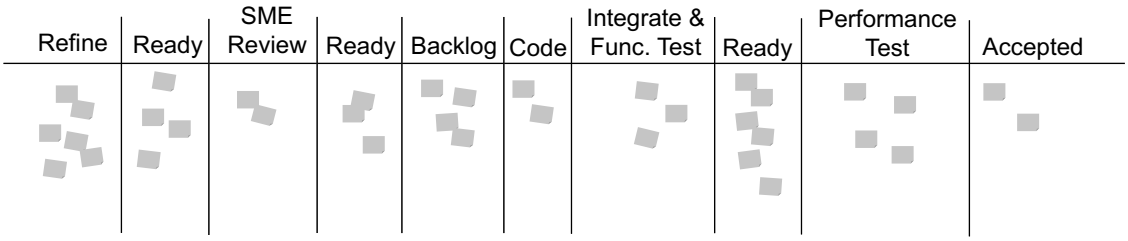
Define → Design → Get SME Approval → Code → Integrate → Test → Done

Step 2: Arrange steps

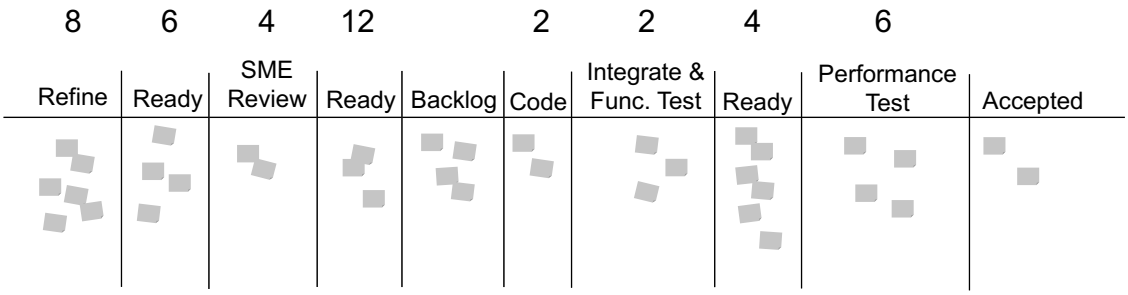
Decide what steps:

a. Should be added, removed, or inversed
b. Need to be merged or split
c. Need buffers before or after

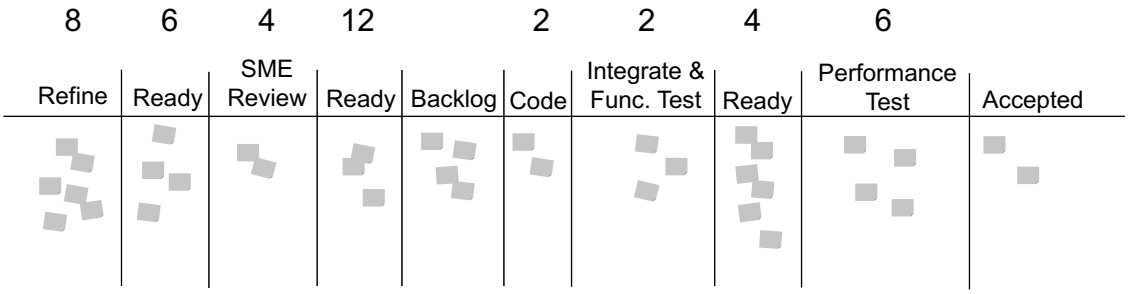
Define → Design → Get SME Approval → Code → Integrate → Test → Done

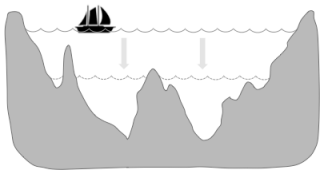


Step 3: Assign initial WIP limits



Step 4: Start operating/adjust

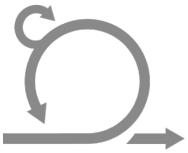


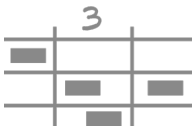


WIP limits must be sufficiently low to see bottlenecks in the process

Applicability

Kanban perfectly extends Scrum by providing granular pull mechanisms that drive more effective Iteration execution.

