Lesson 4
Executing the Iteration

1. Introducing the Scaled Agile Framework
2. Building an Agile Team
3. Planning the Iteration
4. Executing the Iteration
5. Executing the PI

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

Learning objectives

4.1 Visualize the flow of work

4.2 Measure the flow of work

4.3 Build quality in

4.4 Continuously integrate, deploy, and release

4.5 Control flow with meetings

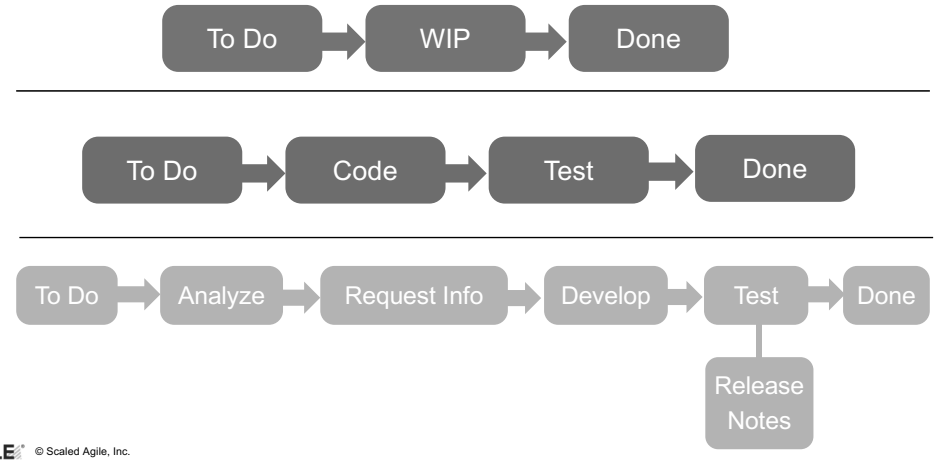
4.6 Demonstrate value

4.7 Retrospect and improve

4.1 Visualize the flow of work

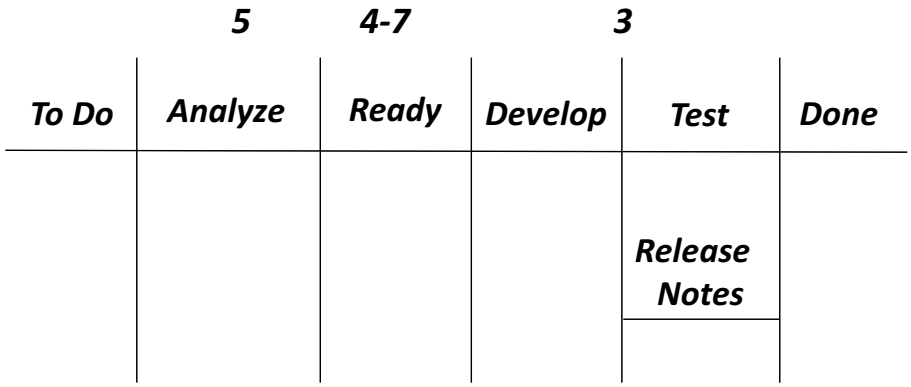
Visualize the flow of work

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



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.



Exercise: Build your initial board

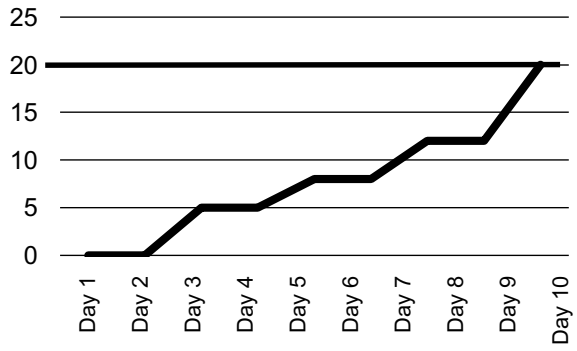
Consider the previous slides and build an initial board for your team.

