University of Houston Clear Lake

Applying Standard Independent Verification and Validation (IV&V) Techniques within an Agile Framework: Is there a Compatibility Issue?

James B. Dabney, UHCL

James D Arthur, VA Tech

IEEE/INCOSE SysCon 2017

VirginiaTech College of Engineering

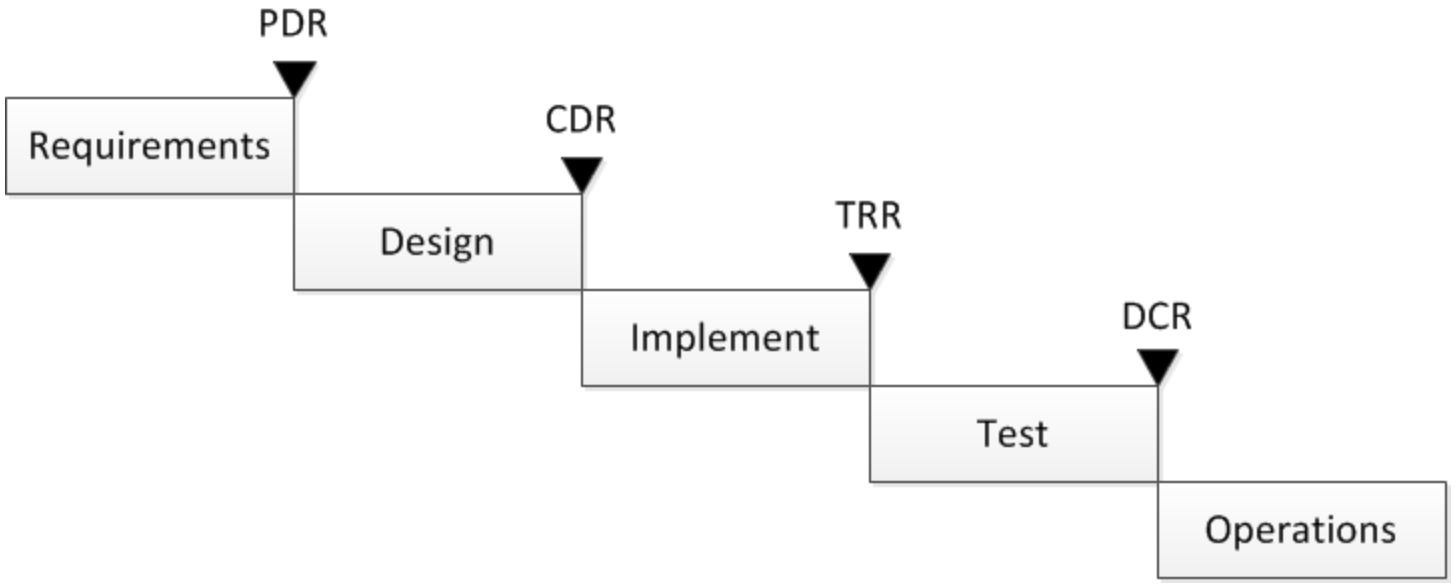
Overview

- Conventional mission-critical software lifecycle
- Conventional IV&V process
- Agile software development
- Hybrid Agile variants
- Adjusting IV&V to hybrid Agile
- Conclusions