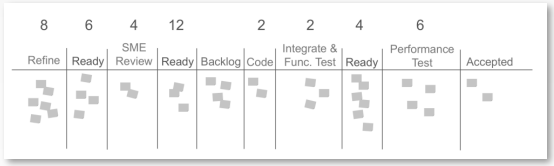




- Kanban connects capacity-based planning in Scrum with a throughput-based approach
- It helps improve Iteration outcomes
- It allows better visibility into the progress of work based on the team-specific work flow

Exercise: Build your own Kanban board

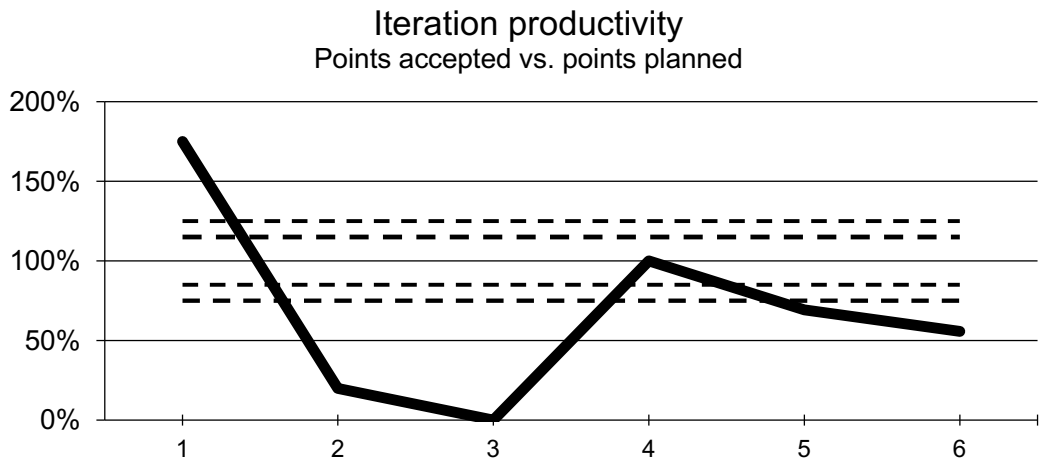
- In your group, select one person's context
- Together, build a team Kanban board using the steps described in the previous slides. Be sure to:
  - Understand the actual workflow
  - Decide how to map it to the board (steps, buffers, etc.)
  - Determine initial WIP limits



5.2 Measure and optimize flow

Winning the 'Iteration'

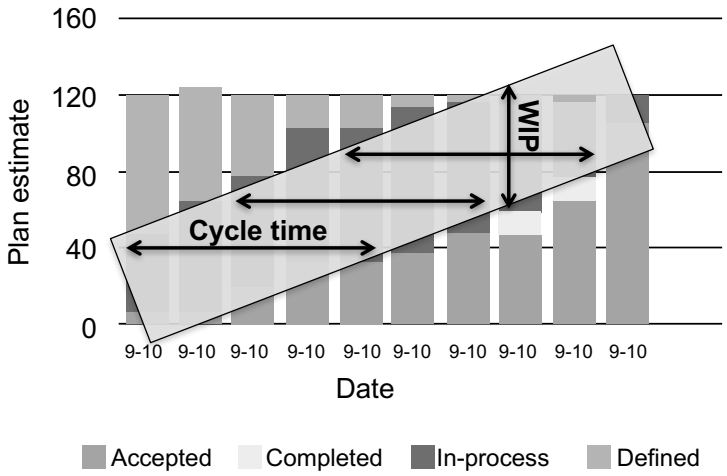
The buffet rule: "Take all you want, but eat all you take" – Jay Packlick