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

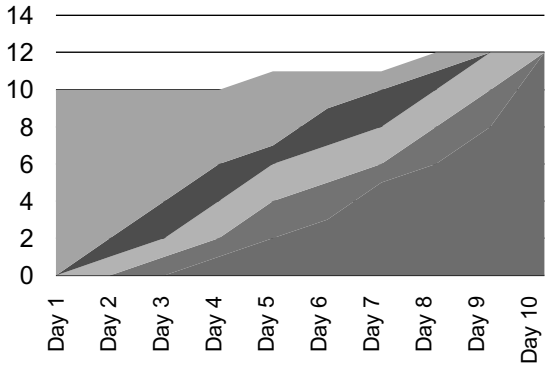
4.2 Measure the flow of value

Track status with burn-up charts and CFDs

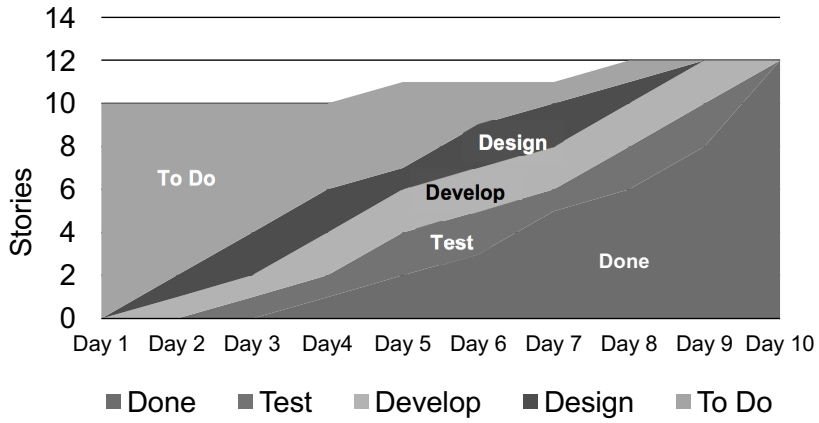
Burn-up



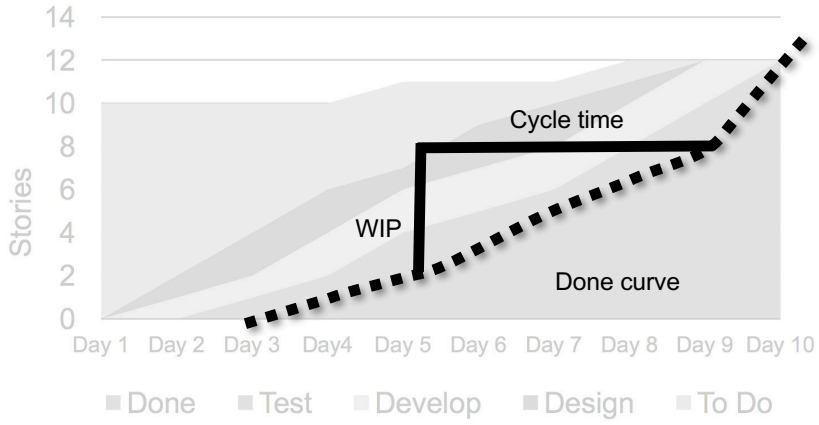
CFD



Understand Cumulative Flow Diagrams (CFD)



What can you learn from a CFD?



4.3 Build quality in

Exercise: Build quality in poster

- Review all the slides in the following section (10 slides)
- Create a poster to reflect the different aspects of building quality in
- Indicate what your team can do to incorporate them into your work

Built-In Quality

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

Building quality in:

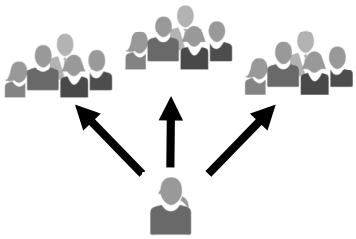
- Ensures that every increment of the Solution reflects quality standards
- Is required for high, sustainable development velocity
- Software quality practices (most inspired by XP) include Continuous Integration, Test-First, refactoring, pair work, collective ownership, and more
- Hardware quality is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

