



# Agile Refresh - BoT

February 1, 2019

# Agenda

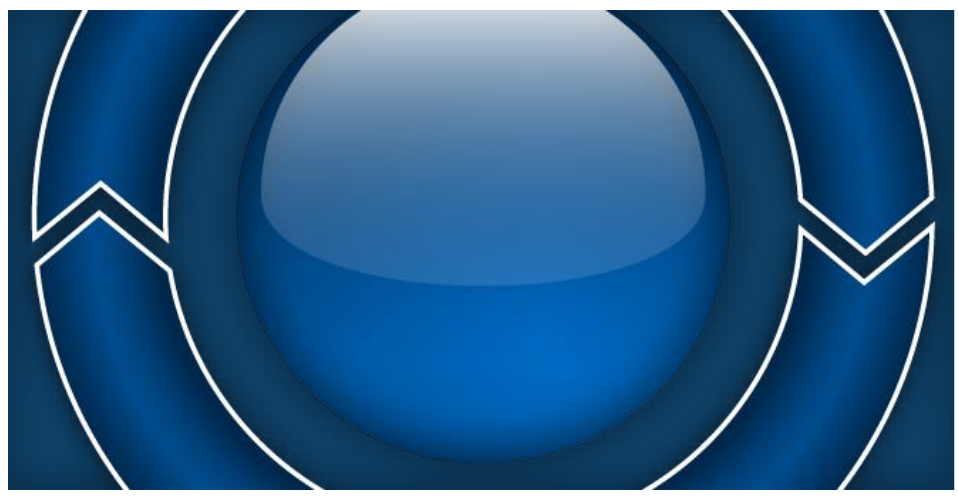
---

- Introductions
  - Name
  - Role (product owner, developer, quality engineer, etc)
  - Team
- Objectives
- Understanding Agile
- Scrum Deep Dive
  - Principles
- Epics/stories
- Retrospectives
- Backlogs
- SAFe

# Objectives

---

- Identify and leverage Agile principles to enhance **business, product and customer value**
- Enhance our ability to **deliver commitments on time** with high quality (focusing on general team practices vs. specific dev/QA/etc. practices) and predictability
- Use Scrum to deliver **business value quickly** and iteratively
- Address **common issues** with Agile and brainstorm solutions for your unique situation
- Establish terminology and process baseline and examples of deployment of standard practices to the broader organization:
  - Epics and User Stories
  - Backlogs and Backlog Management
  - Acceptance Criteria and Acceptance Tests
  - Sprint Planning
  - Estimating using Story Points
  - Daily Stand-Up
  - Definition of Done
  - Scrum Boards
  - Sprint Review and Retrospective



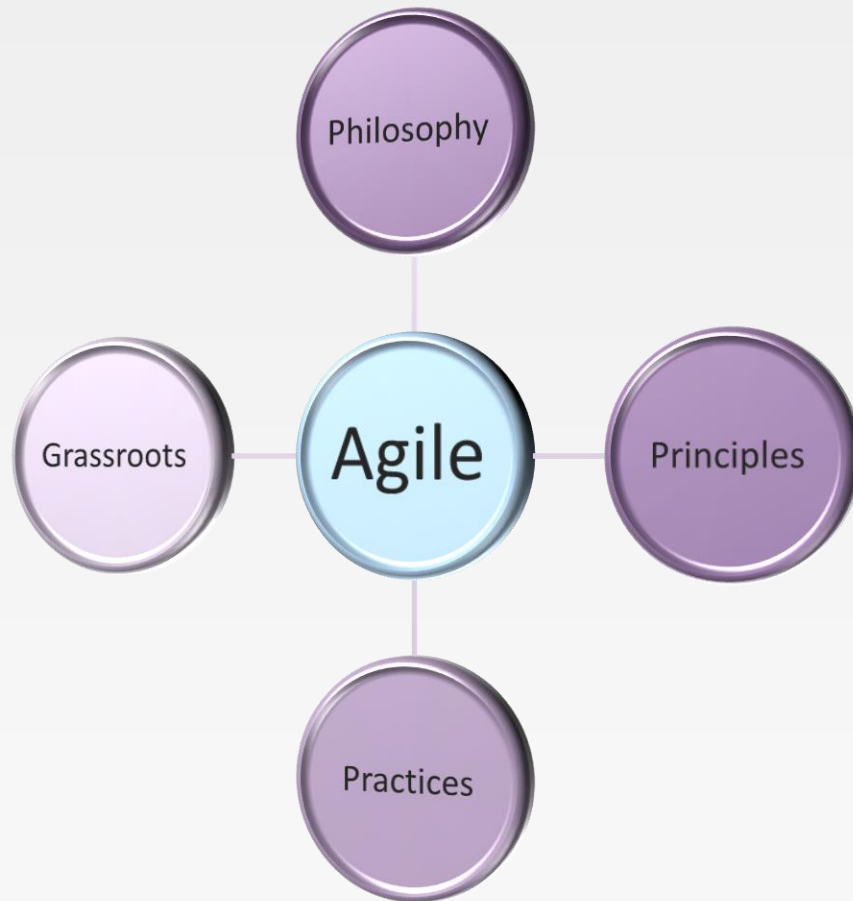
*How much experience do you have with  
Agile?*

## What is “Agile”?

Agile means different things to different people.  
Let's attempt to establish some commonality.

# What is “Agile”?

Agile methodology has gone from being a manifesto to an industry standard.



