THE AEROSPACE CORPORATION

Agile Approaches for Ground Systems

Jason McKenney
Saachi Sahni
Amit Itzhaki

GSAW Tutorials
February 24, 2025

Your Presenters

Jason McKenney, Associate Director, Software Systems and Acquisition

He has more than 20 years of experience with software development, integration, and test activities. Most of his career has been spent with commercial firms developing software, designing continuous integration pipelines, and leading small teams. He is a key author of an unpublished report on agile ground software system integration and test. He has a B.S. in Telecommunications from the University of Kentucky, and an MBA from California State University – Dominguez Hills.

Saachi Sahni, Associate Member of Technical Staff, Software Systems and Acquisition

She is a recent graduate with experience in data engineering, software development, and process improvement. She has been involved with building an enterprise data fabric, developing several software applications and tools, and improving CI/CD pipelines. She is an author of the AI section for an unpublished report on agile ground software system integration and test. She has a B.S. in Computer Science and Economics from the University of Maryland – College Park.

Amit Itzhaki, Associate Member of Technical Staff, Software Systems and Acquisition

She is a recent graduate from the University of Maryland, College Park with a B.S. in Computer Science. She has experience with software development and test, MBSE languages and modeling tools, and data analysis. She is an author of an unpublished report that compiled lists of best practices for agile ground software system integration and test.

Please visit https://agile.aero.org for more materials.

Today's Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles *[Jason]*
- Framework Roles and Responsibilities *[Jason]*
- Requirements Definition *[Saachi]*
- Testing *[Saachi]*
- Risk Management *[Amit]*
- Scaling Agile *[Amit]*
- Measuring *[Jason]*
- Lessons Learned on Ground Systems *[Jason]*

What is Agile?

Agile is an Approach. Agile is a Culture. Agile is a Mindset.

Agile is an iterative approach to delivery that helps teams deliver value to their customers faster and with fewer headaches.

Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments.

Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

Agile “Frameworks” help make this possible:

- ***Scrum*** *is the #1 framework, best for projects*
- ***Kanban*** *is the #2 framework, best for high-change things, like support*
- ***SAFe*** *is a scaled framework that uses Scrum and/or Kanban and builds on top of it for more synchronicity across a larger enterprise and their ideas.*

Please visit http://agilemanifesto.org for more.

What is Agile?

Key Ideas

Developing and Evolving Working Stuff

Production quality stuff is delivered in short cycles (weeks to months)

Learning & Collaborating Continuously

Customers and developers collaborate to prioritize, plan, and define

Maintaining a Sustainable Pace

Predictable business rhythm to establish process transparency and demonstrate progress

Responding to Change

Practices and mindsets allow for changing realities; each cycle informs the next

Ensuring Quality

Continuous integration, test, and review throughout the lifecycle

Agile is NOT just for software development anymore!

What is Agile?

People Often Confuse Agile with "Agility"

A brief history of its origins, **Values**, and **12 Principles**

- February 2001
- 17 software leaders met in the mountains of Utah
- Why is it called Agile?
- Based in Lean Thinking, Systems Thinking, and templates like the "Toyota Manufacturing Model"
- Agreement on some, but not all, causing a branch into "frameworks"

Please visit http://agilemanifesto.org for more.

What is 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.

Please visit http://agilemanifesto.org for more.

Agile Principles

Guiding Concepts and Ideals aid in Implementing Practices

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 to 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.
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.

http://agilemanifesto.org

Frameworks (simply: "Systems of Work")

Something old, something new, something borrowed

Waterfall (Non-Agile) (Traditional)

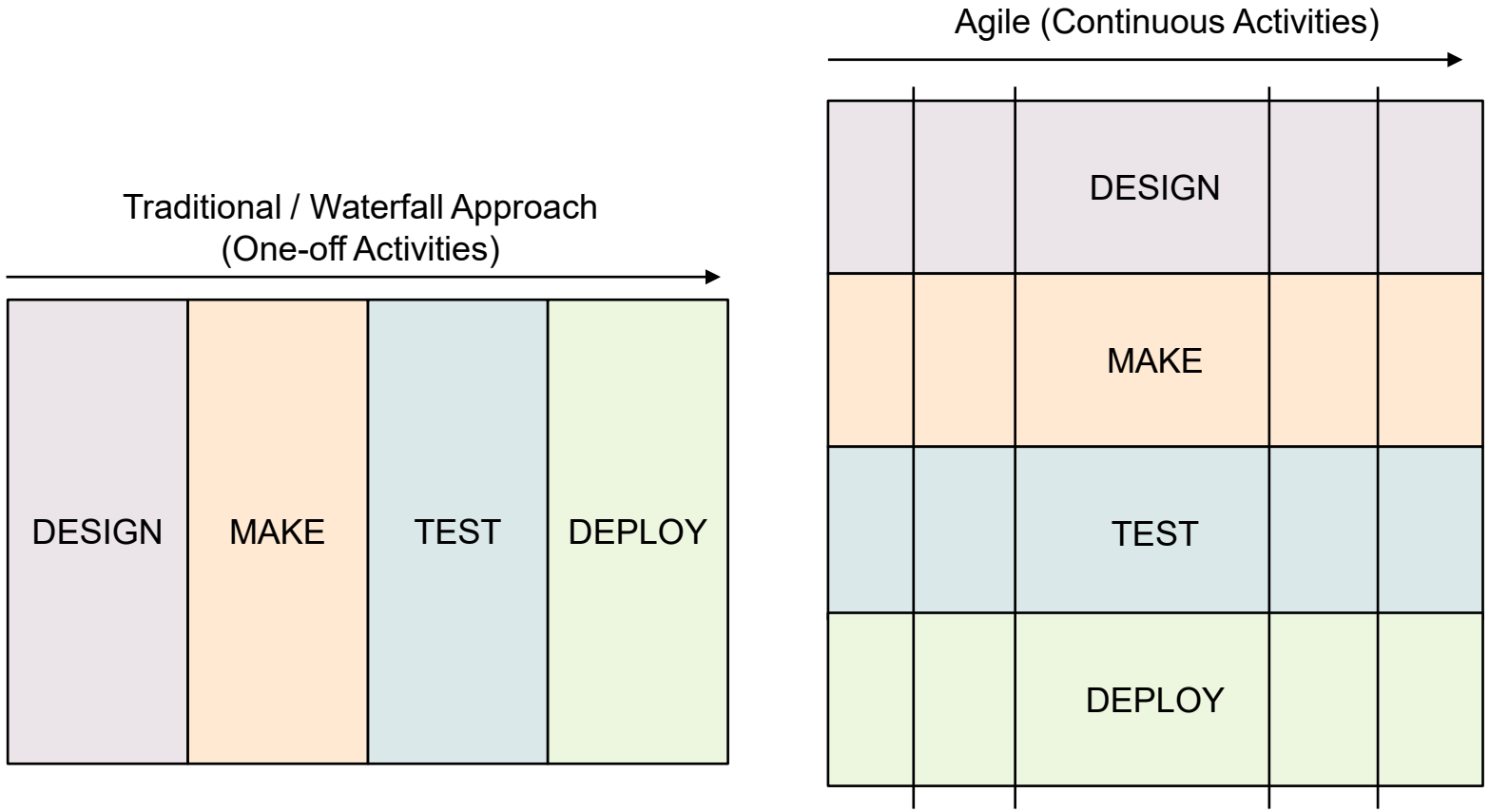
…and…

Agile Frameworks (Modern)

- Scrum
- Kanban
- Scaled Agile Frameworks, Enterprise (SAFe)
- Extreme Programming (XP)
- Crystal
- Lean
- LeSS
- Feature-Driven Development
- Etc.

https://dzone.com/articles/software-development-methodolo

Waterfall and Agile Comparison

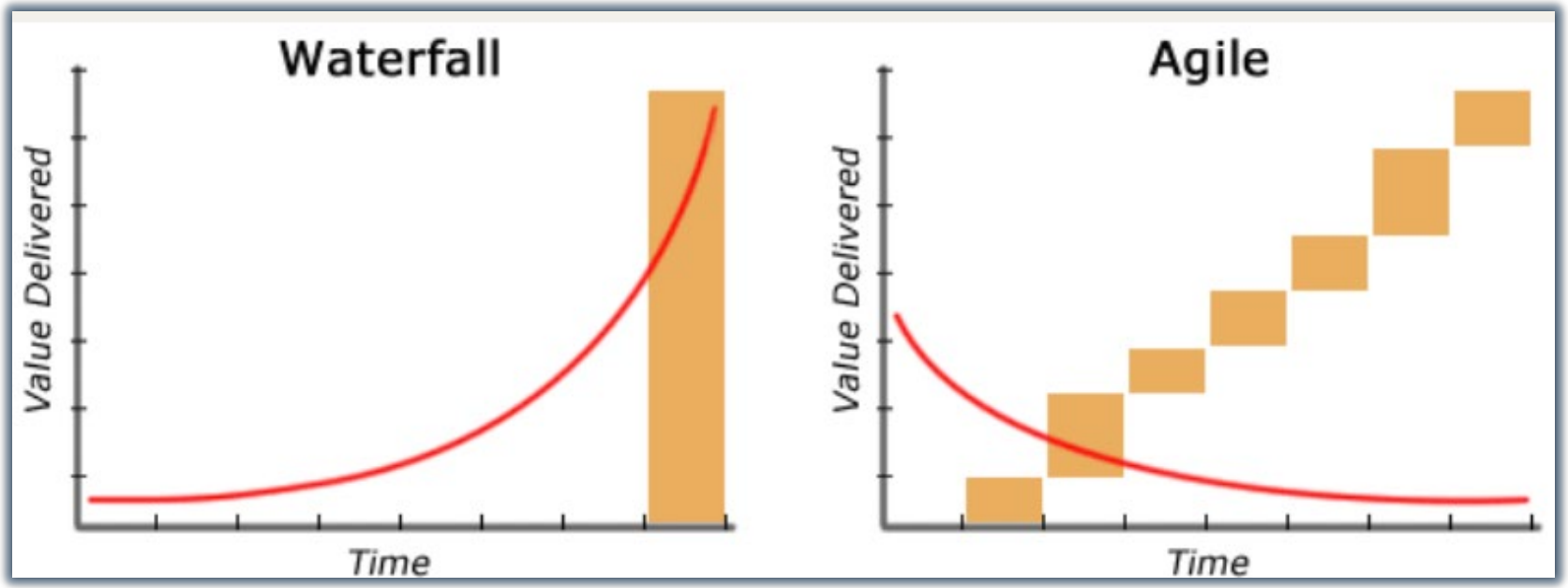


Stage Gates vs. Iterations

Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Agile & Non-Agile

Value-Delivery



Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Agile & Non-Agile

Rate of Change

- Waterfall is best for **low-change** projects
  - Shorter duration
  - Leveraging repeatable processes (already existing)
  - What is designed is what is delivered (for better or worse)
  - If what is designed is changed significantly, it is expensive, extends duration, introduces defects
- Agile is best for **high-change** projects
  - Any duration
  - Leverages frameworks and existing team designs (where possible)
  - Incremental, adaptive delivery
  - More easily accepts changes
- BOTH can handle **fixed dates and fixed scope**
- BOTH can handle **critical paths and dependency planning**
- BOTH can be **understood rapidly and implemented quickly**
- BOTH can be **accountable for quality**

Agile & Non-Agile

Blended vs. Hybrid

- **Blended Agile** is the combination of two or more established Agile methods, techniques, or frameworks
- **Hybrid Agile** is the combination of Agile methods with other non-Agile techniques
- The truth is that most projects of significance are blended, hybrid, or both

WATERSCRUMFALL

SCRUMBAN

BAGILE

WAGILE

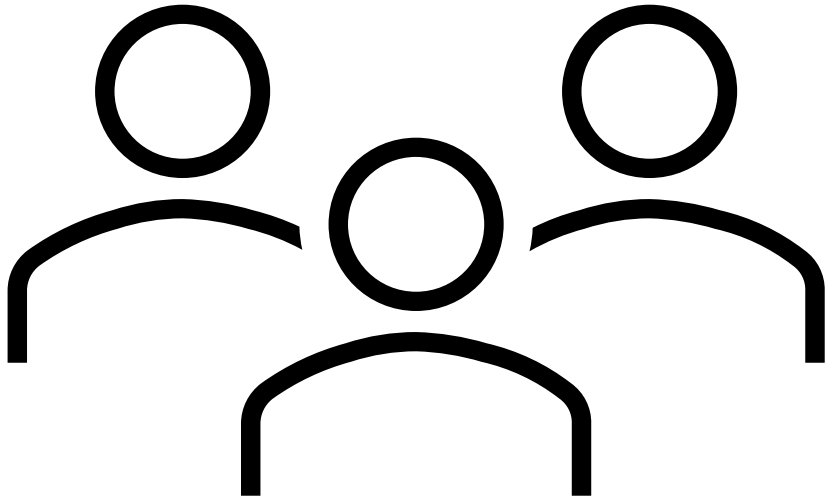
Today's Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles
- Framework Roles and Responsibilities
- Requirements Definition
- Testing
- Risk Management
- Scaling Agile
- Measuring
- Lessons Learned on Ground Systems

Agile & Non-Agile

Team Designs



Agile Teams

What do they usually look like?