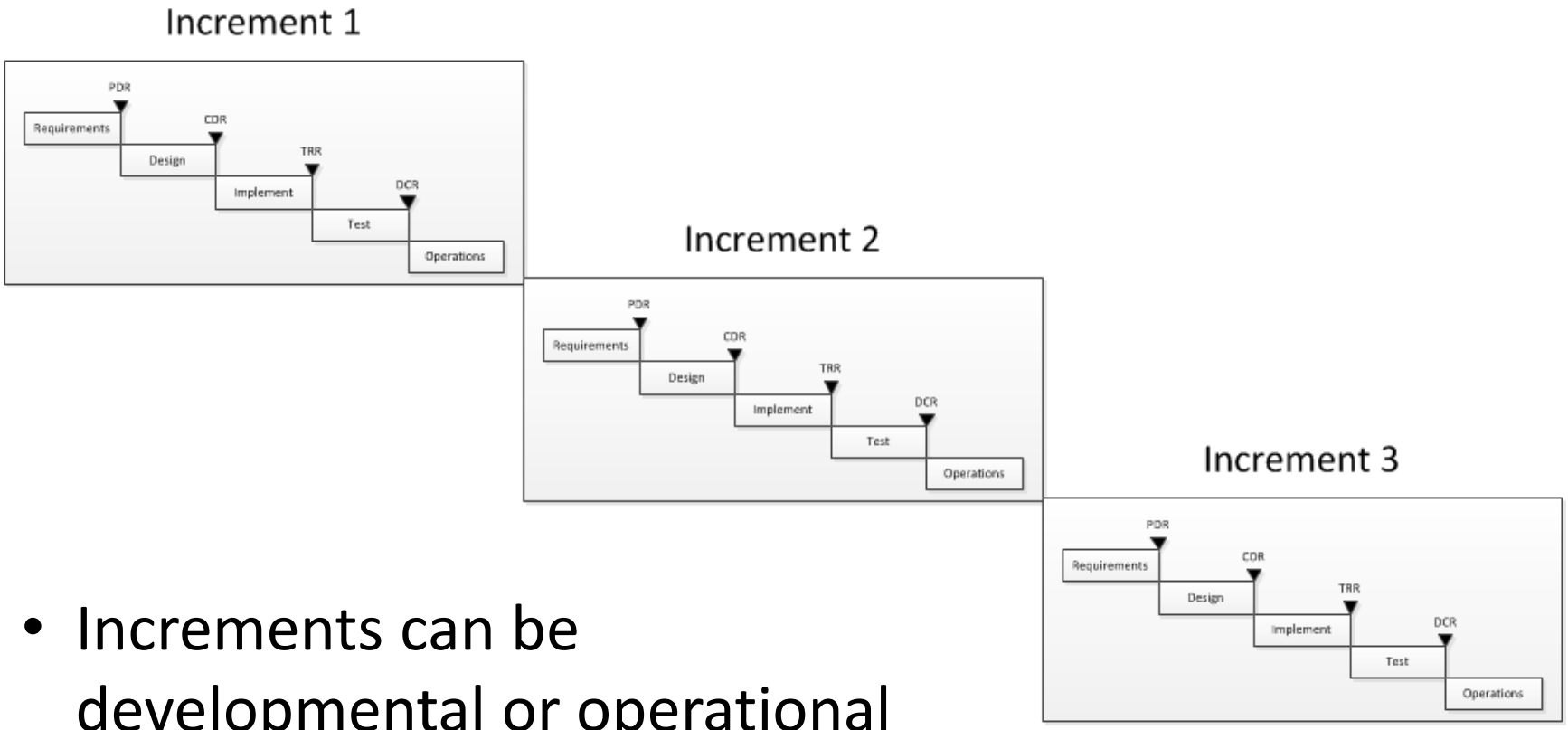
Conventional Mission-Critical Software Lifecycle

- Traditional lifecycle based on waterfall model
- Sequence of milestone reviews
  - Preliminary design review (PDR)
  - Critical design review (CDR)
  - Test readiness review (TRR)
  - Design certification review (DCR)
- Larger projects incremental model
  - Planned series of waterfall lifecycles
- Certification mandated by regulations (e.g. FDA, UL)