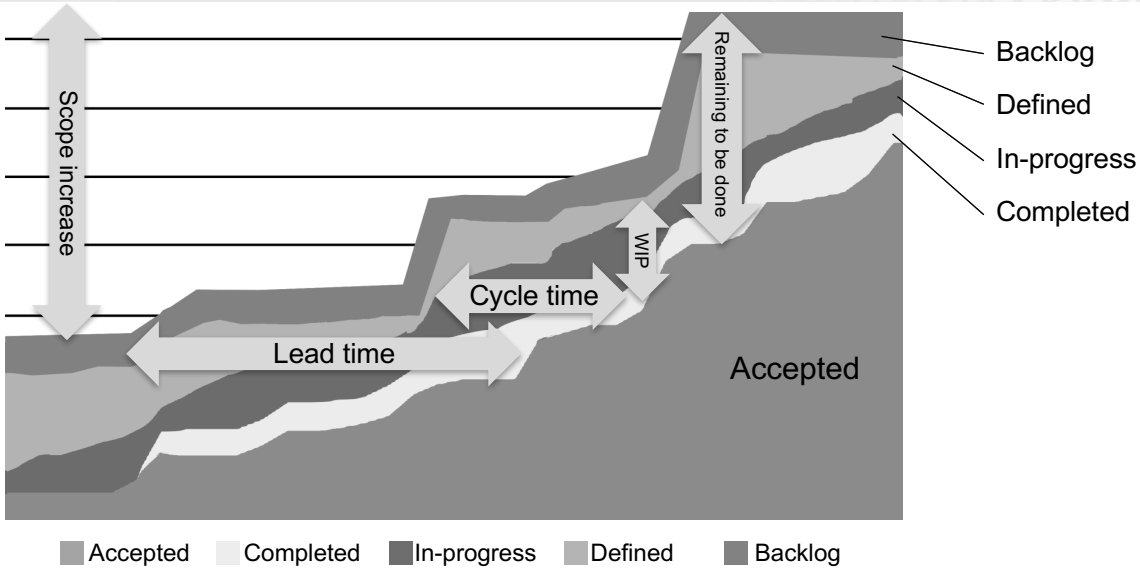
Understanding cumulative flow

Shows:

- Performance of team
- Rate of acceptance
- Points flow (not hours based)
- WIP (Work in Process)
- Average cycle times
- Opportunities to improve

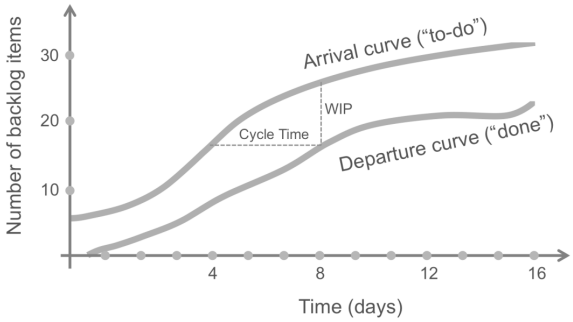


Putting it all together – Cumulative Flow Diagram (CFD)



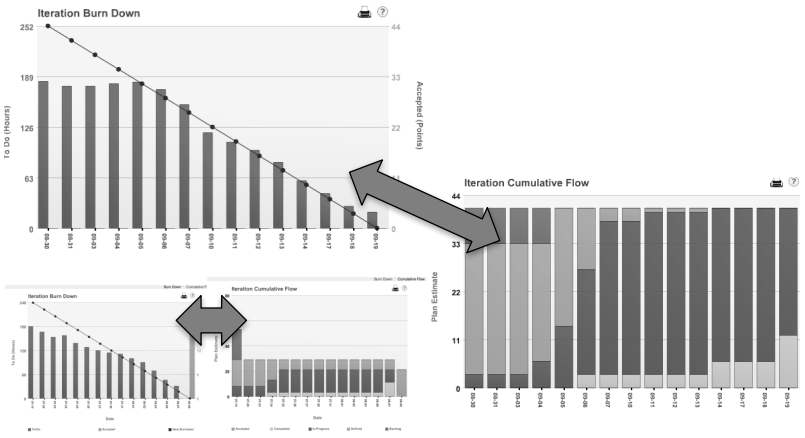
Typical measures in Kanban

- Average cycle time: The average time a backlog item spends in the system after it's been pulled from the backlog and before it is accepted
- Average WIP in the system: The average number of backlog items currently in progress (all items between 'backlog' and 'accepted')
- Throughput: The number of items a team can finish per unit of time