Built-In Quality

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

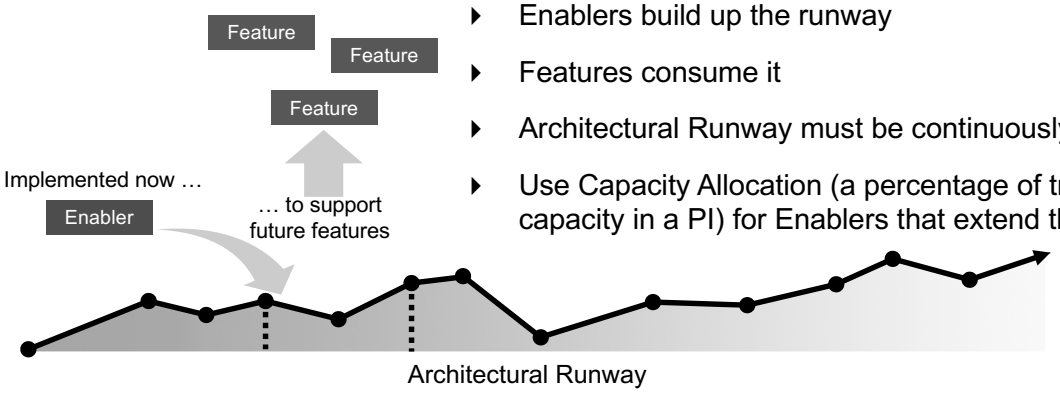


A balance between emergent design and intentional architecture is required for speed of development and maintainability.

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

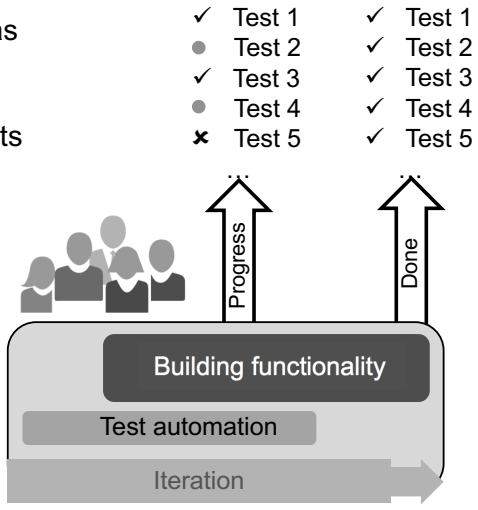


- Enablers build up the runway
- Features consume it
- Architectural Runway must be continuously maintained
- Use Capacity Allocation (a percentage of train's overall capacity in a PI) for Enablers that extend the runway

Test first: Automate now!

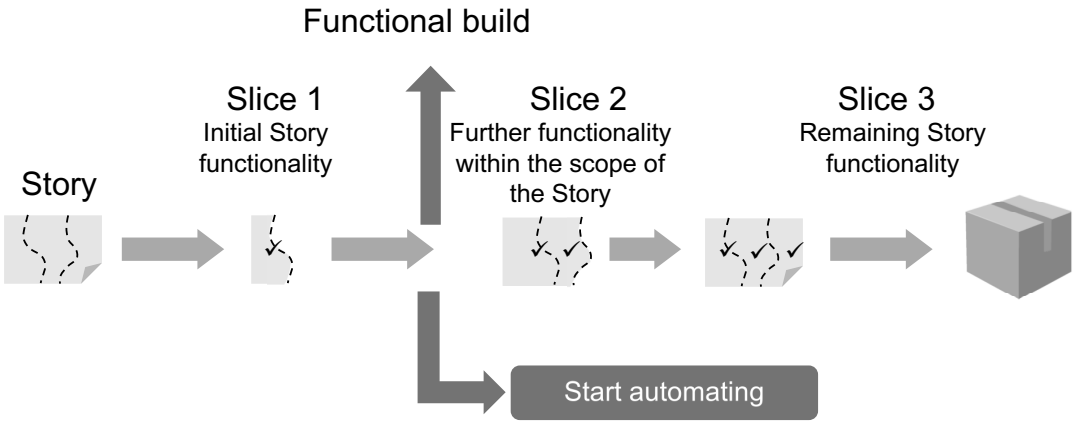
Else, velocity is bottlenecked, quality is speculative, scaling is impossible

- 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

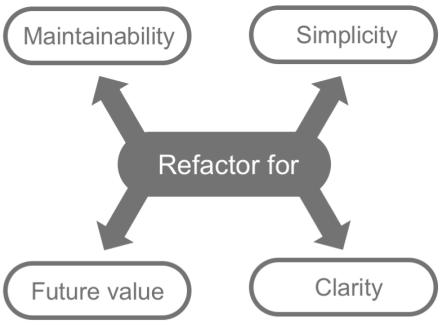
Start automating as soon as the initial functionality is in place.