Example Traditional Waterfall Lifecycle



Example Incremental Lifecycle

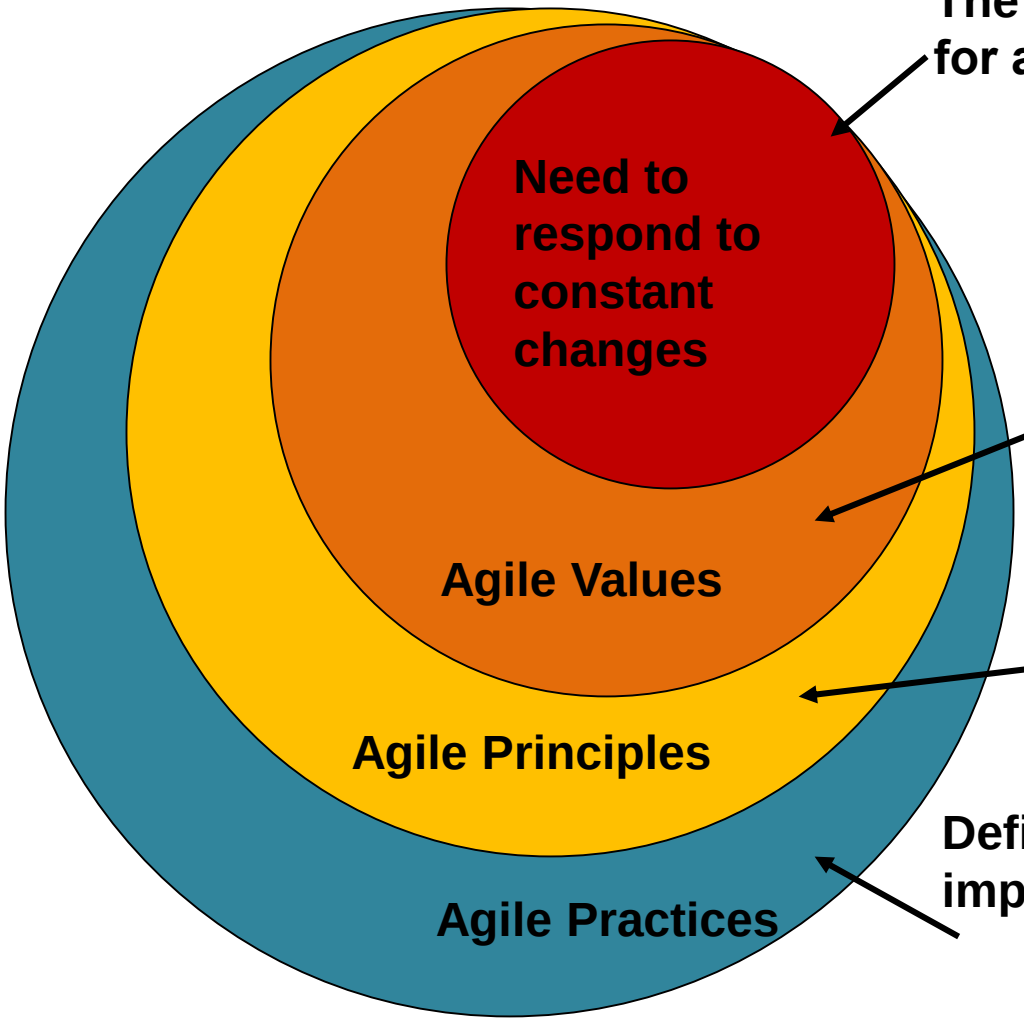


- Increments can be developmental or operational
- Plan several increments ahead

Conventional IV&V Process

- Reduce program risk by analyzing key artifacts
- Strive to find issues in-phase by mirroring development
- Verify during each lifecycle phase that the product satisfies requirements defined in previous phase
  - Requirements meet user needs, complete
  - No unintended functionality specified
  - Design satisfies requirements and no more
  - Testing fully covers design and requirements