CFD shows problems burn-down charts hide

- Same Iterations – different views
- Burn-down looks (mostly) okay
- CFD provides early feedback
  - More WIP = longer lead times
  - Late completion
  - Late acceptance



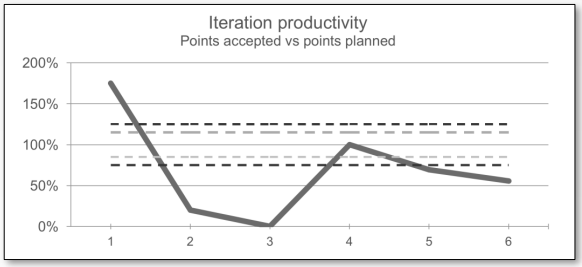
Exercise: Complementing Scrum metrics with Kanban

- How is a team's velocity connected to the Kanban measures of:
  - Average cycle time?
  - Average WIP?
  - Throughput?
- How are the measures above connected with the:
  - Average size of a Story?
  - Variance (spread) of Story sizes?

PREPARE 5 min | SHARE 2 min

Exercise: Example Iteration KPIs

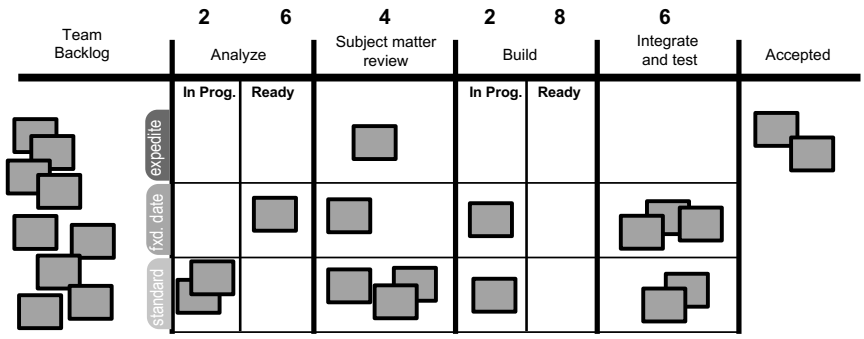
- What is this KPI saying?
- How could Scrum Masters improve flow and predictability?



PREPARE 5 min | SHARE 2 min

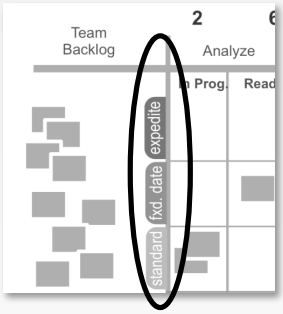
Classes of service to adjust flow

- Standard - Operate normally. Adhere to WIP limits.
- Fixed Date - Adhere to WIP limits. Must be pulled from the backlog early enough.
- Expedite - Can violate WIP limits. No more than one item at a time.



Exercise: Classes of service

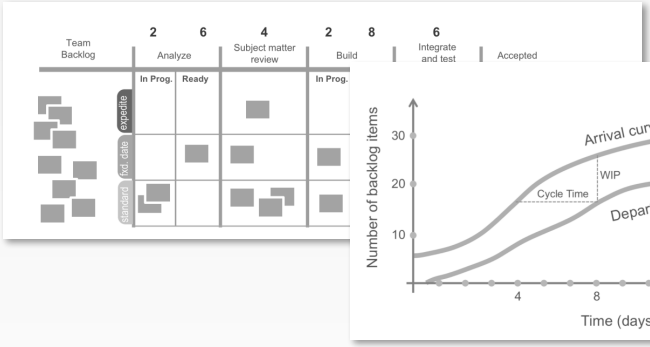
- Provide examples where the three classes of service would apply in your context
- What could be the potential sources of ‘fixed date’ and ‘expedite’ items?
- Be ready to present



PREPARE 5 min | SHARE 2 min

Exercise: Facilitate improvement

How would you leverage Kanban in your team’s retrospective?