Refactoring

Refactoring allows teams to maintain high velocity.

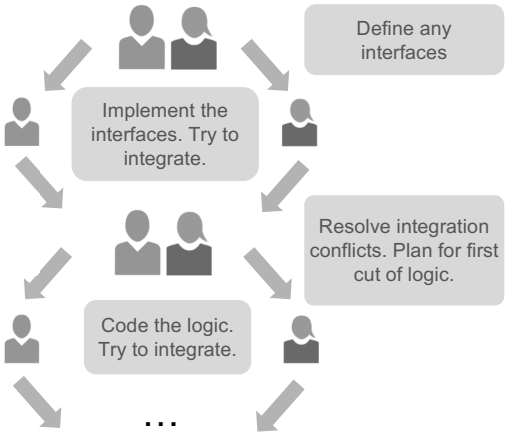
- It is impossible to predict requirements or design in detail
- Refactoring allows teams to quickly correct the course of action
- Emergent design is impossible without continuous refactoring
- Most user stories will include some refactoring effort
- If technical debt is big – teams track and implement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

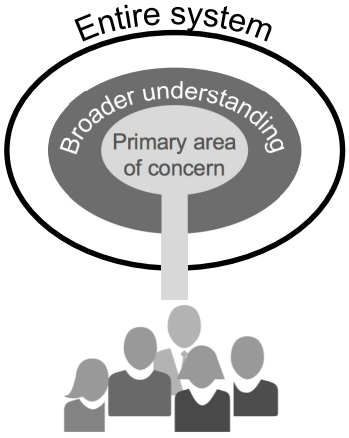


Example user story implementation flow

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



Teams gain a broader understanding of the system

Capture with models, not documents

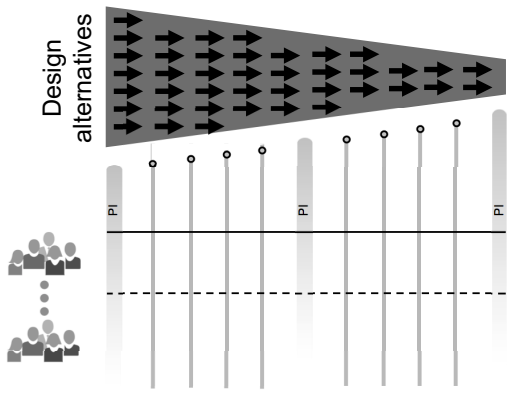
- Models are more adaptable than documents
  - Single changes propagate to all uses
  - Everyone can contribute—all domains, team members
  - Faster, and automated, verification and validation
- Models are more powerful than documents
  - Increases rigor, consistency, accuracy
  - Enables earlier verification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