Understanding Agile



The fundamental reason for a "new" paradigm

Defines the set of most important beliefs of what is truly important

Defines a set ways to meet the values

Defines in detail how this is implemented in practice

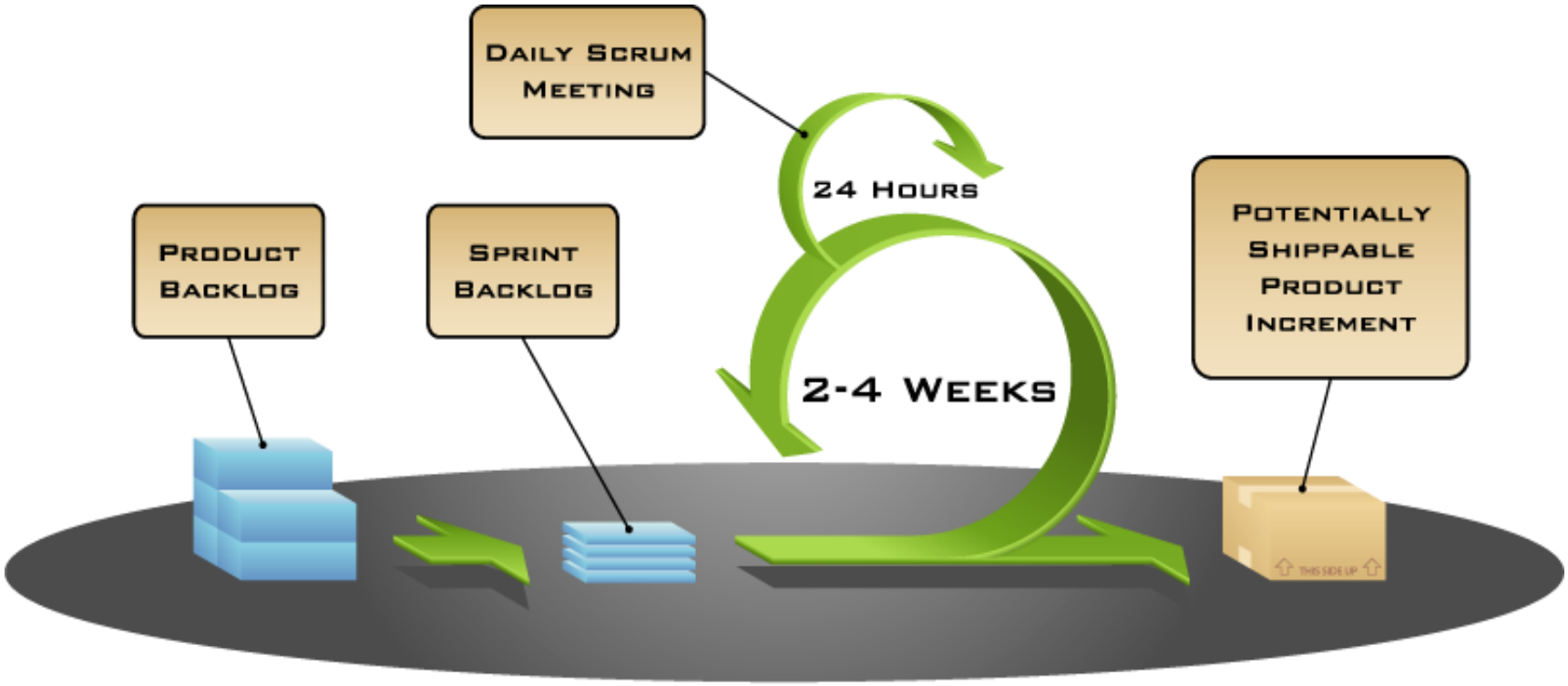
⑩Material adapted from "All about Agile", Ahmed Sidky, Presentation for CS 5704, Va Tech Fall 2006

Agile Manifesto [AM01]

Individuals and interactions	Over	Process and tools	Mission-Critical / IV&V Implication
Working software	Over	Comprehensive documentation	Docs required for IV&V, certification
Customer collaboration	Over	Contract negotiation	End product requirements defined at outset
Responding to change	Over	Following a plan	Change inevitable, must be managed

Agile Principles	Mission-Critical / IV&V Consideration
Customer satisfaction by rapid, continuous delivery of useful software	Often don't need working software until late in program
Working software is delivered frequently (weeks rather than months)	Frequent updates less important than technical rigor
Working software is the principal measure of progress	Safety / health of enterprise principal measure
Even **late changes in requirements** are welcomed	Late changes inevitable but can be costly
Close, daily cooperation between customer & developer	Often integrated product teams
Fact-to-face conversations is the best form of communication	Clear documentation essential due to long operational life
Projects are built around **motivated individuals**, who should be trusted	Some projects span careers, must be able to retain institutional knowledge
Continuous attention to **technical excellence** and good design	Clearly essential
Simplicity	Always desirable
Self-organizing teams	Multi-site, multi-contractor, large staff
Regular **adaption** to changing circumstances	Budgeting, staffing can have multi-year lead times

Agile Planning: The Scrum Process



Agile Planning: Release and Iteration Planning

Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Feature …

Feature 1, Feature 2, Feature 3a

Release Backlog	Iteration 1	Iteration 2	Iteration 3
Story A	Story A	Story C	Story F
Story B	Story B	Story D	Story G
Story C		Story E	
Story D			
Story …			

⑩Material adapted from "All about Agile", Ahmed Sidky, Presentation for CS 5704, Va Tech Fall 2006