PREPARE 5 min | SHARE 2 min

5.3 Build quality in

Built-In Quality

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

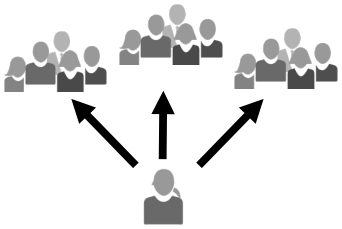
Building quality in:

- 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

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

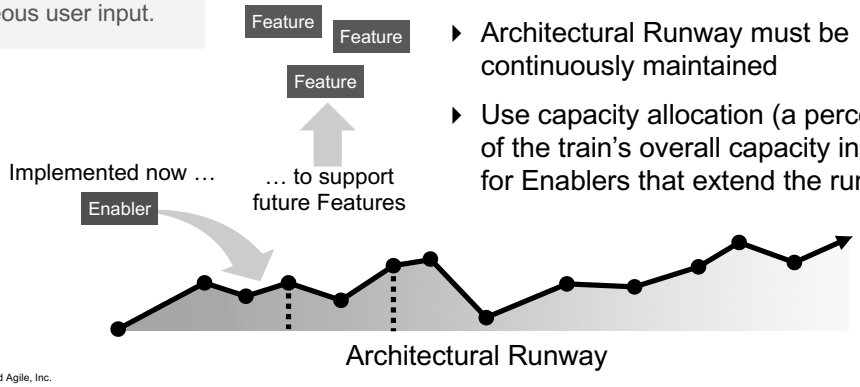


A balance between emergent design and intentional architecture is required for speed of development and maintainability.

Architectural Runway

Architectural Runway is 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.

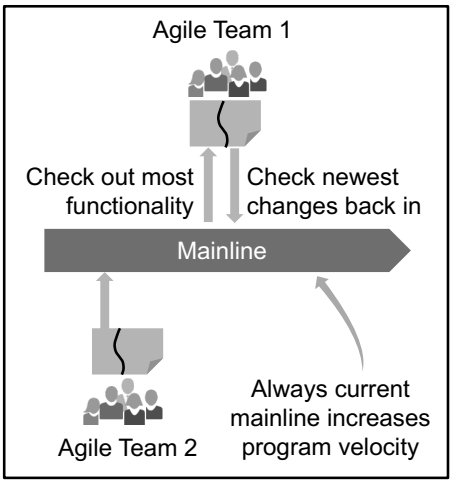


- Enablers build up the runway
- Features consume it
- Architectural Runway must be continuously maintained
- Use capacity allocation (a percentage of the train's overall capacity in a PI) for Enablers that extend the runway

Continuous system integration

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

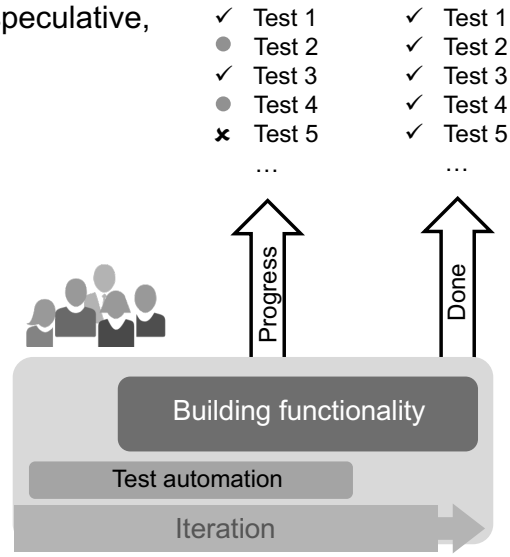
- Integrate every vertical slice of a User Story
- Avoid physical branching for software
- Frequently integrate hardware branches
- Use development by intention in case of inter-team dependencies
  - Define interfaces and integrate first, then add functionality



Test first: Automate now!