- **Cross-Functional**
  - Diverse and self-sufficient
  - Includes all roles that are needed for the functionality to be developed
  - Not just development; includes Testing/QA members
- **Self-Organizing**
  - Team members determine who works on each sprint story
  - Establishes team working agreement
- **Small**
  - Was 6 (+/-3), is now up to max 15
- **Able to keep out change based on framework tools (WIP limits, Class of Service, etc.)**
  - Limit the firefighting!
- **(Can Be) Long-Lived**
  - Teams that have a stable set of members can develop into cohesive and efficient teams

Agile & Non-Agile

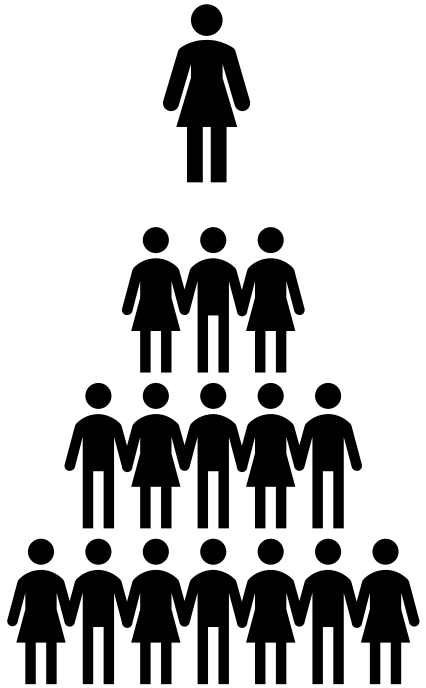
Culture

- **What to expect with an Agile team**
  - All voices – the “Wisdom of the Crowd”
  - Servant leadership
  - Events/meetings based on outcomes
  - Regular cadence (heartbeats)
  - High involvement from customer
  - Working software (outcomes) is the primary measure of success
  - Forecasts based on history not hope
  - Positive response to change
- **What to expect from a non-Agile team**
  - Centralized leadership
  - Command & control leadership
  - Events/meetings based on workstream
  - Limited involvement from the customer (only up front or at milestones, typically)
  - Stage gates (outputs) is the primary measure of success
  - Forecasts based on time remaining / delivery remaining
  - Response to contract

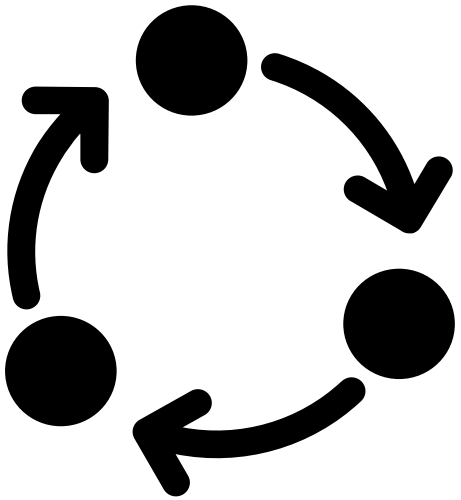
Agile & Non-Agile

Team Designs

Traditional Teams



Agile Teams



Roles

Scrum

- **Product Owner**
  - *“The voice of the customer”*
  - *Owns the backlog and the delivery of value*
  - *Sometimes “Distributed,” but still always centralized*
- **Scrum Master**
  - *Not a Project Manager; coaches, measures, helps remove blockers*
- **Team Member**
  - *Can interoperate with customers*
  - *Designs, builds, and tests the solution*

Roles

Kanban

- Agile Coach
- Team Member
- Who does Product Ownership in Kanban? The whole team

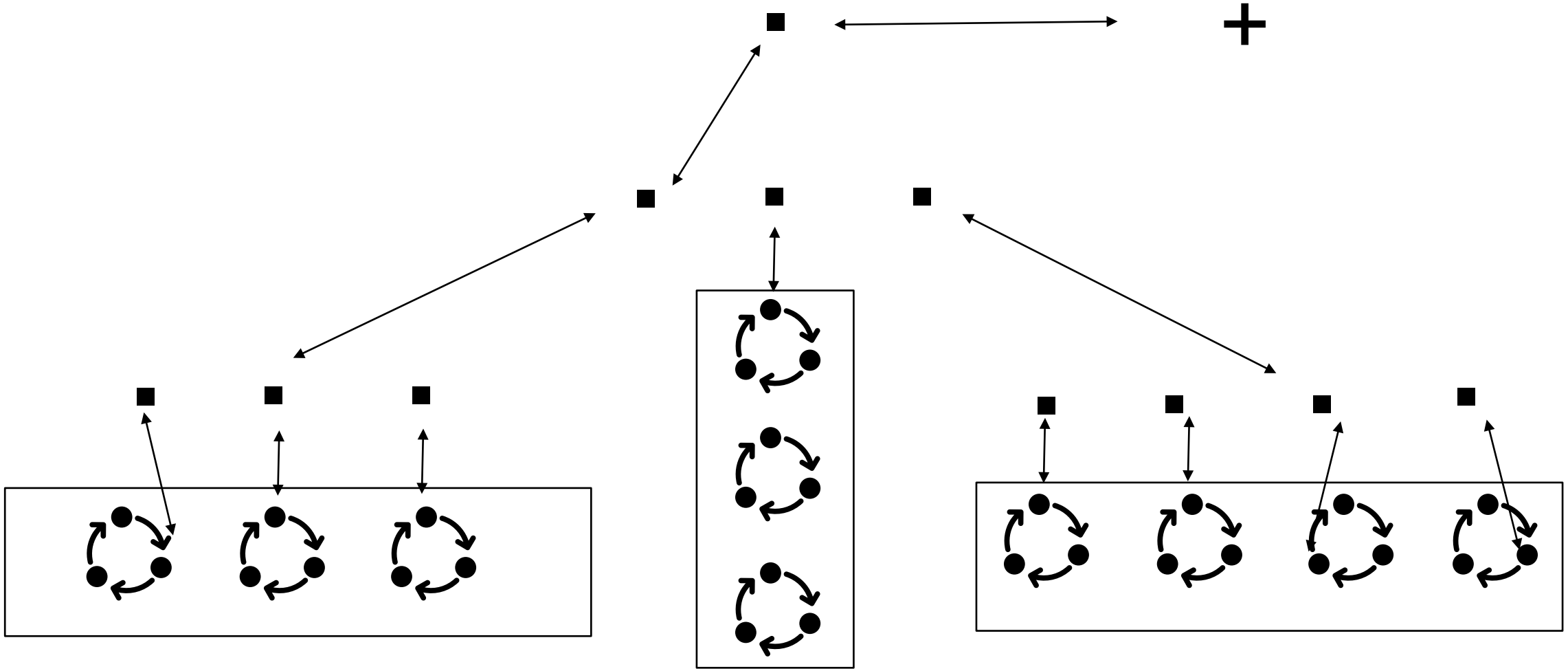
Roles

SAFe

- Release Train Engineer (RTE)
- Enterprise Architect
- Epic Owners
- Scrum and Kanban Roles, too

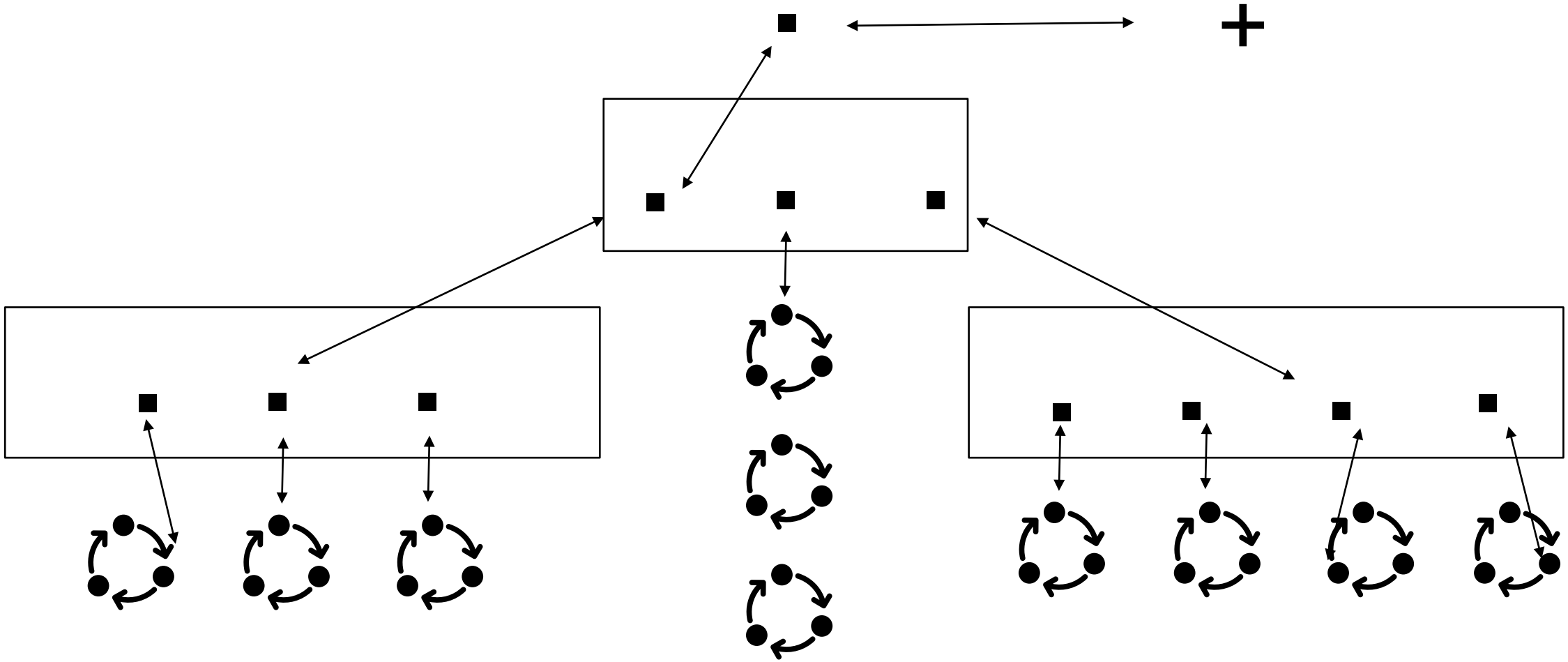
Agile & Non-Agile

Team Designs – Scrum – Team Level



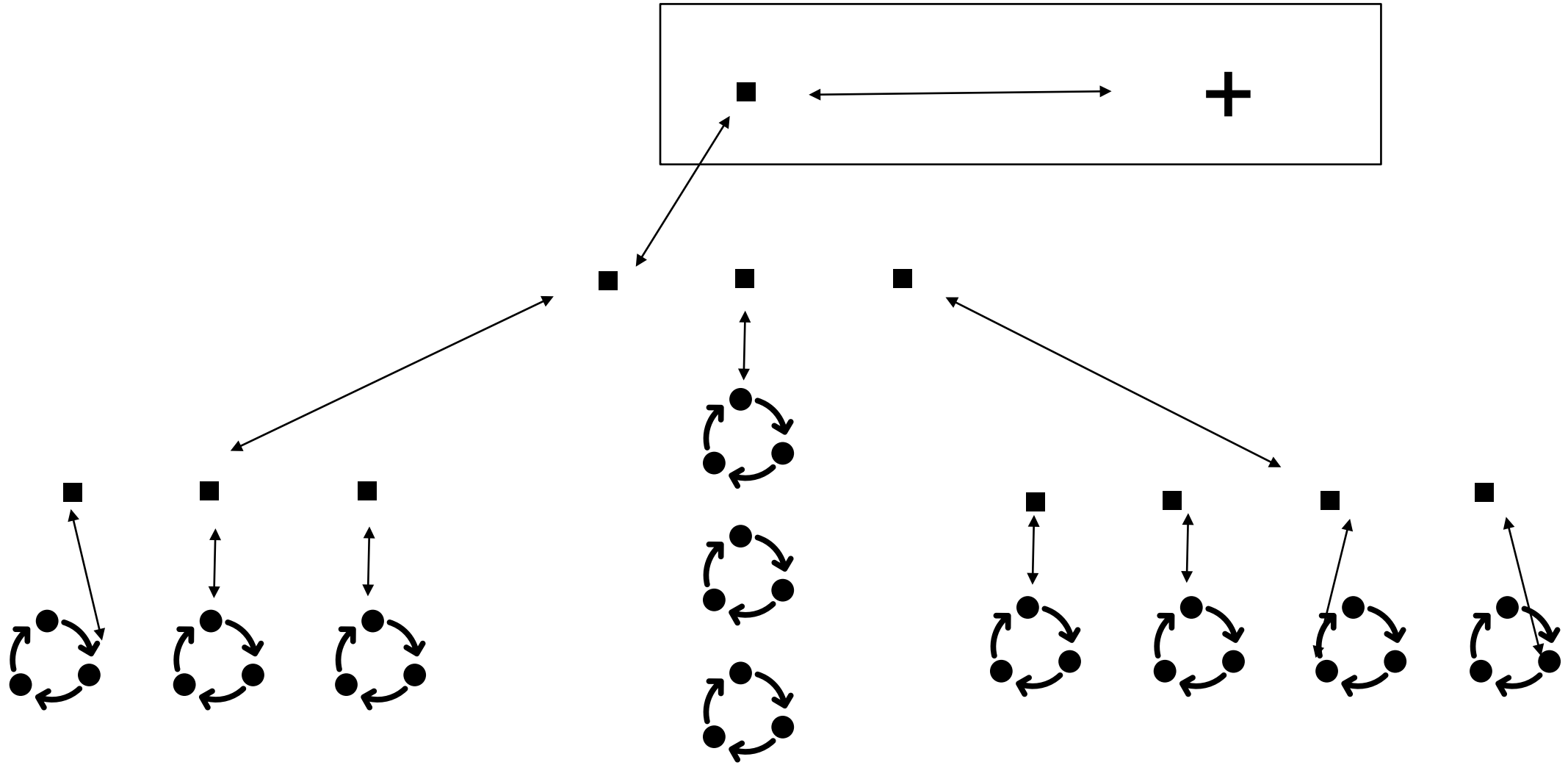
Agile & Non-Agile

Team Designs – Scrum of Scrums



Agile & Non-Agile

Team Designs – Scrum of Scrums of Scrums



Leave Experts as Solo Artists.