Adapting Agile to Large Projects

- Alistair Cockburn (one of the original agile proponents): “small projects, web projects, exploratory projects, agile is fabulous; it beats the pants off of everything else, but for NASA, no” [AM13]

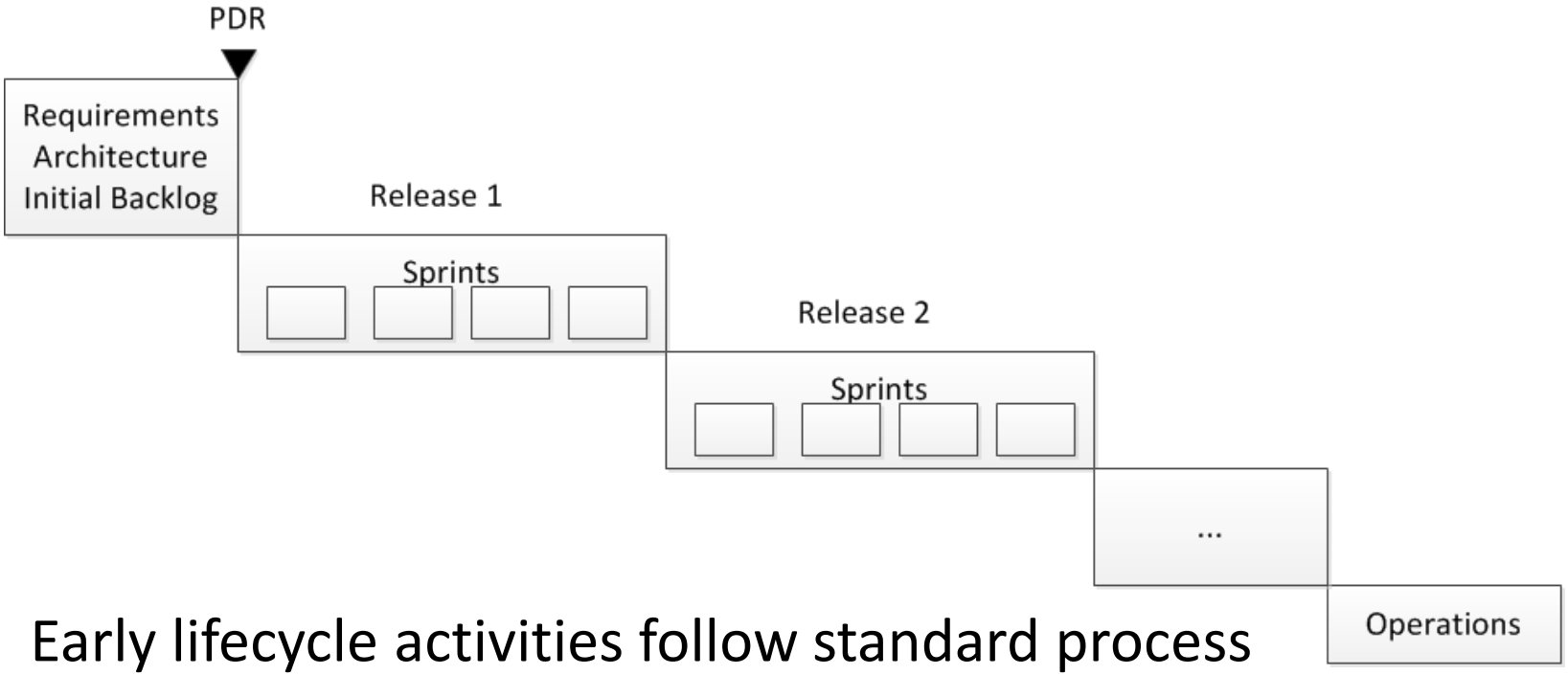
- “Embedded systems have specific product requirements, e.g. safety, which are not obviously addressed by agile practices such as XP or Scrum” [EOS14]
- Key assumptions of Agile (e.g.co-located teams) are difficult to realize on large projects [TFR02]

Variants of Agile for Large Projects

- Scaled Agile Framework (SAFe) Intended for high-assurance environments (medical)
  - Designed to comply with regulatory requirements (FDA)
  - Gaining acceptance
- Incremental Commitment Model (ICM) Merges concepts of classic V-verification, concurrent engineering, Agile
  - Intended for large mission-critical and net-centric systems