- Agile is not:
  - A panacea
  - A single method or methodology
  - Just a set of tools
  - Something that works for every group regardless of size or scope
- Agile does not:
  - Ignore or avoid documentation
  - Ignore or avoid up-front thinking
    - E.g. Architecture, Release Planning
  - Remove accountability

# Quick Comparison

---

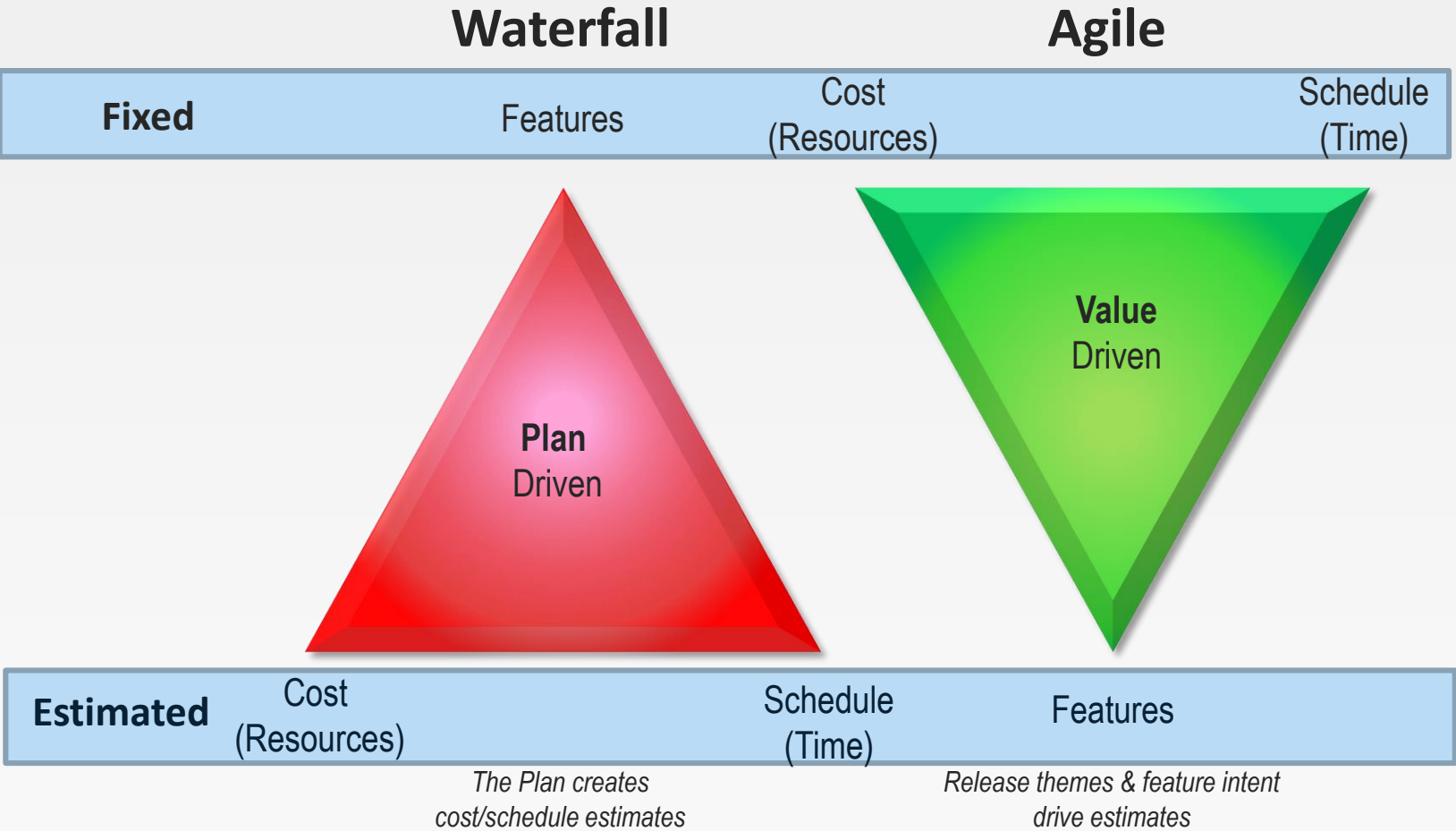
- **Waterfall**

- Traditional, Sequential Gantt design
- Unable to revisit a phase. Risky. Costly. Less efficient
- No value until the value of the end (deployment)
- Quality at the end - too late to discover issues
- No approval from stakeholders. Requirements likely to change
- Reliant on a plan. Maybe obliged to follow to the detriment of the project
- Reliance on a project manager.