Sometimes having several experts that "float in" and advise (maybe with a little task or deliverable here or there) is okay. They do not need to be permanent on a team. Sometimes subject-matter experts' best value is by solving very complex problems for a number of teams while not being on a single team themselves.

Deep Dive – Product Owner (PO)

Critical to Team Success

- **Represents the Customers to the Team**
  - Domain expert resource for the team
  - Collaborates with the Development team to communicate the business needs - ‘Voice of the Customer’
  - Collaborates with internal and external stakeholders to gather needs and requirements
  - Holds the vision for the project
- **Supports Sprint Work**
  - Owns the Product Backlog
  - Determines the business value-based priority for all requirements
  - Collaborates daily with the team
  - Ensures that the product meets customer needs and business goals
  - Accepts/rejects developed items based on quality criteria

Deep Dive – Scrum Master (SM)

- NOT master of the team; master of the practice
- Is a servant-leader, NOT a manager
- Guides the team in understanding and embracing Scrum values, principles, and practices
- Works as a facilitator and supports the team both internally and externally
  - Internally works to resolve team issues and make improvements to process execution
  - Externally protects the team from outside interference
  - Works to remove impediments to team productivity
- Not necessarily a full-time role; can span teams or can be an individual contributor to the development team (depending)

Deep Dive – Agile Coach

- Experienced Agile Practitioner
- Acts beyond Scrum Master, if supplemental
- Coach and Mentor for teams and individuals
- Champions the use of Agile
  - Help the organization achieve above the desired results
  - Help the team develop and get healthier together (or recover more completely when not healthy)
  - Help each person take the next step on their Agile journey
- Should be considered a ‘ley’ person from a contracting perspective
- Recommended GAO Effective Practice
  - “Ensure all teams include coaches or staff with Agile experience. This practice stresses the importance of including on each team those with direct experience in applying Agile. While training is helpful, hands-on experience helps the team members learn and adjust.” [GAO 2012, 2020]

https://www.gao.gov/products/gao-20-590g

What About Traditional Roles?

A Look at How These Map

Role	Agile Team
Project Managers	Role eliminated
Functional Managers	“Outside the circle of the team”
Analysts	Team Member
Programmers	Team Member
Database Administrators	Team Member
Testers	Team Member
UX Designers	Team Member
System Engineers	Team Member

What is the role of Project Managers?

- **Waterfall Projects**
  - Manages all aspects of the project
    - Scope
    - Cost
    - Quality
    - Staffing
    - Communication
    - Risk
    - Procurement
    - Etc.

- **Agile Projects**
  - Project Manager role eliminated or supplemental if multi-team collab
  - Most responsibilities shifted to existing team roles, particularly the PO

Hybrid programs that have Agile team execution but need a top-level person may have a position named "Program Manager" with some of these characteristics.

The most applied framework is…

SCRUM

https://www.scrumalliance.org/

Scrum Deep Dive

Level-Set

- What does the word mean?
- How do we know its standards?
- Main tent-poles
  - Timebox (Sprint) (Consistent) (Usually with a goal) (SPRINT)
  - Roles
  - Events
  - Artifacts
    - Backlog, Sprint Backlog, Increment Delivery
- Other borrowed notions
  - Kanban board
  - User Stories / Story Format
  - “Smallest Deployable Unit”
- Estimation Practices
  - Ditch ‘em, go for issue count instead (certain conditions apply)
  - Story Points
  - Velocity in avg. # story points achieved per sprint

https://aerosource2.aero.org/confluence/display/SSS/Scrum+Guide

Roles

Level-Set

- Product Owner (“The What”)
  - Owns the backlog
  - Owns the ranking / value calculation
  - Works with the customer
  - Owns the P & L
- Team Member (“The How”)
  - Commits the backlog to their queue of work
  - Defines the solution
  - Delivers the solution
- Scrum Master (“The Advisor”)
  - Keeps the team healthy
  - Tracks and removes blockers
  - Provides reporting
  - Facilitates some events
  - Servant-leader

https://aerosource2.aero.org/confluence/display/SSS/Scrum+Guide

Events

Level-Set

- Sprint Planning
  - Create near-term plan & commit
- Sprint Review
  - Demonstrate finished work for customers
- Sprint Retrospective
  - Open and honest inspection by the team, resulting in actions to improve
- Daily Standup (aka Daily Scrum)
  - Risk Management throughout the sprint
- Other Helpful Ideas
  - Backlog Refinement (PO-owned; the what; healthy candidate items)
  - Technical Workshops (Team-owned; the how; emergent architecture or specialty decisions)

https://aerosource2.aero.org/confluence/display/SSS/Agile+Events

How it looks in practice

Typical 10-day sprint (doesn't have to be M-F)

M	T	W	H	F	M	T	W	H	F
Sprint Plan	D	D	D	D	D	D	D	D	Sprint Review
									Sprint Retro
		(Midweek) Refinement					(Midweek) Refinement		

Note: the sprint duration should be "as long a time as the team can keep change out"

How it looks in practice

Typical 10-day sprint (doesn't have to be M-F)

SPRINT BACKLOG

Item-2 | Size = 3
Item-6 | Size = 1
Item-7 | Size = 8
Item-11 | Size = 8

If they finish 100%, total velocity is…

20 points

PRODUCT BACKLOG

Item-2
Item-5
Item-10
Item-1
Item-3
Etc.

Note: the sprint duration should be "as long a time as the team can keep change out"

Three Key Benefits of Agile

Notice it's not the metrics

- Predictability (these people deliver X… always)
- Focus (no more fire drills)
- Improved customer satisfaction (working stuff over and over)

Agile is designed for EFFECTIVENESS, not EFFICIENCY. The latter may come as a result, but it's more about spending your time and resources on the right stuff and getting that right stuff to the customer faster.

https://www.atlassian.com/agile/agile-at-scale/beyond-the-basics-whitepaper

Artifacts

What is essential and what is optional

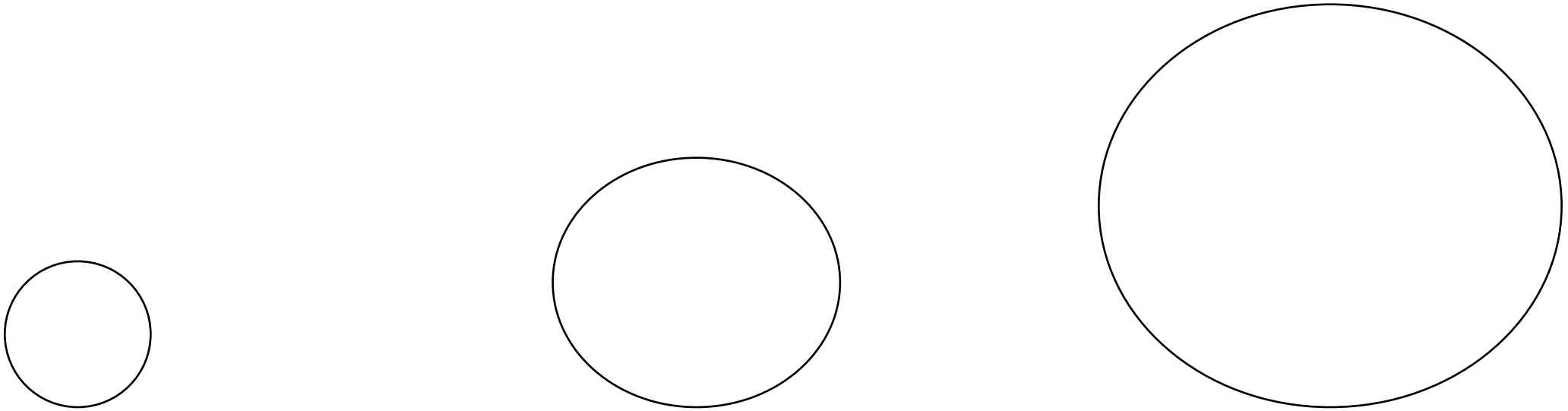
- Product Backlog (featuring Product Backlog Items – PBIs)
- Sprint Backlog (featuring Sprint Backlog Items – SBIs)
- Increment (Result of delivery)

- Other Helpful Ideas
  - Definition of Ready / Definition of Done
  - Team Agreements
  - Roadmaps / Story Maps / Feature or Capability Analysis
  - Lean Technical Manuals / Documentation
  - “Source of Truth”
  - Communications Plan
  - Risk Management Plan / Risk Log
  - Schedule
  - Other reporting or whatever the team needs

https://aerosource2.aero.org/confluence/display/SSS/Scrum+Guide

Sizing

Relative Estimation Has Replaced Fixed Time Estimation



https://aerosource2.aero.org/confluence/display/SSS/Story+Points

Scales

You can use anything, but one is most popular

- Fibonacci Numbers
- Expand with risk

1, 2, 3, 5, 8, 13, 21, 34…

https://en.wikipedia.org/wiki/Fibonacci_number

The second most applied framework is…

KANBAN

https://www.atlassian.com/agile/kanban

Kanban Deep Dive

Level-Set

- What does the word mean? Japanese term for "Visual card"
- Where did it derive from? Toyota Production System in the 1950s
- Main tent-poles
  - Roles
    - Product Owner, Agile Coach, Team Member
  - Flow (No Timebox)
  - WIP Limits
  - Class of Service / Class of Service Policy used to categorize priorities of tasks
- Other important notions
  - Workflow
  - User Stories / Story Format
- Estimation Practices
  - Relative estimation
  - Story Points
  - Velocity in avg. cycle time of work in progress

https://en.wikipedia.org/wiki/Kanban

How it looks in practice

Typical Kanban Board

TO DO	***IN PROGRESS***	***DONE***
(aka "Backlog")	*(WIP Limit = 1)*	
Item-89		
Item-43		
Item-123		
Item-90		
Item-98		
Item-44		
Etc.		

How it looks in practice

Typical Kanban Board

TO DO	***IN PROGRESS***	***DONE***
(aka “Backlog”)	*(WIP Limit = 1)*	
	Item-89	
Item-43		
Item-123		
Item-90		
Item-98		
Item-44		
Etc.		