MBSE
Model-Based Systems Engineering

Set-Based Design

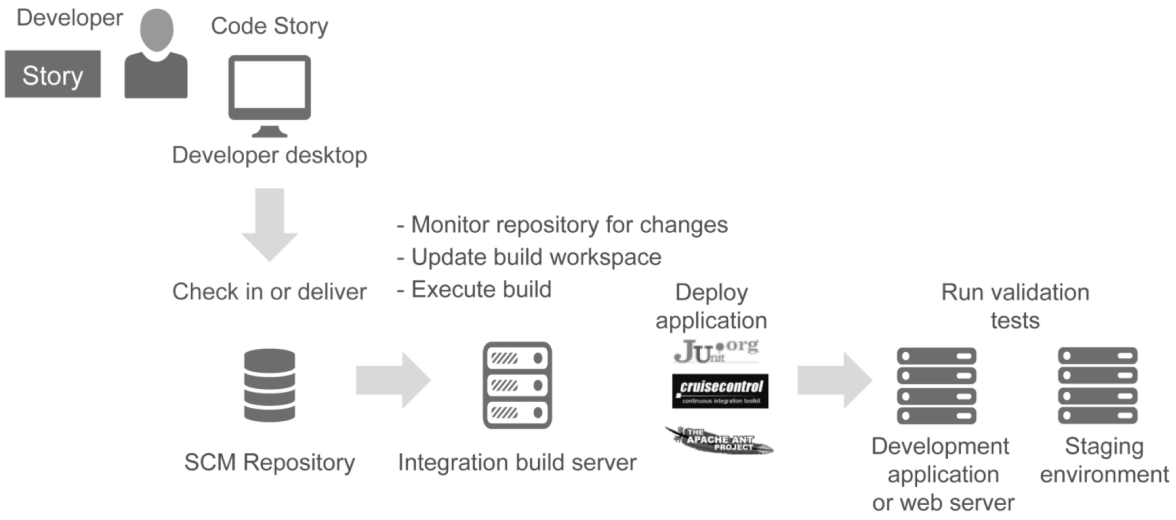
Continuously convert uncertainty to knowledge.

- 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 Continuously integrate, deploy and release

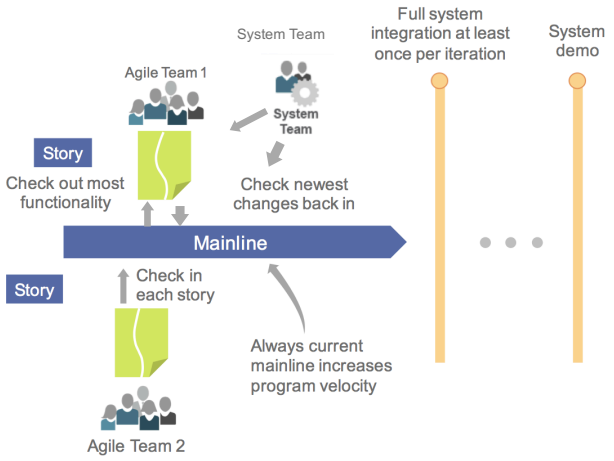
Continuous story integration



Continuous system integration

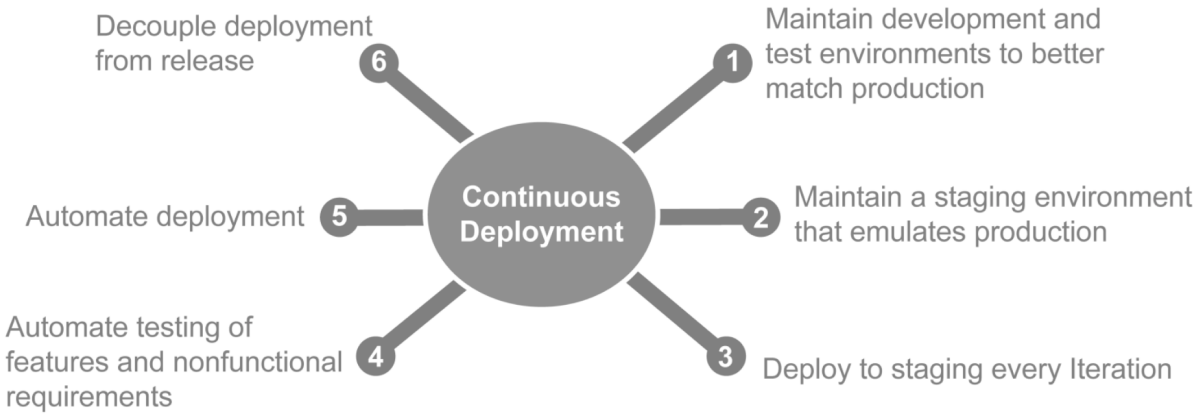
Teams continuously integrate assets (leaving as little as possible to the System Team).

- 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



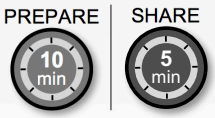
Six Recommended Practices for Continuous Deployment (CD)

CD is an important practice for every team member, team and ART.