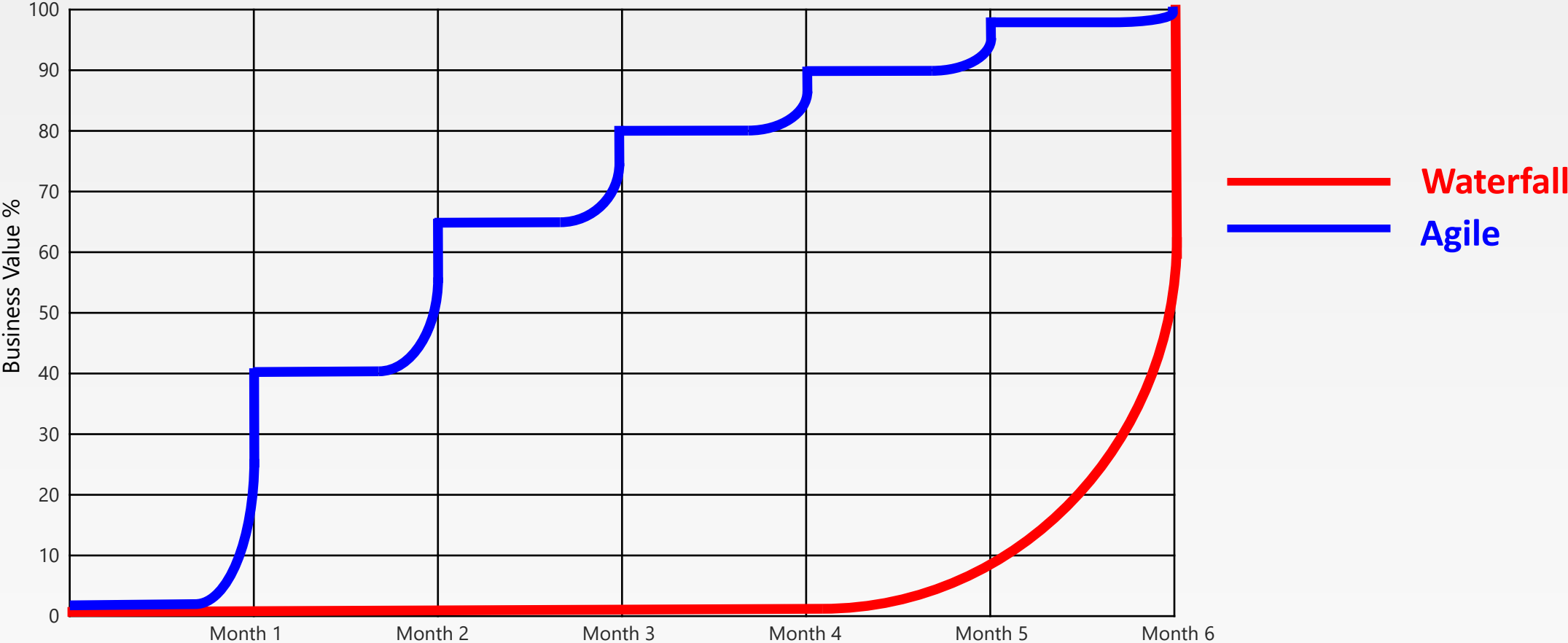
- **Scrum**

- Framework for complex projects.
- Emphasizes team collaboration.
- Balances power to the business and team
- Prioritized workload
- Scrum is about small team
- Building small things that amount to big things
- Short amounts of time and integrating
- High quality