How it looks in practice

Typical Kanban Board

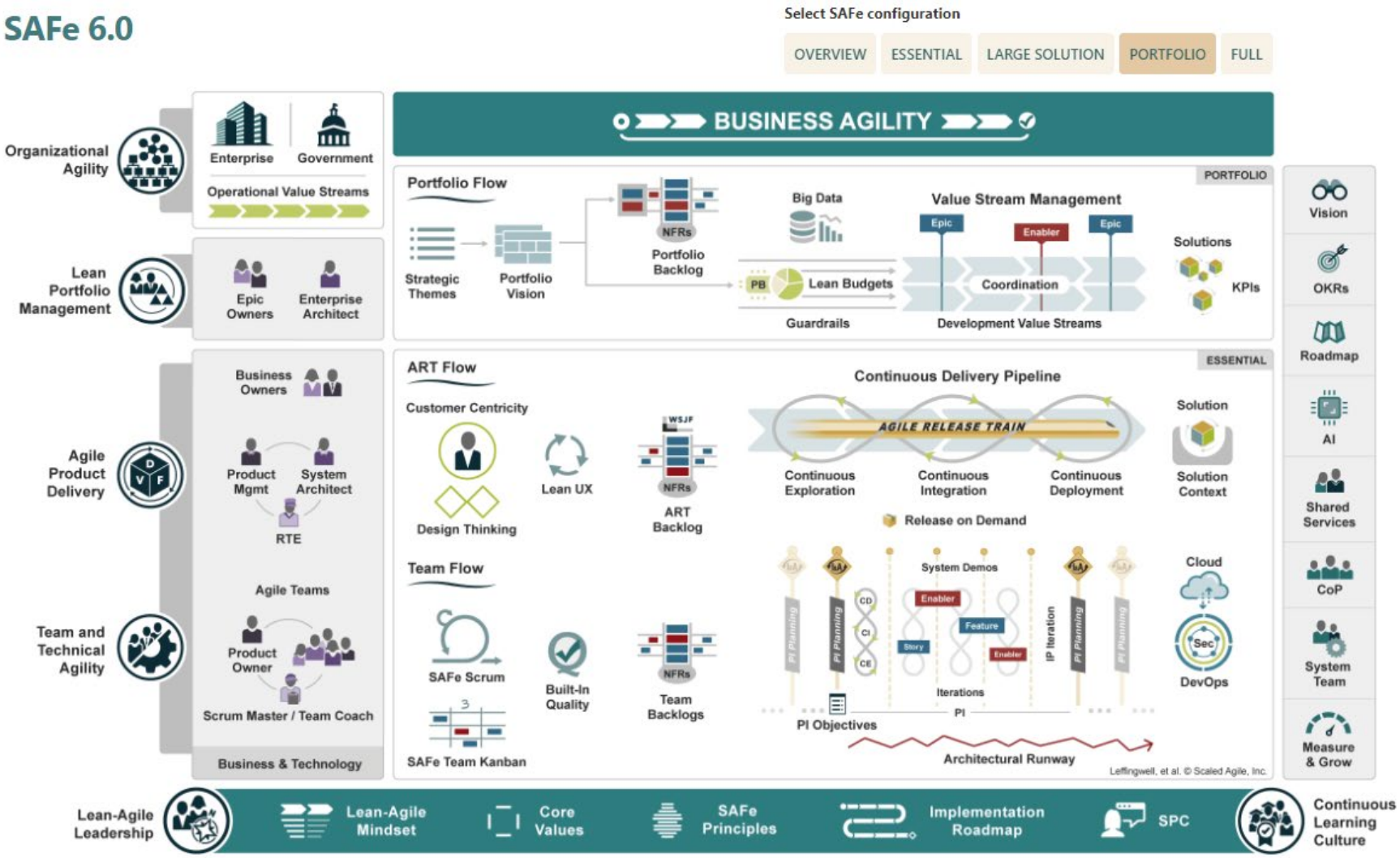
TO DO	***IN PROGRESS***	***DONE***
(aka “Backlog”)	*(WIP Limit = 1)*	
		Item-89
	Item-43	
Item-123		
Item-90		
Item-98		
Item-44		
Etc.		

The third most applied framework is…

SAFe

https://scaledagile.com/

SAFe "Configuration(s)"



https://scaledagile.com/

Additional Practices for Large Scale Programs

- Big and Small timeboxes
  - Sprint / Iteration – 1 to 4 weeks
    - Release a potentially shippable product by the end of each Sprint
    - Focus on work within the team
    - Activities: Sprint Planning, Sprint Demo, Sprint Retrospective
  - Program Increment – quarterly
    - Release a system-level product by the end of each program increment
    - Focus on coordination between teams
    - Activities: Program Increment Planning, System Demo, Inspect and Adapt, Scrum of Scrums, Product Owner sync
- Additional roles
  - Program level
    - Release Train Engineer, Product Manager, System Architect / Engineer

Additional Practices for Large Scale Programs

- Architectural Runway
  - Proactively plan and develop necessary technical foundation and infrastructure needed for developers to implement near-term features
  - Avoid big design upfront, design one step ahead of development team
    - The bigger the aircraft (teams), the more runway is needed
  - Sample architectural runway activities
    - System Engineering team develops overall architecture or design ahead of the development team
    - Infrastructure team performs trade study on COTS selection
    - Tech Support team perform infrastructure upgrades
- Program-level planning or PI Planning
  - Team members from all teams attend a face-to-face planning event at the beginning of each program increment (PI)
  - Plan what to deliver in the upcoming PI timeframe (e.g., quarter)
  - Stakeholders, Customers, Operators should also attend planning

Additional Practices for Large Scale Programs

- System demo
  - System-level demonstration; ideally at the end of each Sprint
  - Demoed product is fully integrated across all teams
- Innovation and Planning (IP) or Hardening, Innovation and Planning (HIP) Sprint
  - The last Sprint of each PI is dedicated for innovation, inspect and adapt, training, preparing for System Demo, buffer for addressing technical debt
  - PI Planning preparation occurs during the IP/HIP, with the PI Planning ceremony at the very end of the IP/HIP
- Bigger than program level?
  - Solution level – multiple programs or including supplier teams
  - Portfolio level – multiple product lines, multiple solutions

Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles
- Framework Roles and Responsibilities
- Requirements Definition
- Testing
- Risk Management
- Scaling Agile
- Measuring
- Lessons Learned on Ground Systems

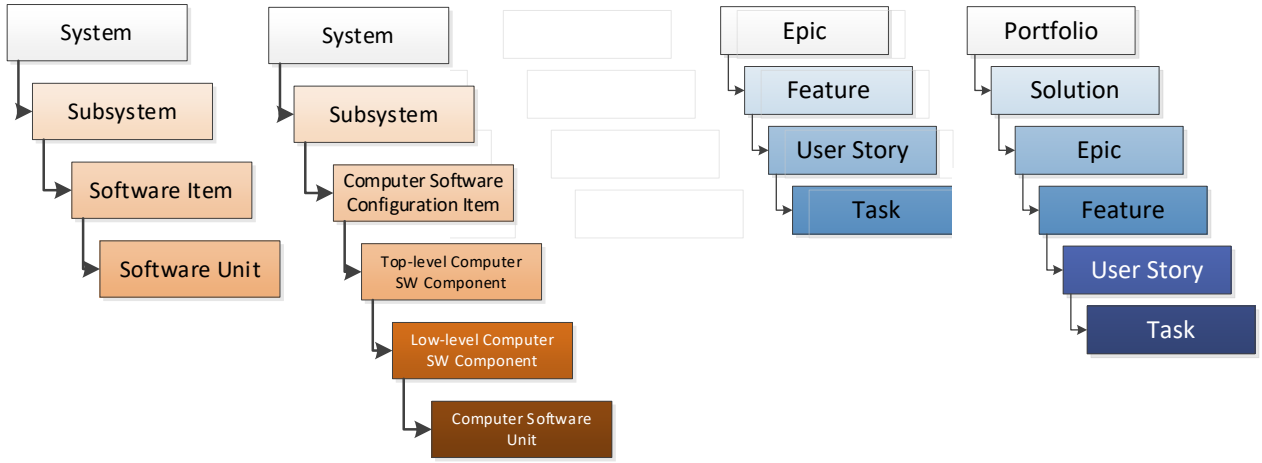
Mind Shift to "Capabilities"

- Most projects have incomplete picture of requirements upfront
  - Generic statements early on and elaborate further
- Requirements may evolve as new information arrives
  - Welcome changes by adjusting details and allowing re-prioritization
- Think about incremental development of end-to-end Capabilities
- Possible use of prototypes or models to explore and illustrate Capabilities
- Agile Capabilities are decomposed until they fit in a Sprint (if Scrum)
  - Ensures ability to deliver working, valuable, tested software
- Stories are written from users' perspective
- Agile requirements
  - A placeholder for later conversations when it is time to develop
  - Discussed with the whole team: developers, testers, customers, user representatives
    - Shared understanding, incremental feedback

Traditional vs Agile Requirement Structure

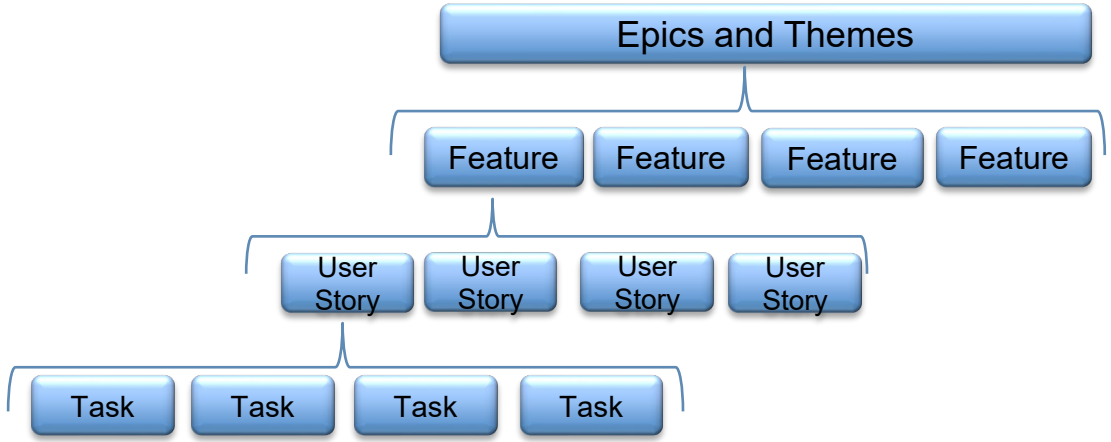
Traditional "Shalls"
- Hierarchically-focused
- System-oriented
- Tightly-coupled
- More on big-bang delivery

Agile "Capabilities"
- Capability-focused
- User/Role-oriented
- Loosely-coupled
- Support incremental delivery



Item Types (A Sample)

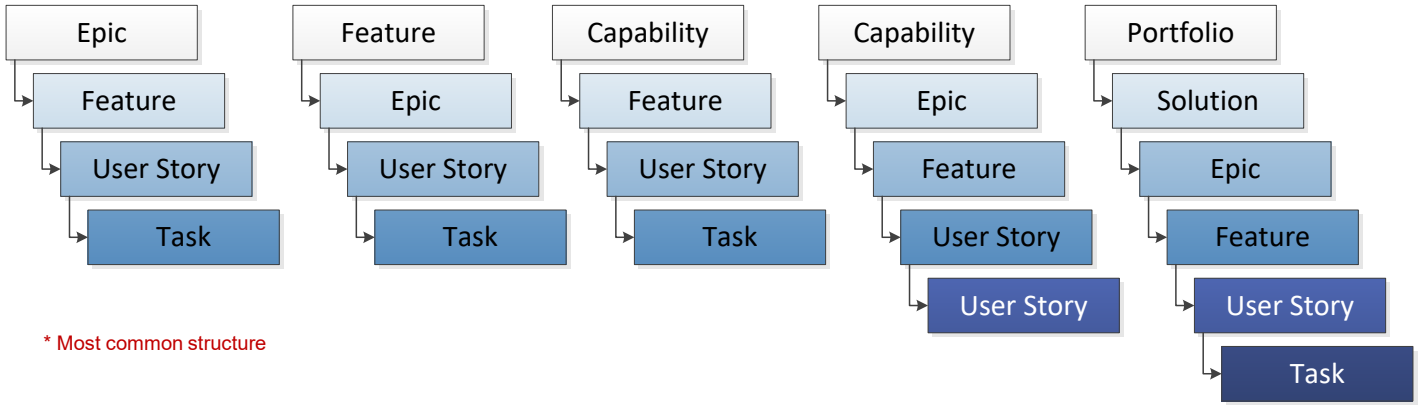
- **Epics and Themes**
  - High level system goals and objective
- **Feature**
  - Description of system functionality that can be accomplished in a single release
- **Stories / Enablers**
  - A description of system functionality as told by a specific user role
  - Should be accomplished within a single Sprint
- **Tasks**
  - A single specific item that needs to be accomplished for an item or story; Should be roughly a day's effort



Agile Requirement Structure

Requirement Structure determined by Program

- Several ways to structure an Agile requirement



* Most common structure

- "**Portfolio**," "**Product Vision**", "**Solution**", and "**Epic**"
  - Generally, refer to high level requirements, typically well-defined in the pre-award stages
- "**Feature**" and "**Capability**"
  - Generally, refer to medium-level requirements, somewhat predefined in pre-award work, but may require further definition in collaboration with the contractor.
- "**User Story**"
  - Low level requirements but may need to be further decomposed into "tasks"

A Model for "Requirements" Flow

Program Level

- Product Vision
  - High level, time-independent vision for functionality of the system/product
- Agile Artifacts - Features
  - Epics decomposed into Features
  - Services provided by the system/product that fulfill stakeholder needs
  - Comprise the Program/Product Backlog
  - May require Acceptance Testing as they span multiple User Stories
  - Most likely take several Sprints to complete
  - Non-functional requirements (NFR)
    - Generally, reflects system behavior
    - Impact spans User Stories and applies as Features are completed and/or new Features are needed; may need to be treated as Program/Product Backlog constraints

Feature examples:
- Alert Presentation and Management via management dashboard
- Data management support for in-motion and at-rest data
- Define processes for framework Ingress and Egress
- Develop data playback capability

NFR
- Application Scalability
- Extensible data model
- Improve/Simplify developer application debugging capabilities

Levels Represented in Scaled Agile Framework (SAFe)

A Model for "Requirements" Flow (continued)

Team Level

- Team Vision
  - Plan for features implementation to accomplish the Product Vision
- Agile Artifacts - User Stories
  - Features decomposed into User Stories
  - Primary mechanism for realizing customer's requirements through the value stream, via needs analysis and implementation in code
  - Brief negotiable statements of intent describing something the system/product needs to do for the user
  - Comprise the Sprint Backlog
  - Elaborated at the last responsible moment with desired functionality and Acceptance Criteria

As an ABC system application user,

I want a capability for applications to create and submit alerts to the ABC for persistence, visualization, and disposition

so that applications can report significant events to dashboard operators for situation awareness and action

As an ABC application user,

I want a C++ SDK for the ABC system that, at a minimum, supports C++11, C++14, C++17 and is developed using source code that follows a documented set of C++ coding guidelines that is referenced in the SDK documentation

so that I can develop or update applications using modern C++ language features and libraries.