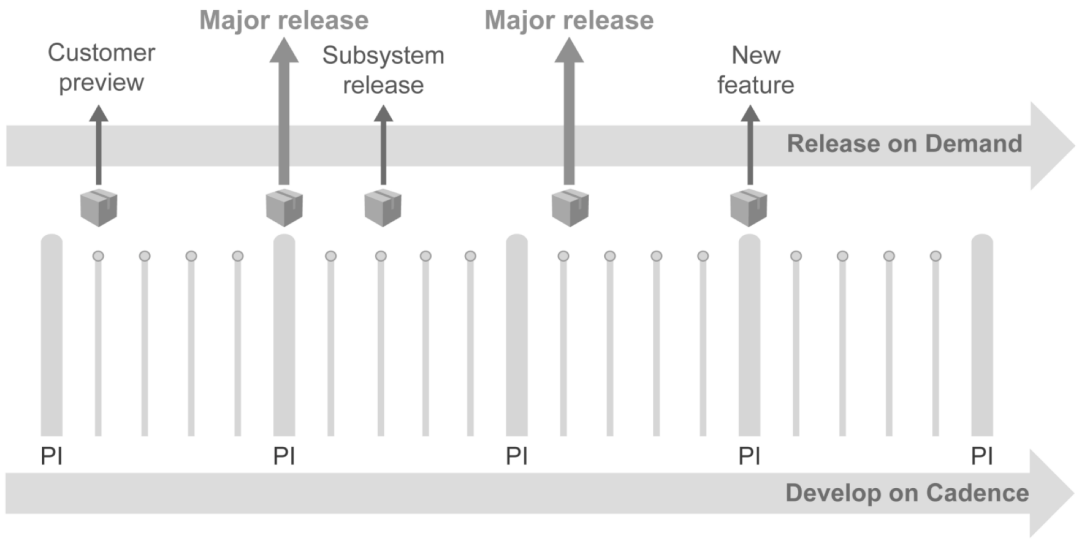


Exercise: Continuous integration and deployment

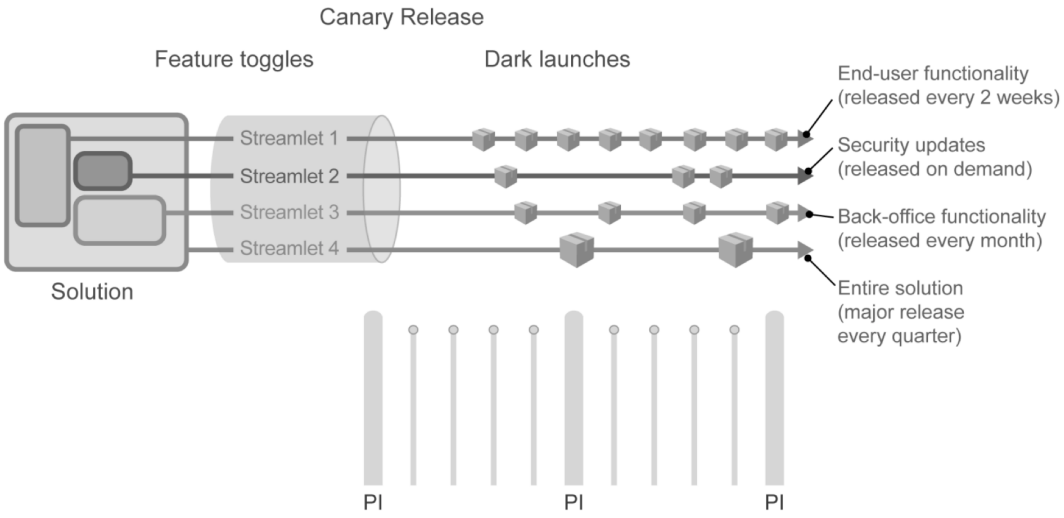
- What are the challenges to continuously integrating?
- What are the challenges to continuously deploying?
- Think about various aspects: environment, culture, tools and people
- In your group, prepare a list of 3 to 5 items that would make it hard to continuously integrate and deploy in your organization and how to solve them
- Be ready to present to the class



Develop on Cadence. Release on Demand Time.