# Paradigm Shift

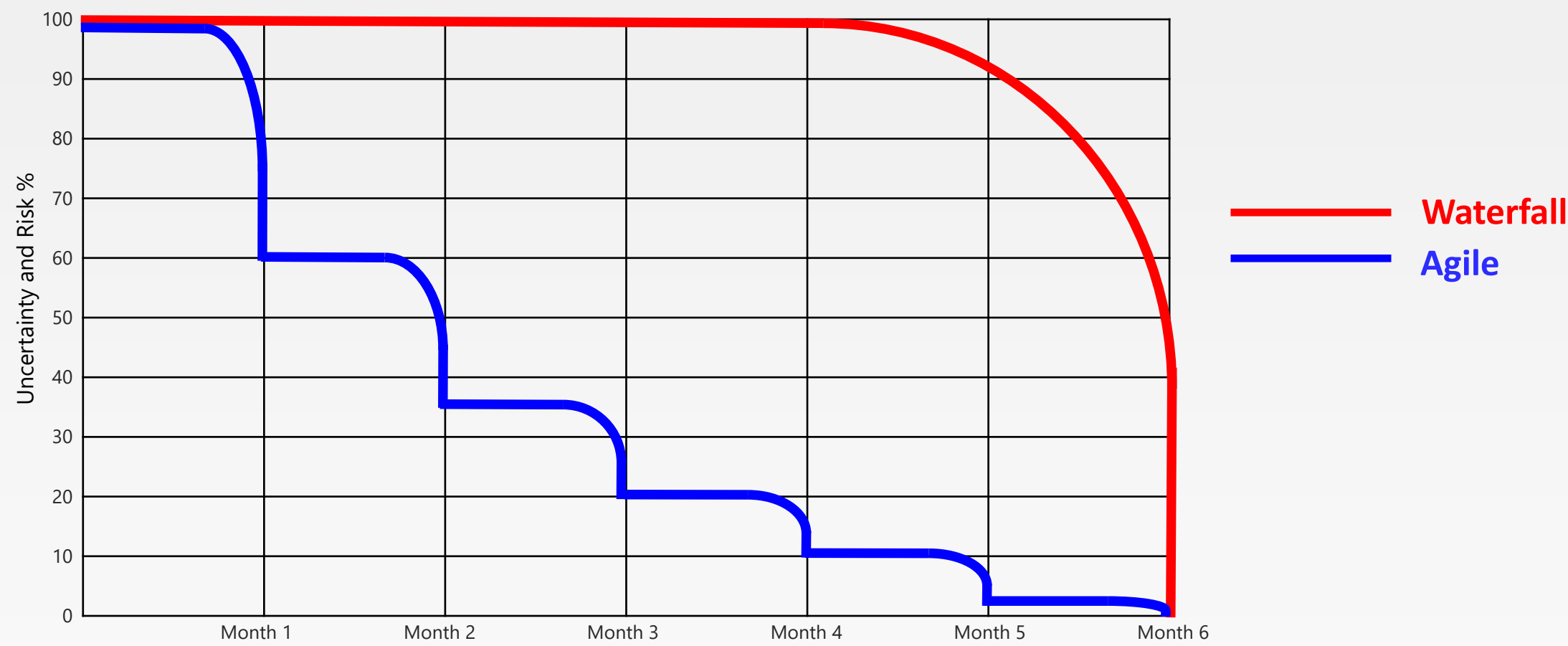


# Business Value Availability

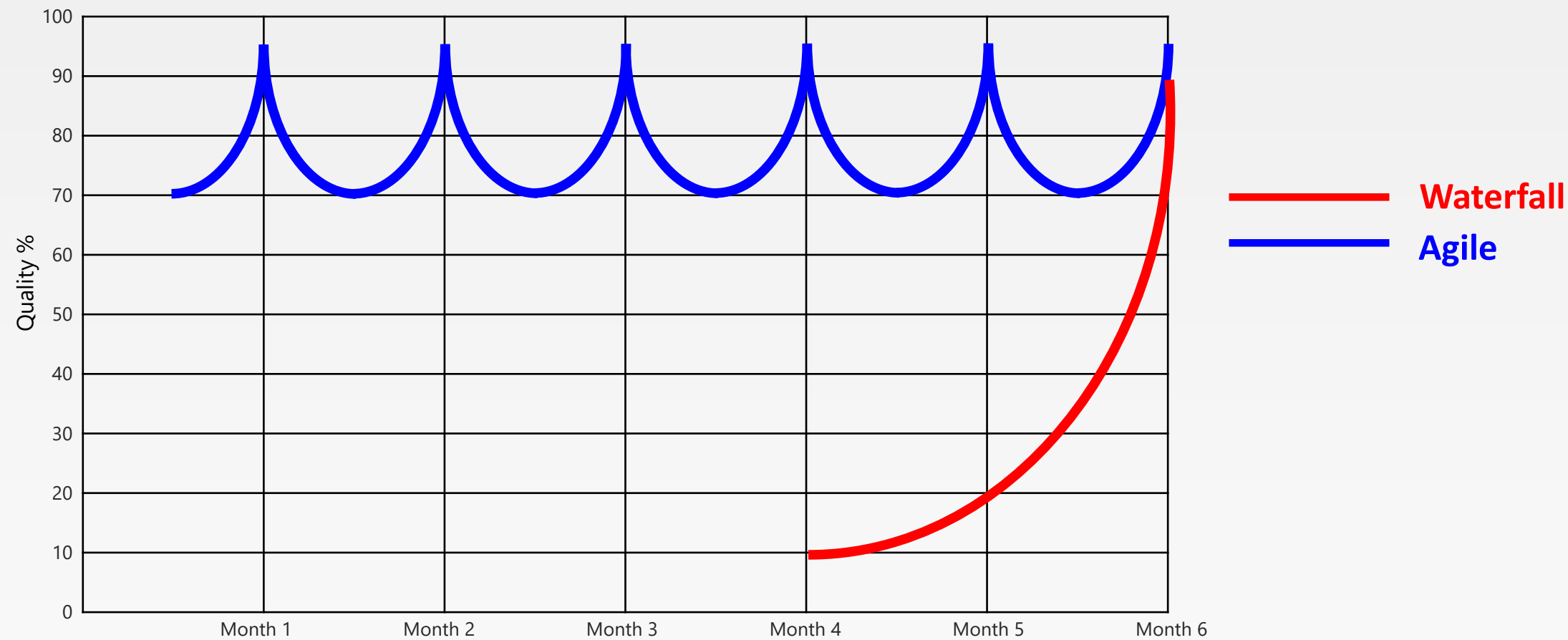




# Risk Management



# Quality Development

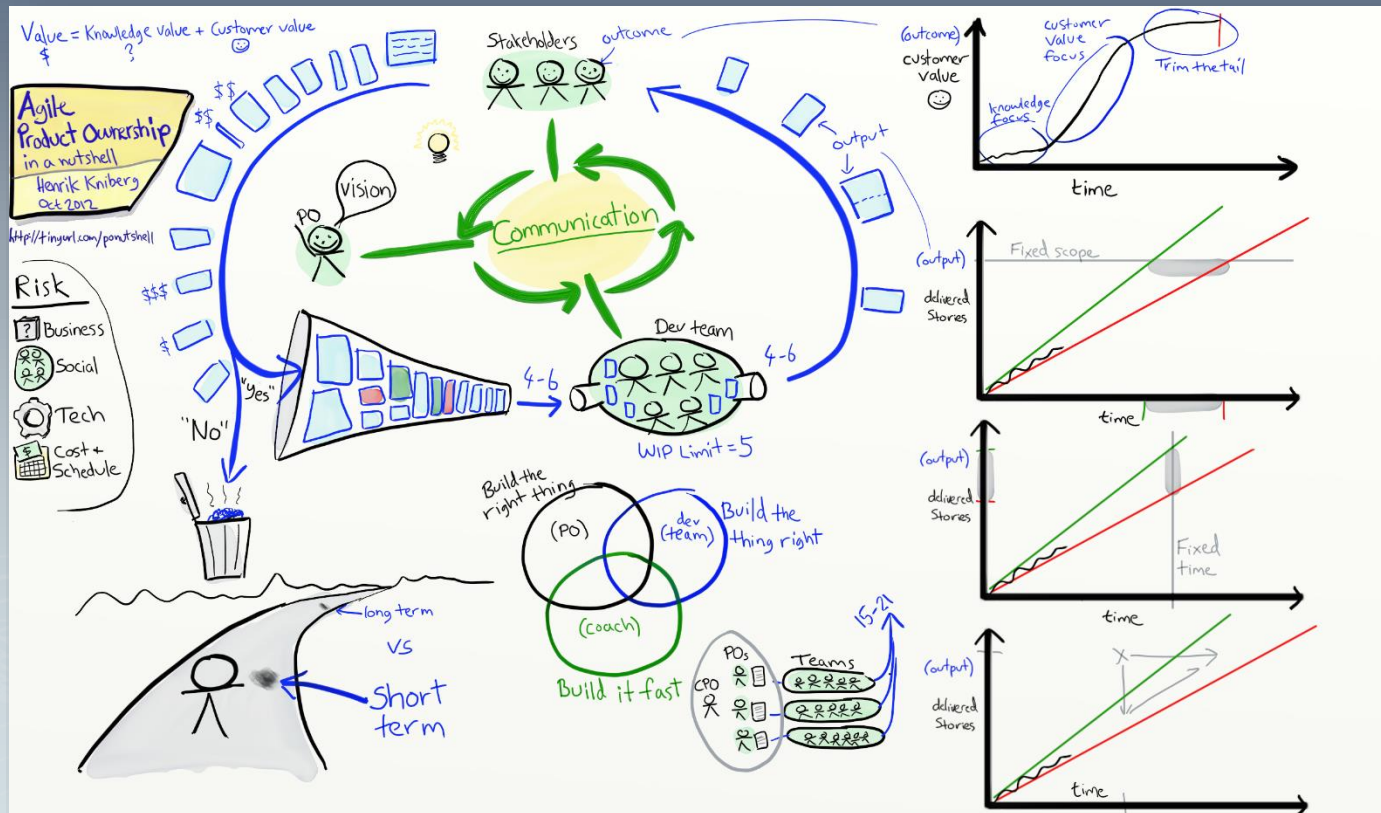


# Key Guiding Principles

- Simplicity (KISS) — Simplicity in everything we do
- Business Value and Alignment — Always work on the highest-value items
- Quality Throughout — Quality throughout the cycle; no “testing in” quality or throwing over the wall
- Commitment and Accountability — Do what we say we will do, but do not commit to unknowns
  - ... but defer commitment to last possible moment
- Rapid and Iterative Delivery — Quick wins for morale; quick fails for correction
- Collaboration and Transparency — Across disciplines; between teams; with stakeholders
- Continuous Improvement and Waste Elimination — Kaizen; avoid unnecessary work