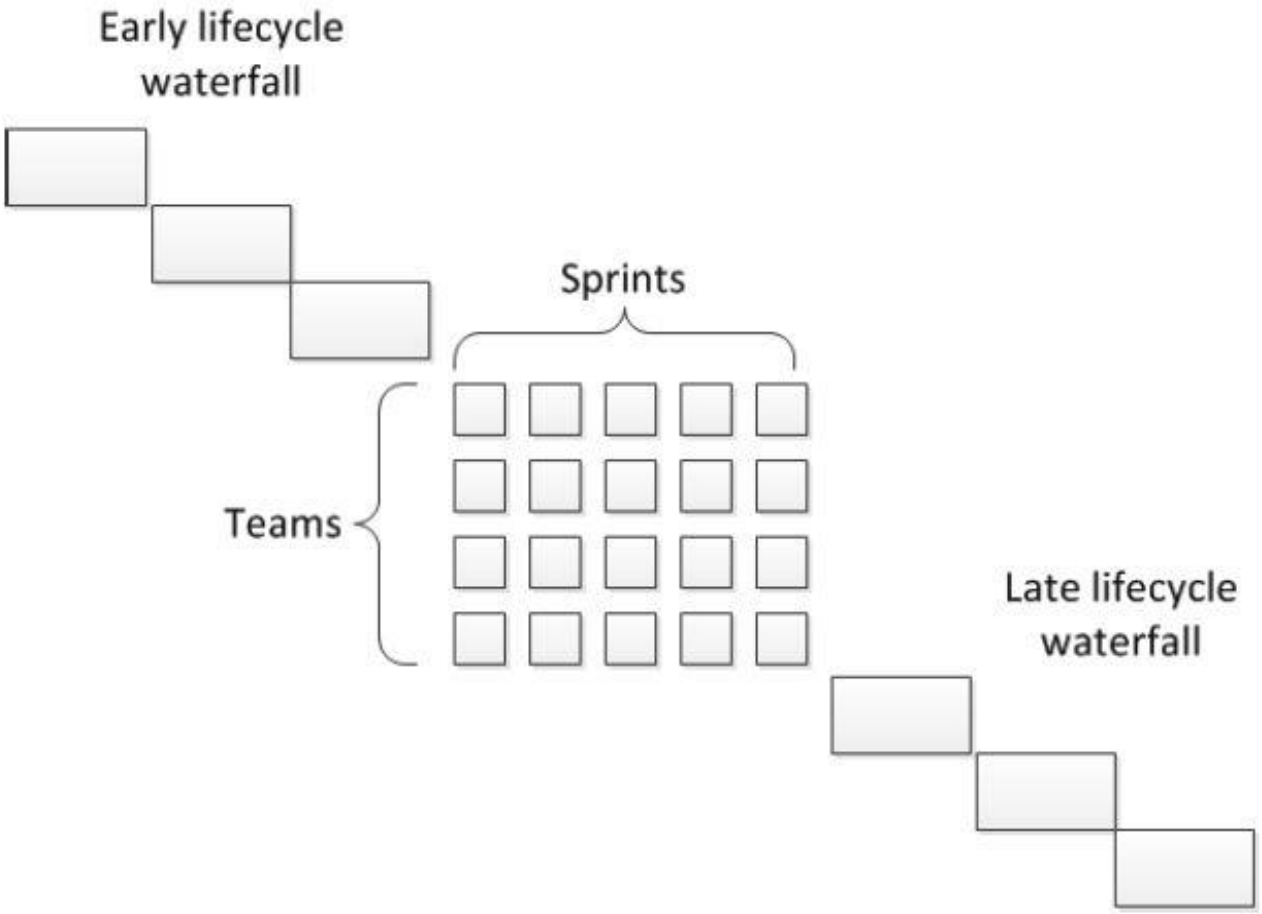
Hybrid Projects

- Similar to SAFe methodology



- Early lifecycle activities follow standard process
- Requirements, design, test follow Agile process
  - Sequence of releases composed of multiple sprints
  - Work down project backlog
- Certification follows standard process