Levels Represented in Scaled Agile Framework (SAFe)

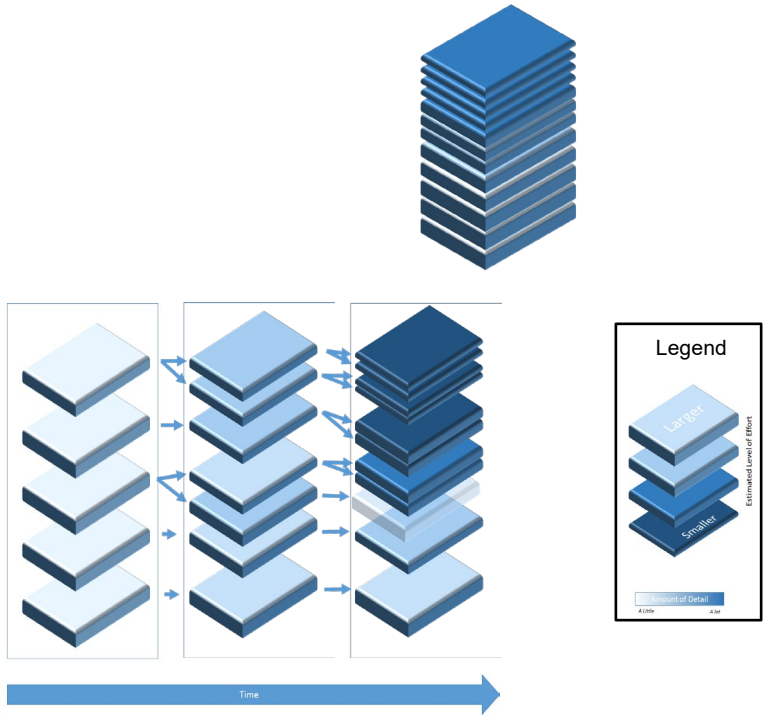
Agile Approaches for "Requirements"

- Ensure that high-level Backlog Items are appropriately broken down into Stories (known as "Refinement")
- Stories address who, what, and why and will also have an associated when (prioritization)
  - Typical format
    - As a <type of user>, I want <some goal> so that <some reason>
      - As an ABC system application user, I want a capability for applications to create and submit alerts to the ABC for persistence, visualization, and disposition so that applications can report significant events to dashboard operators for situation awareness and action
  - Apply INVEST or other quality standard (Definition of Ready, Definition of Done, etc.)

Product Backlog

Prioritize Highest Risk and Complex Items First

- Scrum uses placeholders for requirements
- Gradually refined with more detail until there is enough to work on
  - *Prioritized list of desired functionality*
    - Mission Value
    - Technical Difficulty
    - Level of Risk
  - *Central shared understanding for the team*
    - What to Build
    - Order to Build it
    - When is it considered ‘Done’
  - *Contains Backlog Items*
    - Epics/Features
    - User Stories
    - Defects
    - Technical Improvements
    - Technical Debt



Acceptance Criteria - Sample

User Story:
As a bank customer, **I want** to see my fee for my loan, **so that** I know the remaining balance

Acceptance Criteria:
- The loan fee balance is displayed
- The loan fee balance is calculated
- The balance is not displayed if an unknown user identity is applied

User Story:
As a Paypal account holder, **I want** to withdraw my pending credit from PayPal, **so that** I can have money in my bank account

Acceptance Criteria:
- I can see on Paypal account that there is pending credit
- I can choose what amount of credit to withdraw
- I can see my bank account balance when I have chosen to withdraw credit
- I can't top up bank account when there are no pending credits in my Paypal account

[Source: What Are Acceptance Criteria: Explanation, Examples and Template https://existek.com/blog/what-are-acceptance-criteria/]

Definition of Ready - Sample

User Story Definition of Ready

- Written correctly in the format of
  - “As a <type of user>, I want <to perform some task> so that I can <achieve some goal/benefit/value>.”
- Measurable estimate of effort (i.e., Story Points) assigned
- Acceptance criteria defined in the STP for possible outcomes
- Test descriptions to verify correct operation defined
- Test descriptions to evaluate failure cases as well as common weaknesses defined in MITRE’s Common Weakness Enumeration (CWE) at cwe.mitre.org defined
- Failure scenarios and associated recoveries described
- Data and data sources needed to design, build and test the User Story defined
- Needed user information and feedback to aid in designing and testing of the User Story is available
- Non-functional requirements (NFR), dependencies with other User Stories and system integration requirements and interfaces needed to perform the User Story are identified
- No external dependencies will block the User Story from being completed
- Traceability to the TRD, System/Subsystem Specification (SSS), and Use Case Activity Diagram defined

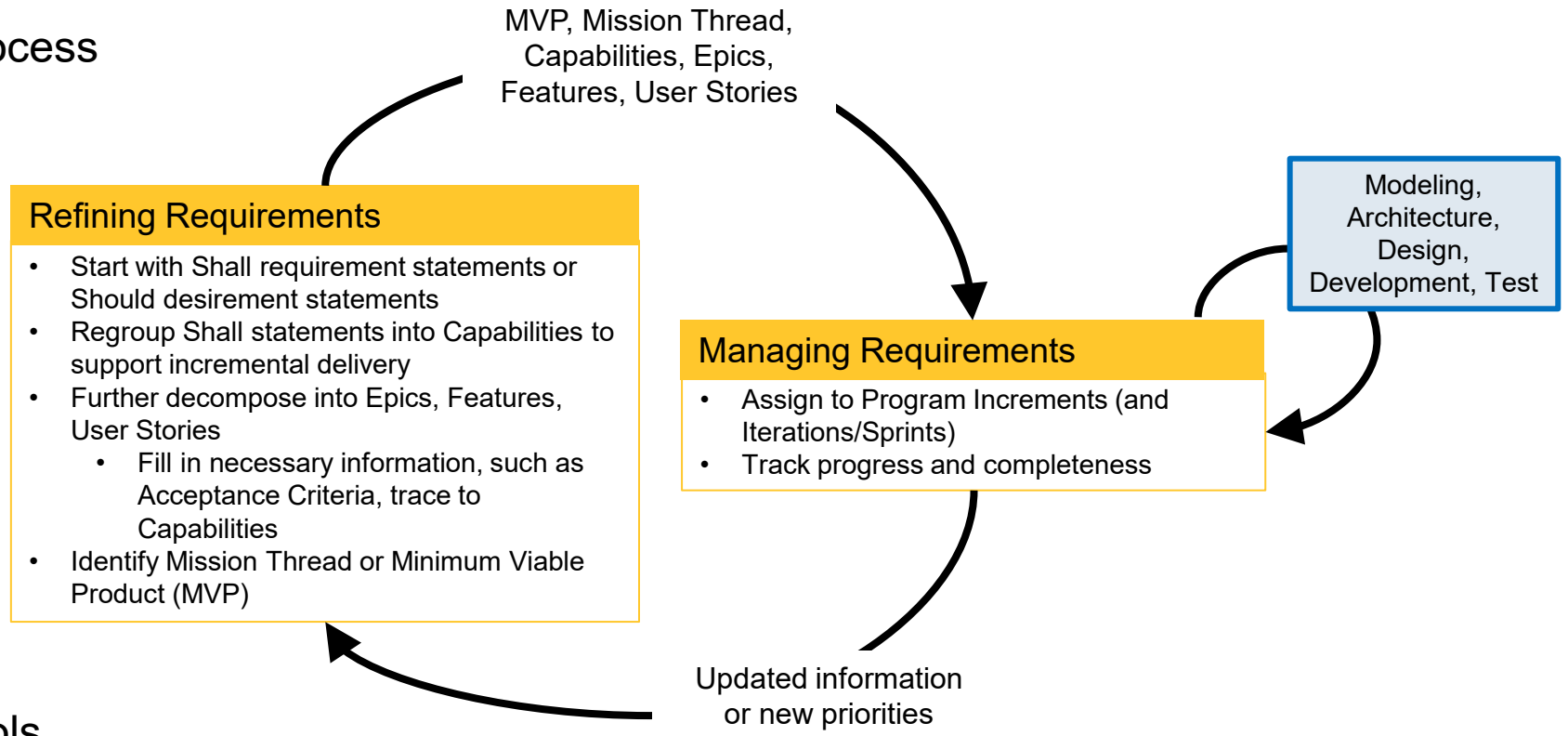
Definition of Done – Sample

User Story Definition of Done

- Architecture and detailed design reviewed
- Coding completed: secure coding standard, code commented, CM check in, peer reviewed
- Unit and subsystem test scripts coded, peer reviewed, and included in integration & test environment.
- End-user documentation updated
- Non-functional requirements (NFR) met
- Testing passed (automated): unit, cumulative, integration, regression, DISA Core Data Centers (CDC), etc.
- Each failure detection and recovery scenario identified in the User Story has been successfully tested and accepted by the Product Owner
- All known defects documented for Program Office review
- Sprint artifacts in IDE updated
- User Story satisfies Acceptance Criteria in the STP
- User Story accepted by Product Owner

Translating Requirements into Capabilities

- Process



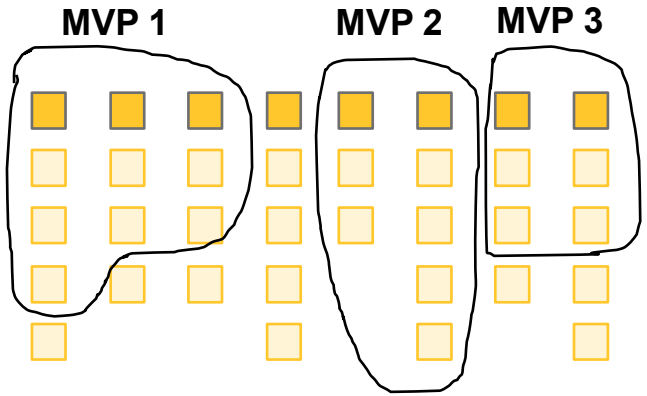
- Tools
  - *Requirements repository (DNG: DOORS Next Generation)*
  - *Model repository (MagicDraw, SparxEA)*
  - *Project Tracking (Jira, VersionOne, Rally)*

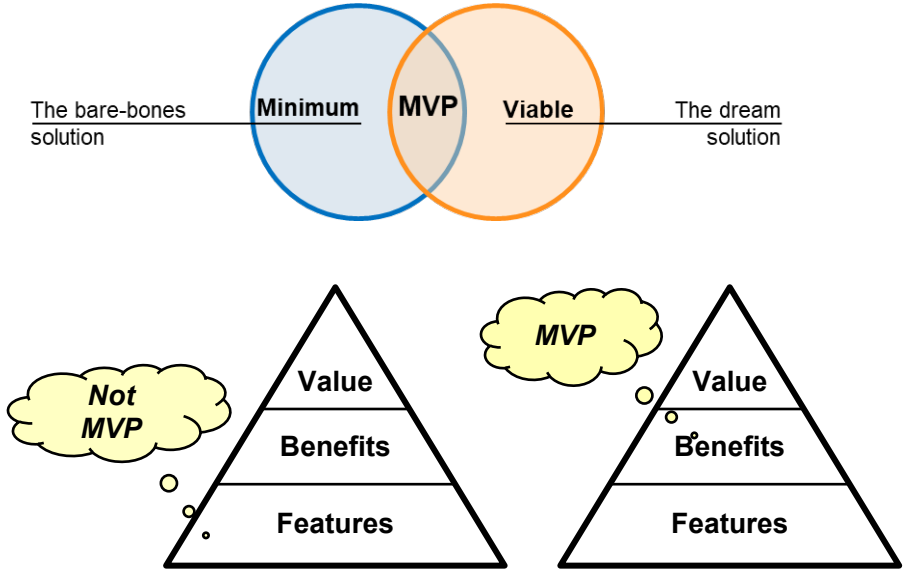
Organize Multi-Disciplinary Teams around Capabilities

Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Minimum Viable Product (MVP)

- Identify key system capabilities
- Decompose each system capability into a series of functionality, in increasing fidelity, to support incremental delivery of working software for each Sprint/Iteration
- Update and prioritize MVPs based on program requirements and objectives
- Enables frequent user feedback





Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Non-Functional Requirements

- Typically, requirements are captured in Features and decomposed into User Stories and do not address any NFRs
- Three methods for capturing
  - User Stories – each NFR captured as a separate User Story and addressed in a single Sprint/Iteration
    - Works well for specific performance requirements
    - May be harder to cover quality attributes that affect all User Stories
  - Acceptance Criteria – include NFRs as part of the Acceptance Criteria for affected User Stories
  - Covers all NFRs but requires the Agile team to catch all the potential NFR issues
  - Explicit List – specific list that contains all NFRs
    - Separate from the list of Features and User Stories
    - The Unified Process calls this a supplementary specification
    - Gives Agile teams a documented list of what to include as part of the Acceptance Test for each User Story

[Source: https://disciplinedagiledelivery.com/strategies-for-implementing-non-functional-requirements/]

Government Role for Agile Requirements

- Requirement structure and Agile terminology to use
- Ownership and/or collaboration in building the Product Roadmap
- Collaboration with contractor in Backlog Refinement (definition, prioritization)
- Collaboration with stakeholders in defining MVP
- Acceptance and sign-off of an Agile requirement
  - Who's responsible?
  - When is it done?
  - How is it done?
- Product Owner and Product Manager roles
  - Who fulfills?
    - If Government is the Product Owner or Product Manager
      - prepare for close collaboration
      - deep understanding of product
      - authority to provide quick direction and feedback

Requirements Management

Common Mistakes

- Inadequate Backlog Refinement
  - Impacts Sprint Planning efficiency as Backlog Items are not “ready”
- To many items or items too old in Backlog
  - Clutter and difficult to prioritize
- Review/estimate everything in the Backlog (and too early)
  - Unnecessary effort by team
- Too much information or Acceptance Criteria
  - Little room to innovate; team less engaged if everything is spelled out
- Assign an NFR to complete in one Sprint/Iteration
  - Example: Impossible to complete security requirements in one Sprint/Iteration
- Prioritization by proxy – priorities dictated by someone other than Product Owner
  - No accountability of Product Owner; unclear and confused multiple sources of truth

Requirements Management (continued)

Common Mistakes

- Assign an NFR to complete in one Sprint
  - Example: Impossible to complete security requirements in one Sprint
  - Should bake in -ilities or NFRs from the beginning
- Prioritization by proxy – priorities dictated by someone else (external to Product Owner)
  - No accountability of Product Owner; unclear and confused multiple sources of truth
  - Should appoint a Product Owner with authority
  - With multiple Product Owners, ensure that they speak with one voice
- Fail to organize and prioritize the Backlog by Feature, Backlog Items not decomposed from Themes or Epics
  - Backlog items not systematically organized or decomposed may lead to completed components that are not operational or deliver mission value

Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles
- Framework Roles and Responsibilities
- Requirements Definition
- Testing
- Risk Management
- Scaling Agile
- Measuring
- Lessons Learned on Ground Systems

Agile Testing vs Traditional Testing

- Agile shifts testing to the left
  - Traditional approach: System-level integration & testing done at later stage
    - Requires test community participation late in the program
  - Agile approach: System-level integration & testing done at every Sprint/Iteration
    - Requires high level of participation and commitment from test community across the program
  - DevSecOps approach: System-level integration & testing performed continuously
    - Requires dashboard, tools, and people to perform continuous monitoring
- Agile and DevSecOps test tools and equipment
  - Requires simulators and test beds much earlier
  - Requires significant amount of automation to support continuous testing

Agile Testing vs Traditional Testing

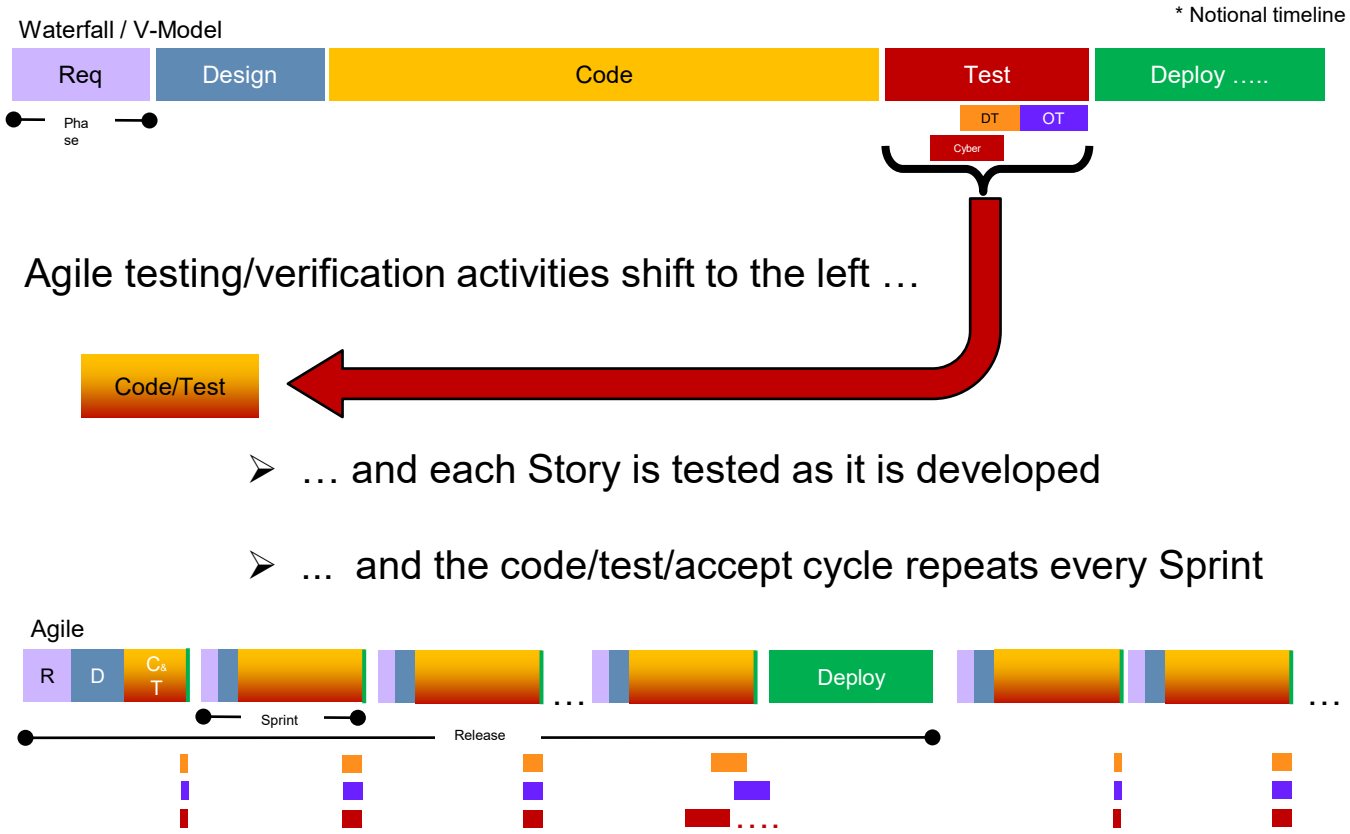
- Silo vs cross-discipline team
  - Traditional approach: separate test and evaluation team
  - Agile approach: developers write unit-level tests and testers/QA are part of every development team
- Test plan and cases
  - Traditional approach: detailed plan and thorough cases
  - Agile and DevSecOps approaches: small, repeatable, automated, incremental

What Happens with Verification & Validation (V&V)?

- Goal: no large testing events at the end of the project
  - Done at the team/Sprint level if possible
  - Government needs to be ready to support at this low level
- Potentially larger staffing footprint to support
- Allows for early / faster delivery of value to the end user
- Leverage Sprint and System Demonstrations (Demos) as opportunities for sell off
- Specify V&V items in the Acceptance Criteria and Definitions of Done
- Automate as much as possible
  - Government involvement needed on automation building
  - Ensures continued regression testing after V&V

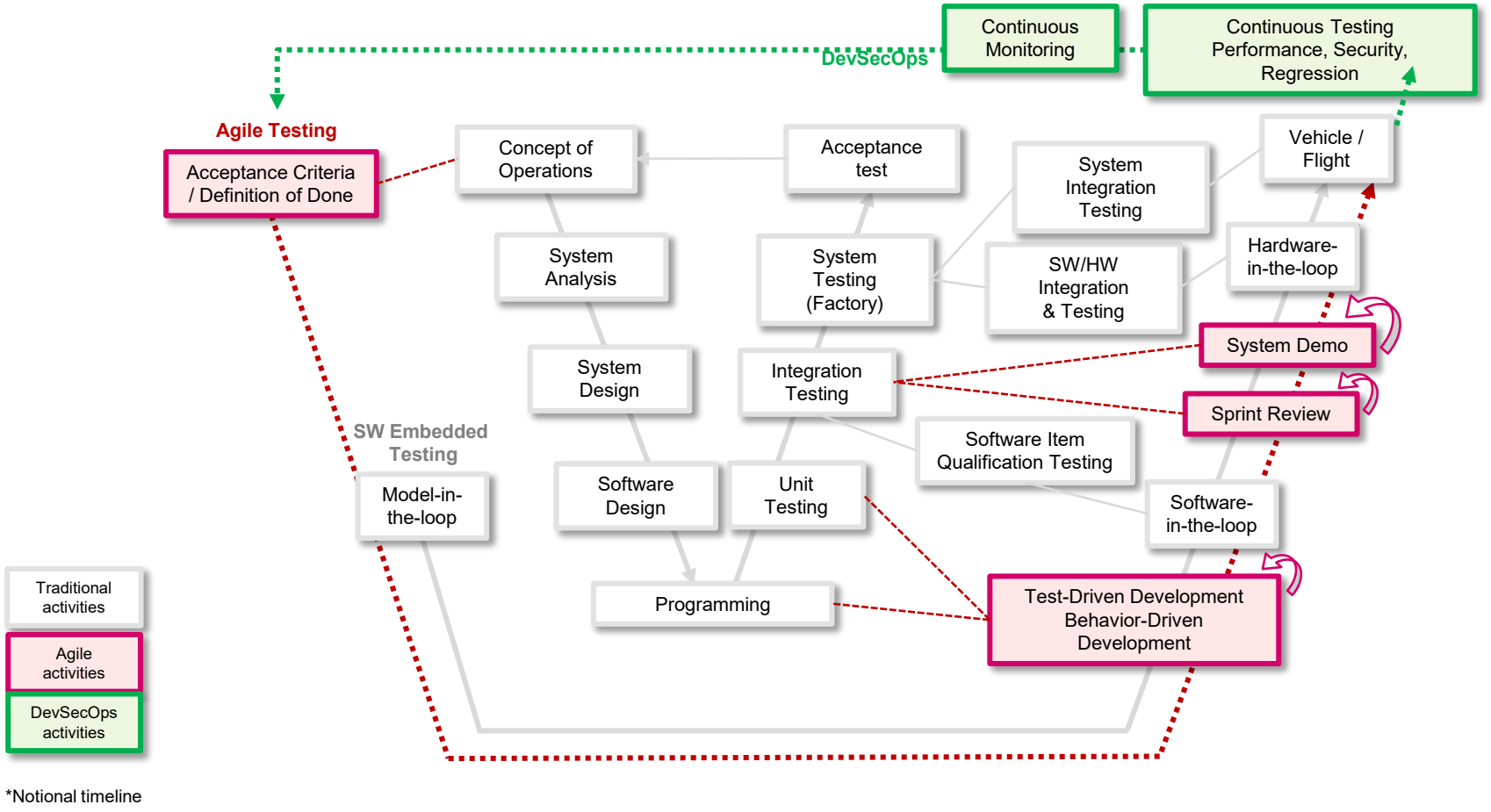
Process Comparison

Traditional vs Agile Testing



Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Software Testing Activities in Various Phases and Stages



Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

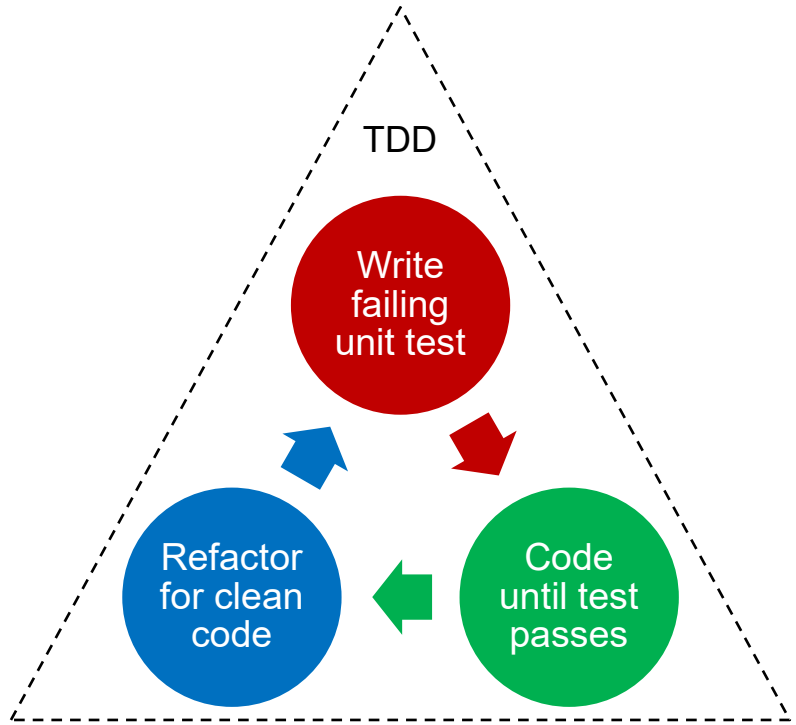
Recommended Practices for Agile Testing

- Ensure a tester is part of the development team
  - Encourage multi-disciplinary team to pursue end-to-end solution instead of silo of excellence
- Prioritize up front to set up Software Factory and automated testing infrastructure
  - Spend early Program Increments / Sprints to organize continuous testing practices and monitoring measurements
- Bake in NFRs or define as part of Acceptance Criteria or Definition of Done
- Use automated testing where possible to support rapid and continuous development
- Use test-first or test-driven development
- End-to-end testing and system-level testing continuously
  - At minimum, every build, ideally every check-in
- Leverage events where all stakeholders are present for sell-off
  - System Demos
  - Inspect and Adapt

Agile Testing Approaches

- Test-first development is a key practice in many Agile methods
  - **Tests are written in advance of development**
    - Essentially represent a specification of the system
    - Once passed, developers can spend time cleaning up code via refactoring

- **Test-Driven Development (TDD)** is core to Extreme Programming (XP)



Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Testing in Agile Teams

- Developers write and execute tests, configure tools for performance and other “-ilities” testing
- Agile teams include QA resources to support testing activities
  - Prepare and execute test plans
  - Create/code automated tests
  - Perform preliminary functional verification
  - Support from Users/Operators, Cyber security, Human System Integration (HSI)
- Product Owners support Feature Acceptance testing
- Users/Operators and stakeholders may be involved with
  - Usability testing
  - Alpha/beta testing
  - Exploratory testing

Modern Software Testing Trends

Leverage Commercial Software Testing Techniques

- **Continuous Testing**
  - *DevSecOps, Test-Driven Development (TDD), Behavior-Driven Development (BDD)*
  - *Automate as much as possible*

- **Artificial Intelligence / Machine Learning in Testing**
  - *Predictive analytics - when responses adapt*

- **Industrial Internet of Things (IIoT) Testing**
  - *Usability, Compatibility, Reliability & Scalability, Data Integrity, Security, Performance*

- **Chaos Engineering:** Chaos Monkey and Simian Army
  - *A discipline to perform experimental testing in a production(-like) environment*

- **Stage Testing and Selective Deployment**
  - *A/B Testing, Blue/Green Deployment, Feature Toggle*
  - *Canary Release, Dark Launches*

Agile Configuration Management

- Every member of the team must do code check-ins on a daily basis
- These check ins must be free from bugs in order to compile
- Developer's code must always be current from the repository
- A designated CM should be able to run automated nightly build tests to make sure all checked in code works and is bug free
- This is done with continuous integration server applications like Hudson, Jenkins or Rational Test
- Integration of the dev code into the trunk requires the CM or a dev to check out an instance of the trunk and merge their code into the main branch
- Developer goes home only when his code is bug free

Bad Practices in Agile Testing

- Accepting manual testing as suitable and effective
  - *Recommendation*
    - Design for test
    - Use Test-Driven Development at the unit level
    - Use Behavior-Driven Development at the integration level
    - Continue to enhance automated functional tests throughout the lifecycle
- Accepting automated unit testing covers everything
  - *Recommendation*
    - Need a good balance between automated and manual testing
    - Need to ensure automation is reviewed by government reps if to be used for V&V or sell off
    - Unit tests generally do not cover integration or interface testing cases
- Not integrating Test with Development or having large monolithic test events
  - *Recommendation*
    - Test as you go, continual testing, automated regression testing

Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles
- Framework Roles and Responsibilities
- Requirements Definition
- Testing
- Risk Management
- Scaling Agile
- Measuring
- Lessons Learned on Ground Systems

Agile Risk Management

Traditional planning does risk management upfront; Agile looks for risk throughout the lifecycle

- Risk Management in Agile projects differs from traditional risk management due to shorter development cycles
  - *Risk is examined and updated at each stage of the lifecycle, not just a periodic Risk Management update*
  - *Risk mitigation and ownership are determined by teams*

- Opportunities may be identified as part of Risk Management process

The Agile process itself has built many features into the development process to continually manage risk

Risk Management Cycle

- Four steps in Risk Management Cycle
  1. *Risk Identification*
     - Team identifies and analyzes project risks
       - *Program Increment Planning, daily Scrum, Scrum of Scrum, Retrospectives, etc.*
     - Team updates Product Backlog based on priority
  2. *Risk Assessment (functional/non-functional)*
     - Analysis of risk exposure based on probability and impact to operations/schedule
  3. *Risk Response*
     - Important to have a strategy to respond to risks that are identified
  4. *Risk Review*
     - Teams review and re-assess

Risk Identification Stages

The Risk Management Process is repeated at every level

- Risk identification can happen at any point in the life cycle
  - ***Program Increment Planning***
    - Discuss risks at program level
  - ***Iteration / Sprint Planning***
    - Team identifies, assesses, and discusses risks and mitigation steps
  - ***Daily Scrum Meeting***
    - Team identifies / reviews risks and issues
    - Risks are added to the risk board and agreed upon response plans documented
  - ***Iteration / Sprint Review***
    - Team discusses risks
    - Successful mitigation activities are highlighted and shared with stakeholders
  - ***Retrospective***
    - Team discusses risks successfully handled/implemented, chances of reoccurrence and what will be done differently for risk mitigation in subsequent Iteration/Sprint
  - ***Product (Sprint) Demos***
    - Constant exposure to the customers/users minimizes risk build up
- Risks are available to all at any time through the Agile Lifecycle Management (ALM) Tool
  - *Do not need to wait for a board meeting to review/update risks*
  - *Should link/trace to user stories / features / capabilities*

[Source: Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn)) Kenneth S. Rubin]

Risk Management Across Levels

From Development Team to Program to Stakeholder

- At Program level
  - *During Program Increment (PI) planning, with all team members and stakeholders, discuss, categorize and quantify risks*
  - *Keep track of risk in risk backlog; use tools (e.g., JIRA, Version One, Rally) to link/trace to user stories / features / capabilities*
  - *Follow up risk during Scrum of Scrum (or equivalent) meeting*
- At Team level
  - *Discuss risk at Sprint Planning, Sprint Review, Backlog grooming, Daily Scrum, Sprint Demo*
  - *Maintain risk backlog*
  - *Use spike story to address and explore uncertainty*
- At Stakeholder (Government team) level
  - *Participate during PI Planning to discuss and re-prioritize*
  - *Have access to risk backlog and ensure that high risk items are handled properly*
  - *Request for risk-related metrics to track risk status*
  - *Help mitigate the risks*

Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles
- Framework Roles and Responsibilities
- Requirements Definition
- Testing
- Measuring
- Scaling Agile
- Risk Management
- Lessons Learned on Ground Systems

Agile for Large Scale Programs

- Commonly used frameworks for large scale programs
  - *Scale Agile Framework (SAFe)*
    - https://www.scaledagileframework.com/#
    - *Used by over half of respondents
  - *Scrum@Scale*
    - *Used by ¼ of respondents
  - *Large Scale Scrum (LeSS)*
    - https://less.works/

Based on 2022 State of Agile Report

Additional Practices for Large Scale Programs

- Big and Small timeboxes
  - *Sprint / Iteration – 1 to 4 weeks*
    - Release a potentially shippable product by the end of each Sprint
    - Focus on work within the team
    - Activities: Sprint Planning, Sprint Demo, Sprint Retrospective
  - *Program Increment – quarterly*
    - Release a system-level product by the end of each program increment
    - Focus on coordination between teams
    - Activities: Program Increment Planning, System Demo, Inspect and Adapt, Scrum of Scrums, Product Owner sync
- Additional roles
  - *Program level*
    - Release Train Engineer, Product Manager, System Architect / Engineer

Additional Practices for Large Scale Programs

- Architectural Runway
  - *Proactively plan and develop necessary technical foundation and infrastructure needed for developers to implement near-term features*
  - *Avoid big design upfront, design one step ahead of development team*
    - The bigger the aircraft (teams), the more runway is needed
  - *Sample architectural runway activities*
    - System Engineering team develops overall architecture or design ahead of the development team
    - Infrastructure team performs trade study on COTS selection
    - Tech Support team perform infrastructure upgrades
- Program-level planning or PI Planning
  - *Team members from all teams attend a face-to-face planning event at the beginning of each program increment (PI)*
  - *Plan what to deliver in the upcoming PI timeframe (e.g., quarter)*
  - *Stakeholders, Customers, Operators should also attend planning*

Additional Practices for Large Scale Programs

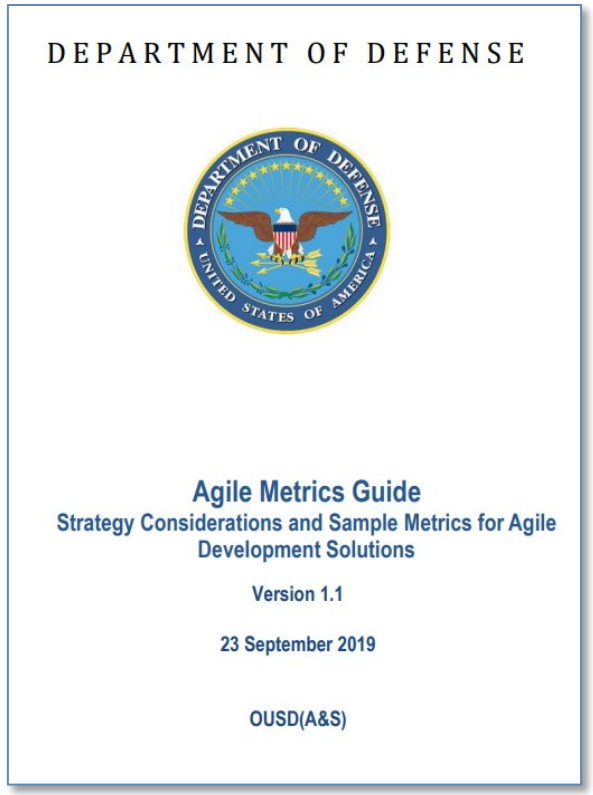
- System demo
  - *System-level demonstration; ideally at the end of each Sprint*
  - *Demoed product is fully integrated across all teams*
- Innovation and Planning (IP) or Hardening, Innovation and Planning (HIP) Sprint
  - *The last Sprint of each PI is dedicated for innovation, inspect and adapt, training, preparing for System Demo, buffer for addressing technical debt*
  - *PI Planning preparation occurs during the IP/HIP, with the PI Planning ceremony at the very end of the IP/HIP*
- Bigger than program level?
  - *Solution level – multiple programs or including supplier teams*
  - *Portfolio level – multiple product lines, multiple solutions*

Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles
- Framework Roles and Responsibilities
- Requirements Definition
- Testing
- Risk Management
- Scaling Agile
- Measuring
- Lessons Learned on Ground Systems

DoD Agile Metrics Guide



DEPARTMENT OF DEFENSE

Agile Metrics Guide

Strategy Considerations and Sample Metrics for Agile Development Solutions

Version 1.1

23 September 2019

OUSD(A&S)

→ Recommended Metrics

Ref: https://aerospacecloud.sharepoint.com/sites/Communities01/AgileSoftwareSystemsCoI/Documents/Forms/undefined

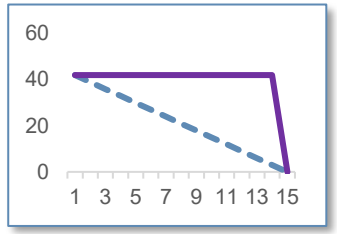
Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Minimum Essential Agile Metric - Sprint Burn Down