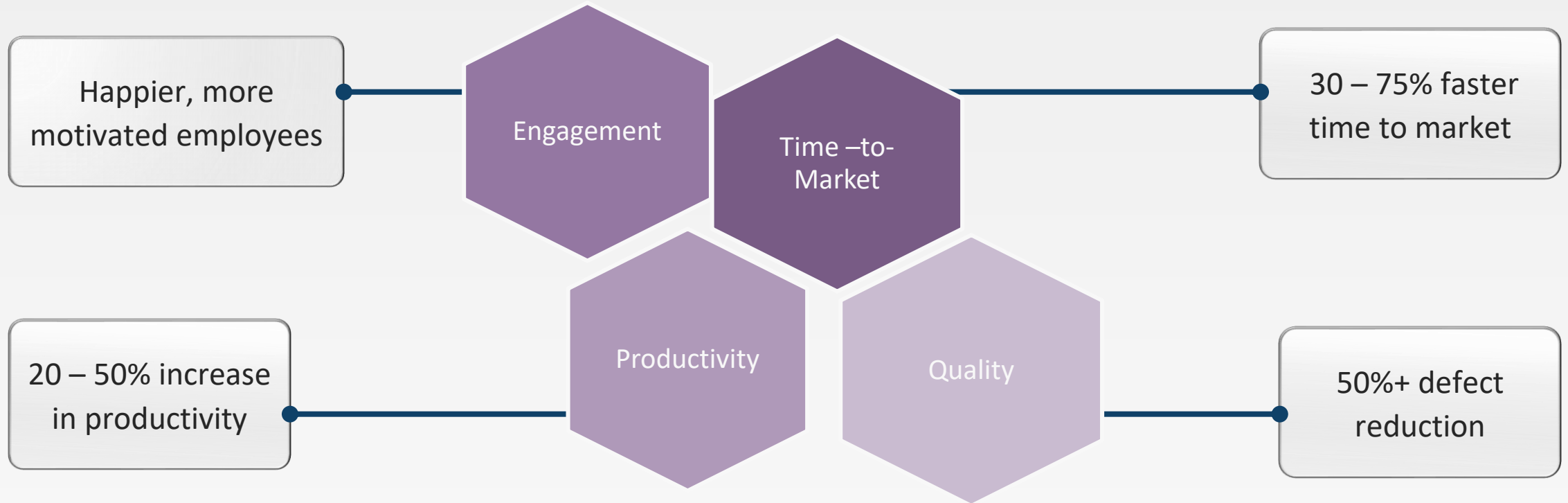


*Waterfall is about following a plan.  
Agile is about making sure the plan is correct.*



# The Result

---





# Scrum

Let's get into detail about the world's most popular Agile delivery methodology.