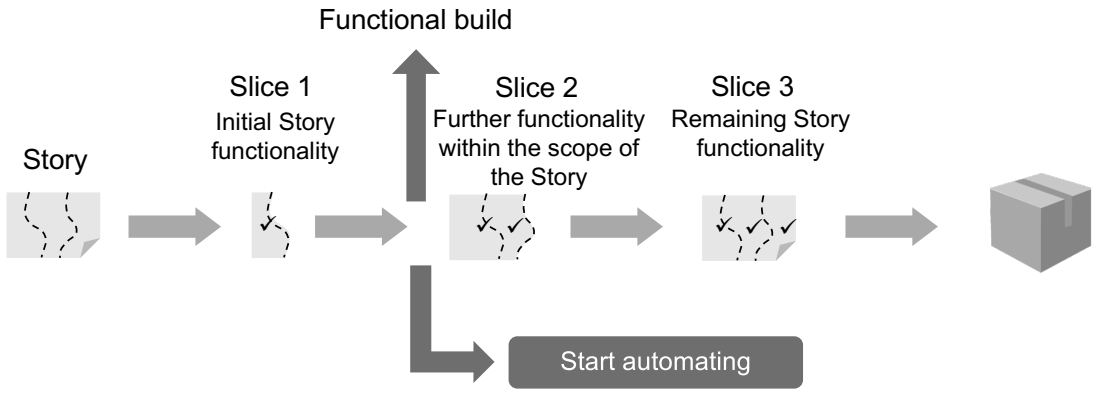
Otherwise, velocity is bottlenecked, quality is speculative, and scaling is impossible.

- 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 indicators



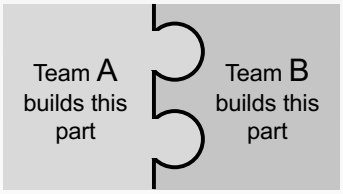
Transition to early automation

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



Exercise: Integration and program velocity (part 1)

- Each table team is assigned A or B.
- Each table team A is assigned a partner table team B, not right next to each other.
- Each team will build its own component, as shown in the picture. Use standard sticky notes and the scissors on your table to build your component of the whole.
- The partnered teams *cannot* communicate with each other.



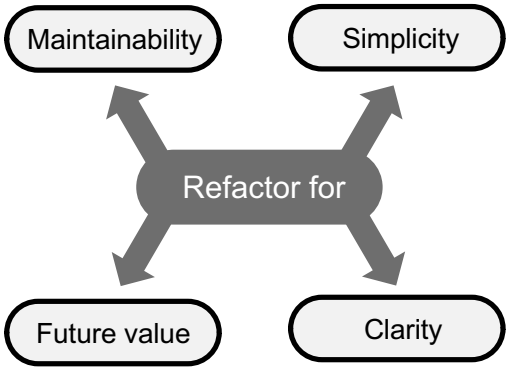
Exercise: Integration and program velocity (part 2)

- Once the teams are done building their components, put the components together for every pair of teams
- What do the results look like?
- Is any rework required?
- Based on what we've just seen, what impact does late cross-team integration have on the program's velocity?

Refactoring

Refactoring allows teams to maintain high velocity.

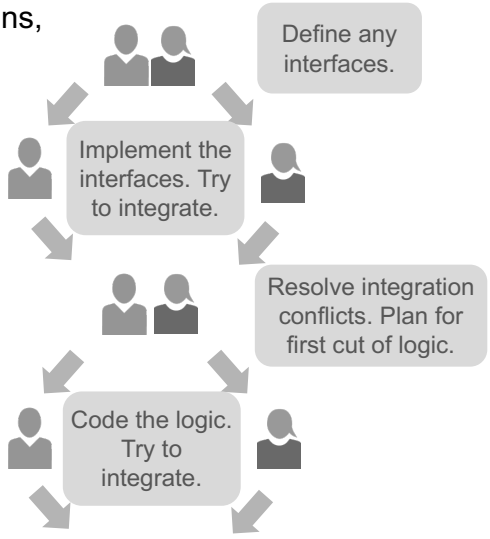
- It is impossible to predict requirements or design in detail
- Refactoring allows teams to quickly correct the course of action
- Emergent design is impossible without continuous refactoring
- Most Yser Stories will include some refactoring effort
- If technical debt is big – teams track and implement 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