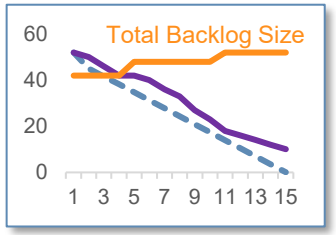
- Potential dependency, bottleneck
- Why a story is “undone”?
- No progress / did not update the tool
- Unfinished stories, need to carry over

- Ahead of schedule

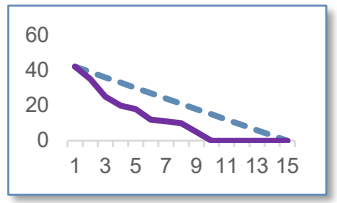
- Amount of work planned for a Sprint
- Track progress and potential impediments
- Burndown can also be used to track a Release
- Once a Product Owner accepts, Dev claims Done



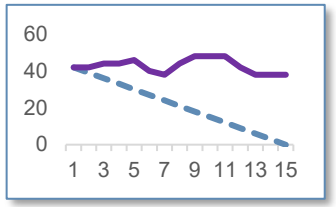
Big impediment or
ignore the tool;
Watch out for quality



New Stories added in the
middle of the Sprint



Overestimated or
Not enough stories in the backlog;
If done early, add stories from backlog



We need to talk !!
Failed significantly
Need training

Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Additional Useful Team-Level Metrics

The following metrics are mainly for the development team, but they are useful for programs to understand

Functionality

- # Stories (loaded at beginning of Sprint)
- # Accepted Stories (defined, built, tested, and accepted)
- % Accepted
- # Not accepted (not achieved within the iteration)
- # Not accepted : deferred to later date
- # Not accepted : deleted from backlog
- # Added (during iteration; should be 0)

Quality and Test Automation

- 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

Source: https://www.scaledagileframework.com/

Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Quality Assurance Measures

Category	Performance Item	Measure	Performance Level Standard
Productivity	Sprint Results	Story Points accepted divided by Story Points planned	80% Minimum 90% Average
	Release Results	Feature is fully completed	90% of all Features
Predictability	Sprint Results	% Variance of planned vs. actual	Between -10% variance and +15%
	Sprint Results	Average number of User Stories are in the Team Backlog	
	Release Results	Consistency of Velocity through the release, Consistency of stories accepted through the release	
Completeness	Integration Test Coverage	Percentage of testable User Stories with integration test cases and test results	100%
	Unit Test Coverage	Percentage of code covered by unit test	85% Minimum
	Automated Test Coverage	% of automated test coverage	Report for +/- trends
Qualitative Analysis	Testing	Do Test Cases test appropriate functionality	100% of the time
Findings	Security	Number of critical security findings	<2%
Quality	Code	SONAR SQALE rating	Level A 90% of time
	Code	Number of critical issues in SONAR	All are addressed by the next Sprint
	Code	Number of critical issues in SONAR per 1,000 lines of code	2
	Code	Number of defects per 1,000 lines of code	5 Average
	Code	Number of defects found in each Sprint (Defect Density)	<2%
	Data	Several measures	90% of data ingestions satisfy the SLA
Timeliness	Application Performance	Several measures	Satisfy the SLA
Qualitative Analysis	Agile Compliance	Analysis on how well the Team is adhering to Agile standards and Team Process Agreements	100% of the time

[Source: https://confluence.di2e.net/display/HBBAGILE1/Quality+Assurance+Measures]

Agenda

Agile Approaches for Ground Systems

- The Agile Manifesto and its Principles
- Framework Roles and Responsibilities
- Requirements Definition
- Testing
- Risk Management
- Scaling Agile
- Measuring
- Lessons Learned on Ground Systems

Anonymous Examples from "Agile-But" Projects

Key Agile principles and practices are complementary and necessary

- Make incremental deliveries, BUT… at *annual* intervals
- Use small teams and short iterations, BUT...
  - *Pre-plan the content of all iterations ahead of time*
  - *Push undone work to the next iteration without replanning & reprioritizing*
  - *Don't deliver (or even consider) working software at each iteration*
  - *Make the teams only superficially cooperative*
- Engage the customer and end-users, BUT...
  - *Do so in formal, large-scale, and scripted events, not one-on-one*
  - *Have great access to, but little cooperation from stakeholders*

Observed on projects called Agile

Evidence a Project Might Not Really Be Agile

General Rule: attributes should be consistent with Agile principles

- **Lack of automated testing:** the rapid pace of Agile projects virtually demands automation; manual testing just can't keep up
- **Limited stakeholder involvement:** limited long-term planning in Agile requires rapid, ongoing stakeholder feedback to guide process
- **Team members are assigned work:** Agile emphasizes self-organization; assignments limit adaptability and flexibility
- **Lack of refactoring:** code rots, and emergent design takes wrong turns; plan to revise design and clean up code periodically
- **Large teams:** Agile teams top out at about 10 persons; larger projects need cooperating teams, but watch for interdependencies
- **Long iterations:** insight and feedback are harder, risk is higher, complexity is greater; should be no longer than 1-4 weeks

A far from exhaustive list of examples

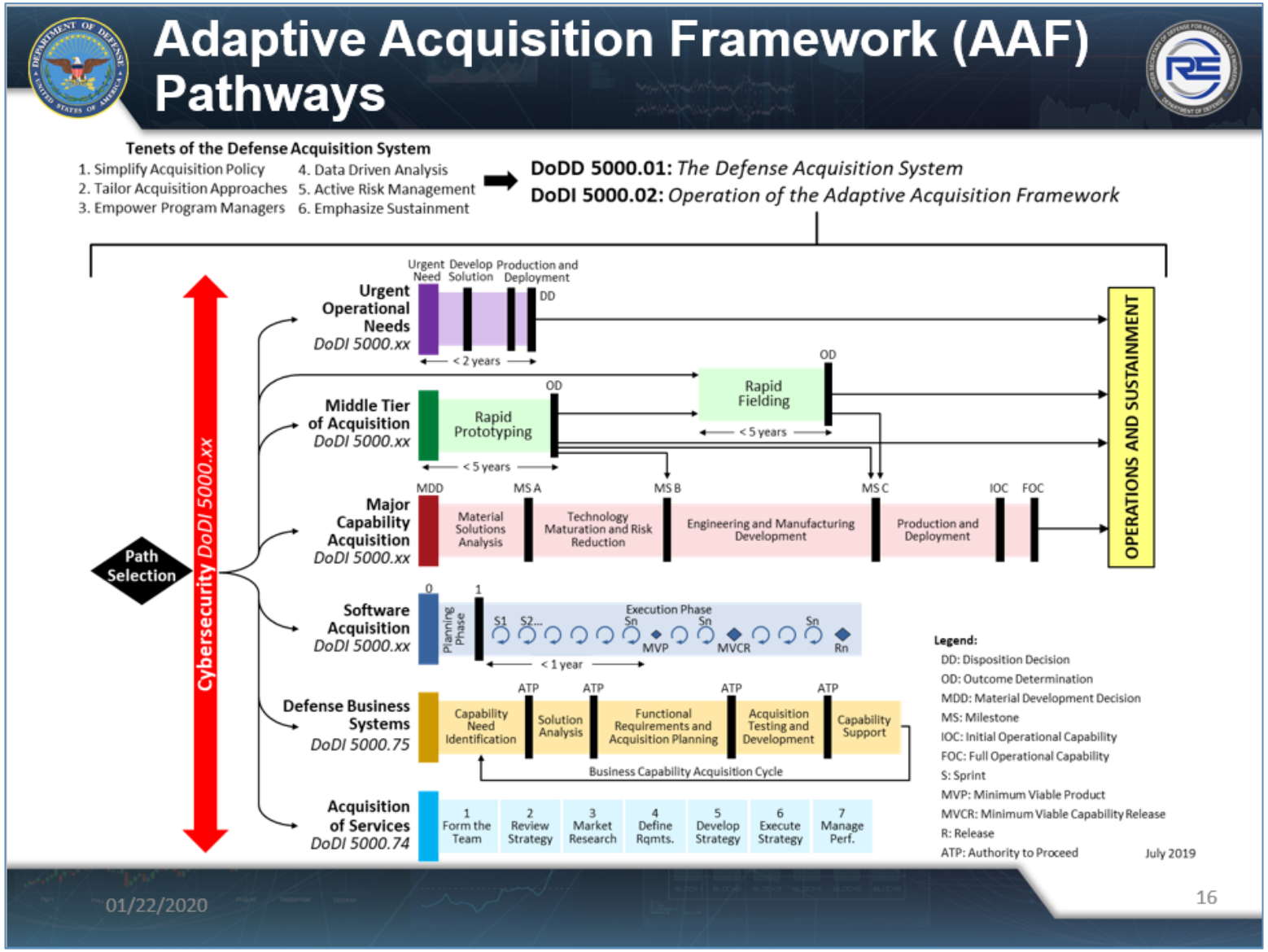
Common Challenges when Adopting Agile Methods

- **Maintaining a constant and rapid pace:** requires the streamlining of decisions and processes, the commitment of all stakeholders
- **Involving stakeholders:** customers and developers working together daily throughout the project may be a difficult adjustment
- **Supporting ongoing communication:** face-to-face interactions with external organizations can be difficult to maintain
- **Avoiding scope creep:** a reasonable concern of acquirers when users are involved; program office must be highly involved to mediate planning
- **Integrating hardware and external/internal providers:** software development may outpace hardware capabilities and internal processes (testing, systems engineering); simulation, automated testing, “mocking” interfaces may help

It's All About the Feedback

- Agile methods require frequent, rapid feedback from stakeholders
  - *End-users should be ready to offer feedback every Sprint/Iteration, maybe daily*
  - *Program Office and end-users choose and prioritize next Sprint/Iteration's workload*
- Quality feedback is (in some sense) the substitute for detailed planning: you only plan in detail for the reach of your headlights
- Expect to review incremental deliveries very frequently, including both "potentially-deliverable" software and documents
- Expect to help plan the next iteration every cycle, including establishing priorities and possibly deferring requirements
  - *Program Office must mediate end-user expectations and priorities when planning*
  - *Program Office must be the bridge between the acquisition and technical worlds*
- Impossible to maintain the pace without heavily-automated testing

Recent Direction for Government Acquisitions



[Source: S. Standard, Cybersecurity T&E Guidebook v2.0 Change 1 Overview and Cloud Addendum, SMC TEWG, 22 January 2020]

Ref: OTR 2023-00353: Agile Approaches for Ground Systems, Sasine 2023

Recent Direction for Government Acquisitions

Software Acquisition

- Mandate the use of Agile methodologies including
  - *RFPs – using user stories*
    - Instead of pre-defined unverified technology claims or expectation
  - *Established success metrics based on the targeted end-goals*
  - *Allow for rapid change of scope based on learnings contingent to meeting end goals*
- Incentivize the use of
  - *Microservices / Smaller code*
  - *Containers, Container management solution – to allow for scale and rapid deployment*
- Leverage
  - *DevSecOps solution*
- Perform continuous monitoring
  - *No need for phases and milestones*
  - *Government checks on Product backlog*
  - *Continuously delivered product in testing/staging environment or production*
  - *Real-time telemetry (system dashboard)*

[Source: N. Chaillan, DoD Enterprise DevSecOps Initiative (Software Factory)]

Recent Direction for Government Acquisitions

Software Testing

- Contract language considerations added to address
  - *Software development test practices*
  - *Testing system development environments*
  - *Software assurance testing*
  - *Flow down language from prime contractor to sub-contractors*
- Suggested language added to address
  - *Contractor-government integrated cybersecurity testing during system development*
  - *Contractor testing during prototype and rapid fielding development*
  - *Contractor remediation of system vulnerabilities discovered during contractor or government testing*

[Source: Lt Col Kenneth Thill, DevSecOps – Testing Software]

Recent Guidance for Government Acquisition

Software Testing

- Embedded Testers
  - *Have Government Developmental Testers (DT) embedded in contractor development environment; critical to moving fast*
  - *Ability to conduct functional level testing and requirement verification in near real-time and provide immediate feedback on bugs/issues to reduce tech debt and release time*
  - *Limitations*
    - Personnel
    - Funding
    - Security/Access
- Test and Evaluation (T&E) Management
  - *Program Requirements Management Team (PRMT) assign each capability or feature a mission criticality rating recommendation*
  - *Program Planning Meeting (PPM) review and recommendation to the Operational Acceptance (OA) authority (AFSPC A3/6) who has final approval authority*
  - *Once rating assigned, each capability will be tested and delivered based on the OA Categories and Capability Maturity matrix*
    - OA will be delegated to the appropriate command level within the unit receiving the capability

[Source: Lt Col Kenneth Thill, DevSecOps – Testing Software]

Q & A

[Source: Lt Col Kenneth Thill, DevSecOps – Testing Software]

Thank You

[Source: Lt Col Kenneth Thill, DevSecOps – Testing Software]