Water-Scrum-Fall Process



Mapping Traditional V&V to Agile

- Assessed applicability of standard V&V methods to hybrid Agile
- For each method specified for project elements, assessed
  - Inputs
  - Timing in lifecycle
  - Feasibility of executing method given the timing and available information
- Methods fall into three classes
  - Early lifecycle methods generally compatible
  - Methods involving tracing need to be tailored
  - Methods involving completeness need to be replaced

Methods Requiring No Tailoring

Verify Implementation of Requirements or Design in Source Code or Scripts through Manual Inspection
Reuse applicability by comparing operational environments
Validate Safety Requirements by Inspection of Traces to Fault Trees and FMEA
Verify Software Behavior for Off-Nominal Conditions using Independent Testing
Validate Software Architecture by Inspecting Traces to Essential Properties
Verify Critical Software Changes By Inspecting Change Requests
Verify System/Software Architecture Using a Discrete Model of Performance Requirements in Stressing Scenarios
Assess Architecture Completeness by Inspection Against an Architectural Standard
Validate Feasibility Study Conclusions by Inspection
Validate Test Procedure by Inspection and Traces to Requirements
Validate Mission Project Operational Concepts by Generating Use Cases from Concept Documentation
Validate System Security Categorization and Regulatory Security Requirements by Inspection using Security Risk Management Framework (NIST-SP-800-37, Step 1)
Verify Security Control Selection and Threats/Risks Identification by Inspection using Security Risk Management Framework (NIST-SP-800-37, Step 2)

Methods Requiring Tailoring

Verify Software Code Quality using Static Analysis Tools
Validate Test Plan by Inspection
Validate Requirements by Inspecting Bidirectional Traces
Verify Test Execution by Inspection of Test Cases, Inputs and Results
Verify SW Interface Implementation by Inspection Against Interface Design
Verify Critical Software Changes By Inspecting Change Requests
Validate Test Cases by Inspection and Traces to Requirements
Verify Scripted Timeline Via Manual Multi-Directional Tracing
Verify Software Design by Inspecting Traces to Requirements and Software Architecture
Verify Software Capabilities through Independent Testing of Operational Scenarios

Methods Incompatible with Agile

Validate Interface Requirements by Inspection Against Component Interfaces
Validate Requirements by Inspecting Against Quality Criteria and System/Software Background Artifacts
Validate Test Design by Inspecting Traces from Scenarios
Verify Software Implementation by Inspecting Traces to Requirements
Verify Software Interface Design by Inspection Against Interface Requirements
Verify System Software Safety by Comparing Concept Documentation, Requirements, Testing, Design and Code with Hazard Analysis Documentation to Establish a Safety Case, Across the Software Development Life Cycle
Verify and Validate Requirement Implementation using Flow Diagrams to Uncover Missing, Conflicting, or Unnecessary Behavior

Interface Requirements Objectives

- Correlate integration requirements to specific interfaces and examine coverage to ensure that all interfaces are specified and all interface requirements relate to a necessary interface
- Correlate integration requirements to ensure they are required, incorrect behavior is prevented, unexpected inputs responded to appropriately.
- Verify interface requirements are correct, consistent, complete, accurate, verifiable, where

Interface Requirements Hybrid Agile Variant

- Capture the interface requirements as they emerge during each release
- As interfaces are defined to clear blocks, developer artifacts can be used to build a picture of the interface and refine requirements
- Interface map using tool
  - Starts with interface template or estimate
  - Incrementally capture interfaces and track properties
  - Track completeness and measures of risk burndown

Requirement Validation Objectives

- Ensure system requirements satisfy acquirer needs relative to system software
- Ensure software requirements meet system needs from functional and non-functional perspectives
- Ensure requirements for software interfaces are adequate in terms of operational environment, dependability, fault tolerance, and functional and non-functional perspectives
- Analysis steps address
  - Unambiguous
  - Verifiable
  - Consistent
  - Correct
  - Complete

Requirement Validation Agile Variant

- Unambiguous, verifiable, consistent compatible
- Correct, complete are challenge
- Potential solutions
  - Predictive model of requirements
  - Risk burndown model