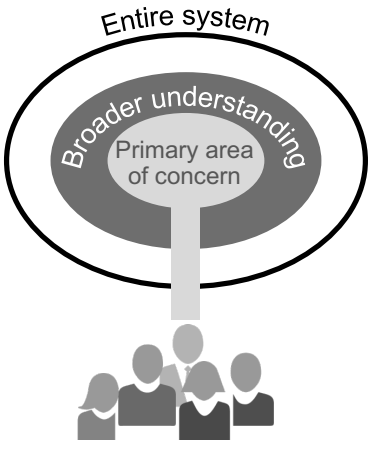


Example User Story implementation flow

Collective ownership

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

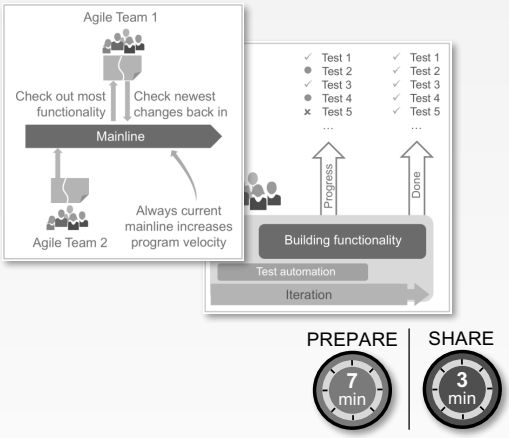
- Collective ownership fosters Feature orientation.
- Collective test ownership is even more important. It facilitates shared understanding of system behavior.
- Collective ownership is supported by:
  - Design simplicity
  - Communities of Practice
  - Pair work
  - Joint specification and design workshops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



Teams gain a broader understanding of the system.

Exercise: Facilitate the adoption of integration and testing

- Identify current problems in your team's experience with integration and test automation.
- Build a realistic plan for enhancing your team's integration and testing ability.
- How does that plan connect with the team's Definition of Done?



5.4 Foster engineering craftsmanship

Foster adoption of technical practices

A Scrum Master facilitates the adoption of technical practices.

- Helps the team mature in their Definition of Done
- Creates transparency and urgency around continuous system integration
- Encourages small, automated acceptance tests at the beginning; evolves from there
- Encourages team members to coach each other in TDD, refactoring, simple design
- Helps the team adopt a 'thinking backward' approach: What is the expected behavior of the functionality that we are about to code?
- Helps understand and use the power of human-readable acceptance tests
- Encourages pairing and peer review

Recommended reading

Scaledagileframework.com/built-in-quality

Rachael Davies and Liz Sedley, *Agile Coaching* (Chapter 9)

Ron Jeffries, Ann Anderson, and Chet Hendrickson, *Extreme Programming Installed*

Robert C. Martin, *Clean Code*

Ken Pugh, *Lean-Agile Acceptance Test-Driven Development*

Jez Humble and David Farley, *Continuous Delivery*

Encourage learning

Scrum Masters create an environment for continuous learning.

Team Inside-Outs	... short presentations to the team
Book and Coffee Breaks (BCBs)	... discussions of new topics over coffee
Coding 'Dojo'	... coding in front of a group
Communities of Practice	... self-organizing groups to build knowledge using the above formats

Learning is not compulsory … neither is survival.

—W. Edwards Deming

Encourage learning: Inside-Outs

Team Inside-Outs

A team member prepares a short presentation or flip-chart talk for their team.