# Defining Scrum

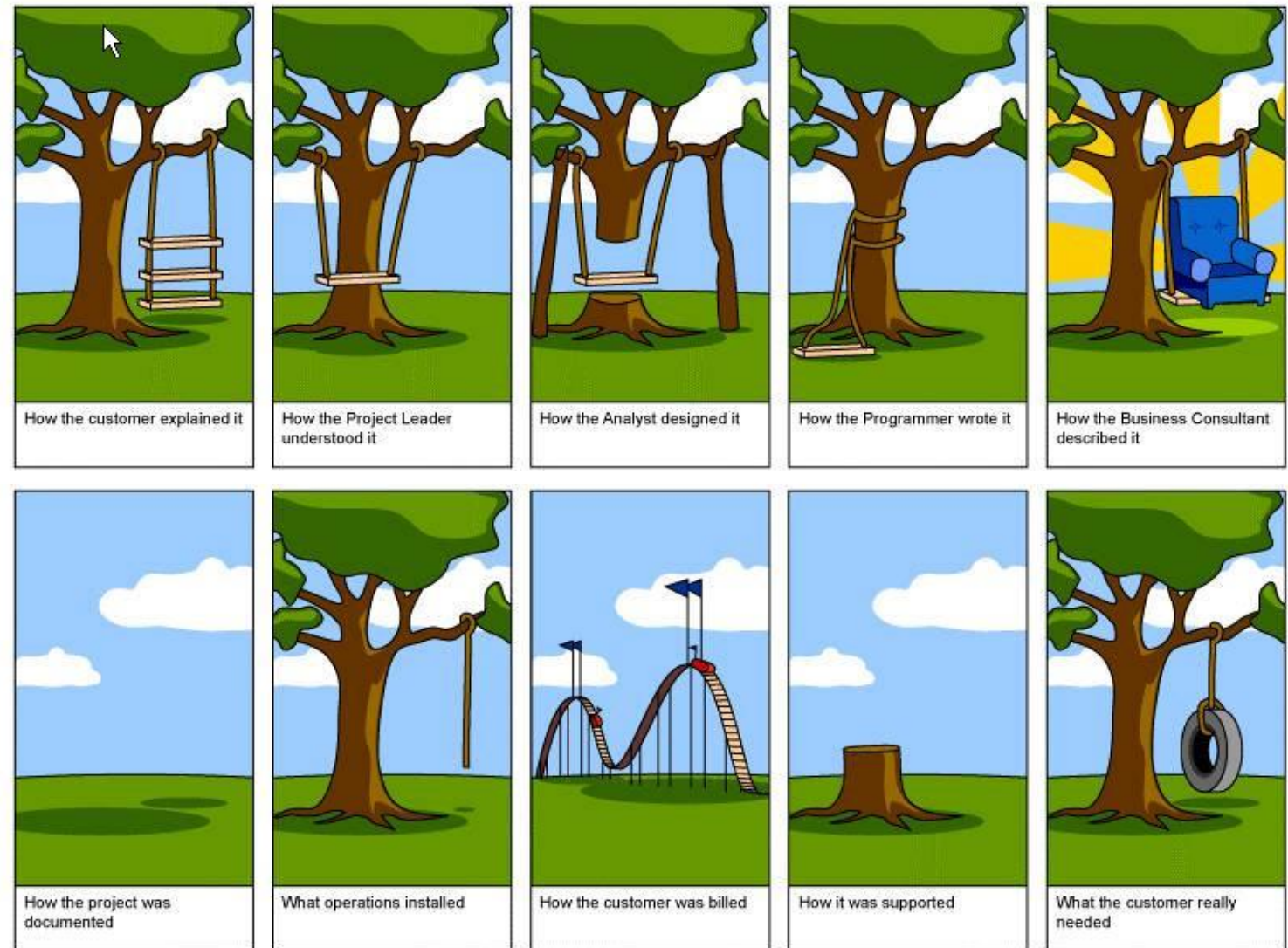
---

- **Scrum *n*.**

1. *Sports.* (From Rugby) A play in which the two sets of forwards mass together around the ball and, with their heads down, attempt to gain possession of the ball.
2. *Agile methods.* A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.
3. *Scrum.* A daily standup meeting where each team member answers three questions: what was done since the last scrum, what will be done before the next scrum, and what is impeding progress.

# Customer Needs & Agile

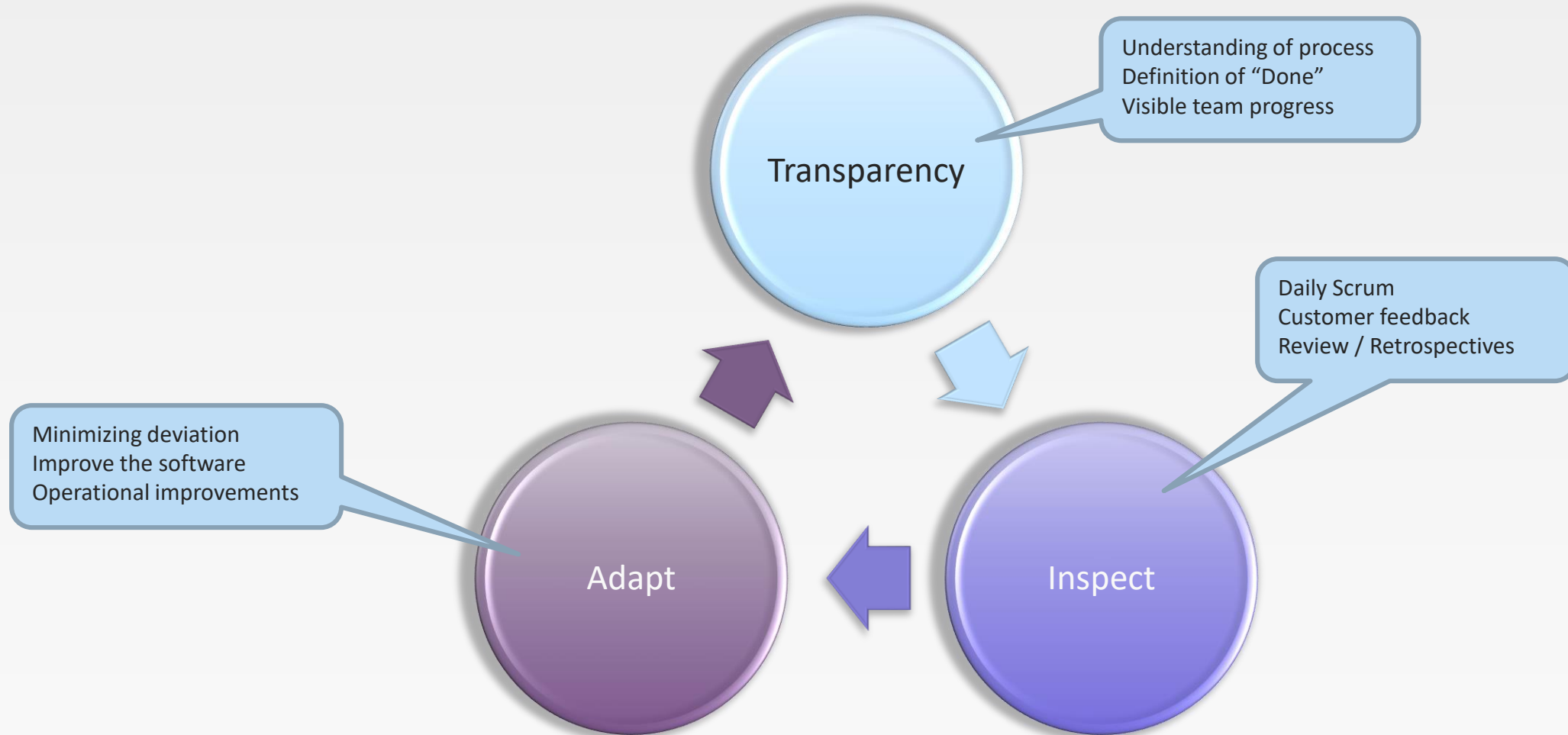
How do you think Agile (or Scrum) can help solve this problem?