Test Design Objectives

- Ensure test designs correctly specify a feature or combination of features
- Ensure the test environment is sufficiently complete, correct, and accurate
- Analysis steps
  - Develop a set of scenarios considering correctness and adverse conditions
  - Validate the scenarios with walk-through
  - Trace requirements to scenarios.
  - Trace scenarios to software structure
  - Trace the scenarios to test design and environment

Test Design Agile Variant

- Identify relevant scenarios
- Map requirements to scenarios as requirements emerge
- Potential solutions similar to Method 2
  - Predictive model of test design
  - Risk burndown model

Software Implementation Objectives

- Ensure that software components can reliably perform required capabilities under nominal and off-nominal conditions, perform no undesired behaviors, and that documentation is adequate to support maintenance
- Ensure that the code satisfies dependability and fault tolerance requirements, is capable of detecting identified hazards, and introduces no hazards.
- Ensure that all applicable requirements are implemented (for example, from SRS and IRS) and no unspecified behavior is introduced.
- To accomplish these objectives
  - Determine required nominal conditions from operations documentation and technical reference
  - Locate the source code relevant to the required functionality
  - Analyze implementation for completeness, correctness, behavior under unexpected conditions
  - Trace implementation to nominal and off-nominal scenarios
  - Analyze code in terms of fault tolerance and hazard response

Software Implementation Agile Variant

- Partial assessment at end of each release
- An analytical framework to track
  - Implemented functionality
  - Expected functionality
  - Recognize unexpected functionality
- Example techniques
  - Quality Function Deployment
  - Safety cases
  - IV&V reference models

Interface Design Objectives

- Ensure all relevant requirements represented in design documentation
- Find evidence that all relevant assurance goals are achieved for all interfaces with hardware, operators, other software functions, and other systems.
- To accomplish these objectives
  - Compare requirements and design documentation including analysis of algorithms, commanding, state/mode definitions, exception handling, error logging, configuration data, performance criteria (e. g. timing, latency, bandwidth), interface specifications (all layers)
  - Verify that interface requirements are unambiguous, complete, accurate, consistent, testable/verifiable, traceable
  - Verify interface flows are correct and consistent (sequences, flows, control, formats, standards)

Interface Design Agile Variant

- IV&V interface model
  - Captures information from each sprint
  - Builds an understanding of the as-built interfaces
  - Check for common interface errors
- An interface IV&V approach which deals with both requirements (Method 1) and design (Method 41) may be the best approach.

System Software Safety Objectives

- Known software-based hazards are controlled
- Dependability and fault tolerance requirements are satisfied via lower level requirements, software design, and implementation and that testing is in place to verify the fault tolerance behavior is not compromised by modifications
- All required functionality is implemented correctly and no unnecessary behavior is implemented
- To accomplish the objectives
  - Define a set of top-level claims related to critical events such as collision avoidance during docking, parachute operations, deorbit
  - Using hazard reports and system requirements documentation, establish the first levels of supporting claims
  - Continue developing supporting claims as the project proceeds, tracing in turn to requirements, design, code, and test
  - Capture and document evidence (requirements, design, code, test) at the lowest level of the safety case

System Software Safety Agile Variant

- Depth-first approach required
- Develop a complete safety case using postulated claims and evidence
- Update the case incrementally
  - Establish the top level claims using concept documentation, the system architecture, and high-level requirements
  - Develop the safety case as deep as possible using the early lifecycle artifacts
  - Postulate successively lower level claims across the breadth of the safety case, down to the evidence level
  - At each release(or more often if possible), revise the safety case to reflect the functionality implemented

Requirement Implementation Objectives

- Ensure requirements represented in design
- Design does not introduce capability that is not required
- Ensure all elements of the design are in code components
- Code does not introduce capability that is not required
- Ensure all requirements trace to code
- Ensure code components can reliably perform
  - Nominal conditions
  - Off-nominal conditions
- Documentation (both embedded and stand-alone) adequate for code maintenance.

Requirement Implementation Agile Variant

- Hierarchical requirements trace tool
- System model
  - UML/SysML
  - Flow diagrams
- Risk burndown tool

Conclusions & Future Work

- Pure Agile not appropriate for mission-critical or safety-critical projects
- Hybrid Agile gaining acceptance
- Adapt IV&V methodology to hybrid Agile
  - Maintain technical rigor
  - Accommodate project flows