- Frequency: Once per 1 – 2 Iterations
- Durations: 30 – 60 minutes
- Example: *We will soon start using Hibernate for data persistence. John has experience and is willing to share his knowledge.*

Your role

- Help kick-start the first 2 – 3 Inside-Outs and help participants prepare
- Maintain the Inside-Out schedule
- Invite shared resources (System Architect, User Experience, infrastructure, etc.) or people from other teams to discuss useful topics

Encourage learning: BCBs

Book and Coffee Break (BCB)

Normal coffee break joined by 3 – 4 people with a book on a new technology, practice, or domain topic that the team is trying to master.

- Frequency: 2 – 3 times per Iteration
- *Durations: 15 – 30 minutes*

Example: The team is about to build their first crawler and Andrew reads them some excerpts from Soumen Chakrabarti's book Mining the Web.

Your role

- Lead a few BCBs and acquaint people with the format

Encourage Learning: The Dojo

Coding (and Testing) Dojo

A session where developers and/or automated test engineers gather to discuss programming and testing challenges. One or two people sit at the computer and project on a screen. As they code, people comment out loud. After 5 – 8 minutes, people rotate:

- Frequency: Once per 1 – 2 Iterations
- Durations: 60 – 90 minutes

Your role

- Arrange facilities and equipment
- Help brainstorm fun, challenging exercises (could be a spike, a script for retrieving data, or even code in one of the main modules)
- Be sure to encourage variety and introduce different exercises
- Similarly, testers will enjoy learning how to write test scripts

More info: http://codingdojo.com

Encourage Learning: CoPs

Communities of Practice (CoPs)

Communities of practice are self-organizing groups that form to discuss new topics, challenges, and best practices.

Frequency: Once per 1 – 2 Iterations

- Durations: 30 – 60 minutes
- Format: Any of the formats previously discussed (Inside-Out, BCB, Dojo)

Example: An Automated Testing CoP gathers to attend Ivan's presentation on creating FIT tests for complex branching scenarios.

Your role

- Work with other Scrum Masters and the Release Train Engineer to create and maintain the CoPs
- Unite people from different teams in the program around the same process objectives or activities (e.g., unit testing, automated acceptance testing, system design, infrastructure, deployment, etc.)

5.5 Facilitate collaboration with Architects, System Team, and Operations

System Team, Architects, Ops

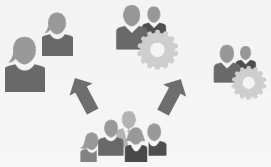
System Architects - Provide architectural guidance to teams, collaborate on new technical research, address technical questions from team members

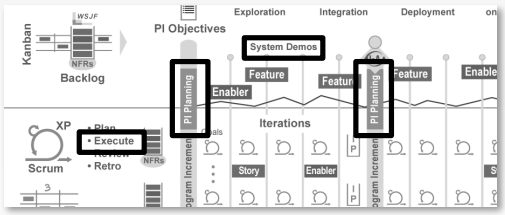
System Team - Assist the program with frequent system integration and testing and development infrastructure support

DevOps - Enable Continuous Delivery Pipeline through infrastructure and process support

Exercise: Collaboration with special teams

- Determine what kind of collaboration with Architects, System Team, and DevOps would be useful in your case
- Consider three main 'venues' for such collaboration:
  - PI Planning
  - Iteration execution
  - System Demo



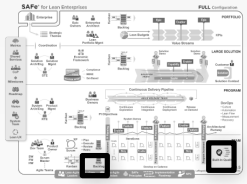


PREPARE 5 min | SHARE 3 min

Lesson summary

In this lesson, you:

- Examined Kanban as a tool to visualize work flow and support Iteration execution
- Built a Kanban board
- Learned some typical ways to measure flow in Kanban
- Explored techniques to build quality into the development process



Suggested Scaled Agile Framework reading:

- *'Team Kanban' article*
- *'Built-in Quality' article*
- *'Integration' guidance article*
- *'Test-First' guidance article*

Also, watch the 'Building Your Kanban board' professional development video on the SAFe Community Platform