# Scrum Pillars

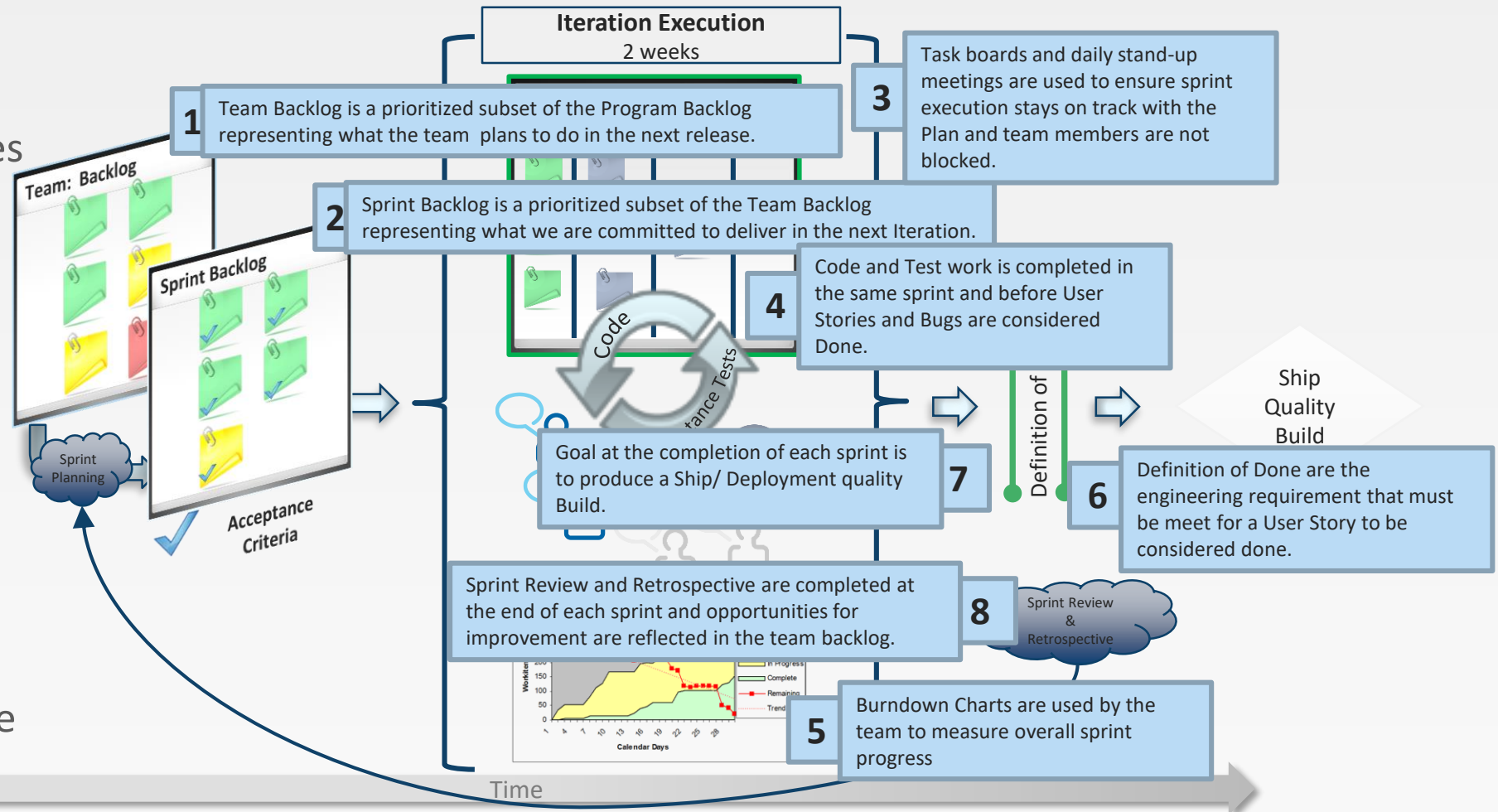
---



# Scrum Overview

## Standard Practices:

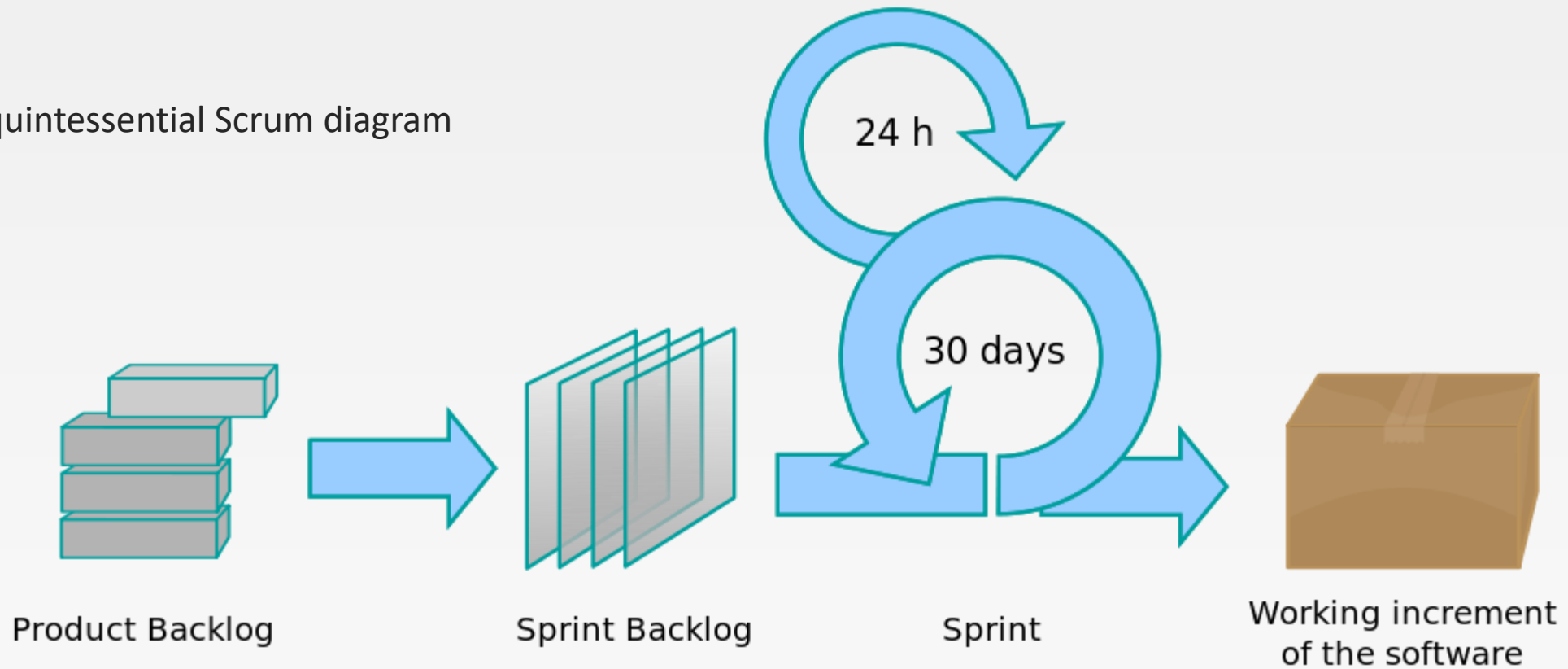
- Backlog Refinement (Grooming)
- Sprint Planning
- User Story Estimates using Story Points
- Daily Stand-Up
- Cumulative Flow Diagram
- Definition of Done
- Scrum Boards
- Acceptance Test Definitions
- Sprint Review
- Sprint Retrospective