Decouple the release from the solution



Continuous Delivery Culture

- Watch the Bing video
- What cultural changes did the organization need to go through?
- How is your culture or environment ready for continuous delivery?
- Discuss at your table and be prepared to share

Bing: Continuous Delivery
https://youtu.be/3sFT7tgyEQk
3:28

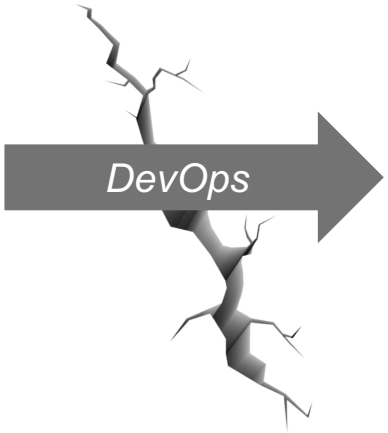
PREPARE 5 min | SHARE 3 min

What is DevOps?

DevOps is an agile approach to bridge the gap between development and operations to deliver value *faster and more reliable.*

Development:

- Create Change
- Add or Modify Features



Operations:

- Create Stability
- Create or enhance services

DevOps is a capability of every Agile Release Train

A CALMR approach to DevOps

Culture Establish a culture of shared responsibility for development, deployment and operations

Automation Automate the continuous delivery pipeline

Lean flow Keep batch sizes small, limit WIP and provide extreme visibility

Measurement Measure the flow through the pipeline. Implement application telemetry.

Recovery Architect and enable low risk releases. Establish fast recovery, fast reversion, and fast fix-forward.

4.5 Control flow with meetings

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
- Meeting 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

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 4 or 5 volunteers to simulate a DSU in front of the class
- 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

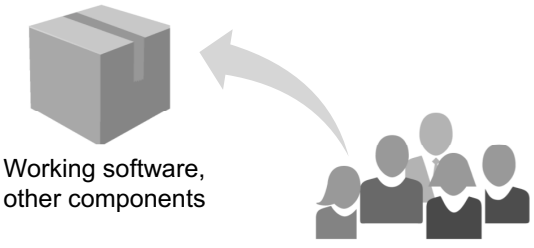
The backlog refinement session is a preview and elaboration of upcoming Stories.

- 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 Product Owner
- Most of the focus is on the next Iteration, but it allows time to discuss future Iterations and even Features for the next PI

4.6 Demonstrate value

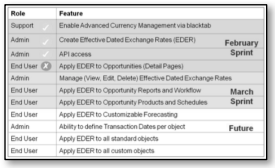
The Iteration Review

- Provides the true measure of progress by showing working software functionality, hardware components, etc.
- Preparation for the review starts with planning
- Teams demonstrate every Story, Spike, Refactor, and NFR
- Attendees are the Team and its stakeholders



Iteration Review guidelines

- Timebox: 1 – 2 hours
- Review 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 Iteration Review 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 team backlog and Team Pi Objectives as needed

The Iteration Review: Two views

The Iteration Review provides two views into the program:

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

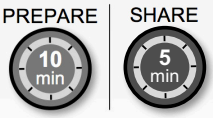
2. How we're doing on the PI
   - Review of PI objectives
   - Review remaining PI scope and reprioritize if necessary

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 are 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 meet 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 meet acceptance criteria • 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
- As a team, build a definition of what it means to you to finish a Story
- Be ready to present to the class



4.7 Retrospect and improve

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?

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

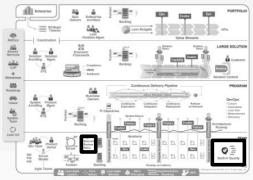
As a group, let's retrospect the course so far

- 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 and visualized the initial flow of work with your team
- Explored how to measure the flow of work
- Recognized techniques to build quality into development process
- Discussed how to continuously integrate, deploy and release value
- Explored how to demo value to team stakeholders
- Practiced running retrospectives



Suggested Scaled Agile Framework reading:

- *"Built-in Quality" article*
- *"Iteration Execution" article*
- *"Iteration Review" article*
- *"Iteration Retrospective" article*