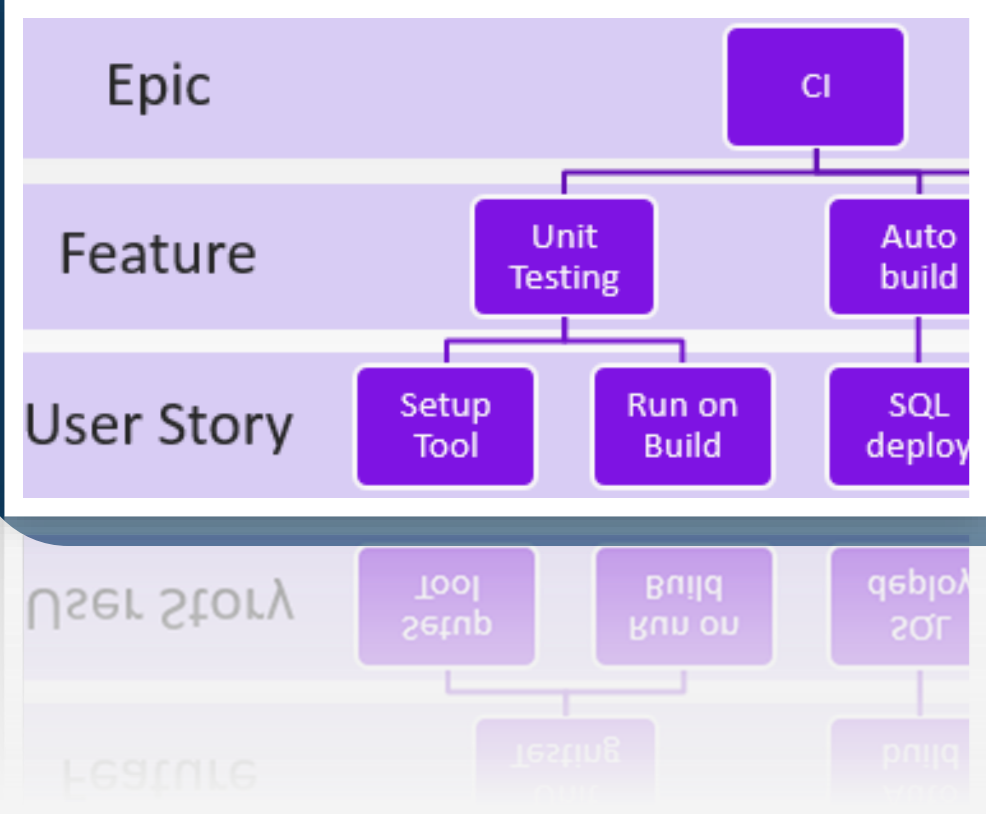


# Scrum Overview - Simplified

---

The quintessential Scrum diagram





# Understanding Epics

# Epics

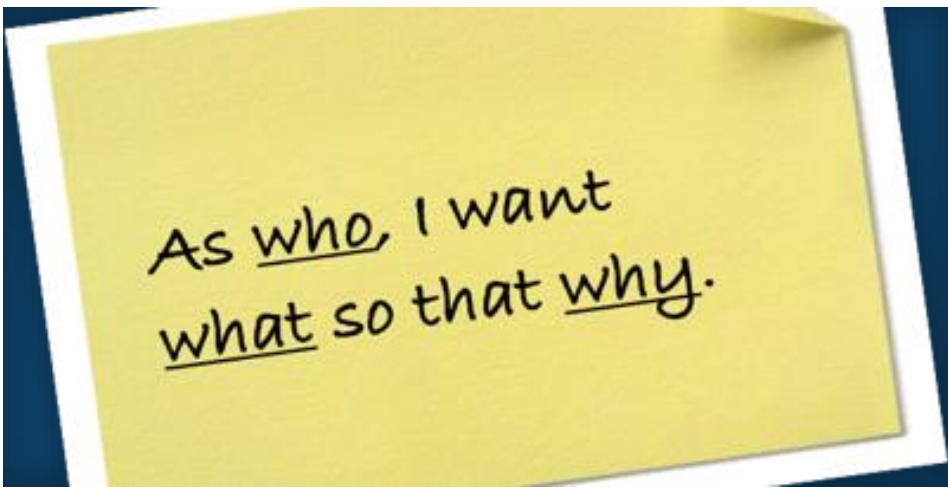
---

- **Definition:** a set of functionality that fulfills stakeholder needs, typically as part of a theme. It provides a measureable benefit within a release and is worked on within a single release.
- Characteristics:
  - Measurable benefit
  - Typically part of a theme
- Scope
  - Deliverable within a Release
- Estimation
  - Wideband Delphi - Days or Weeks
- Prioritization
  - Stack-ranked to create a Program Backlog
- States:
  - Not started
  - In Progress
  - Done
- Benefit statement – a short description which describes the benefit to the user or business

# Epic MVP

---

- Accomplished within a Program Increment (4 sprints) – many times a MVP
- A minimum viable product (MVP) is a product with just enough features to satisfy early customers, and to provide feedback for future product development.
- Less expensive than developing a product with more features, which increases costs and risk if the product fails, for example, due to incorrect assumptions.
  - Be able to test a product hypothesis with minimal resources
  - Accelerate learning
  - Reduce wasted engineering hours
  - Get the product to early customers as soon as possible
  - Base for other products
  - To establish a builder's abilities in crafting the product required
  - Brand building very quickly



As who, I want  
what so that why.

# Writing a great user story

# User Story Overview

- **Definition:** A desired user behavior typically defined in everyday language of the user, like:

**As a (role) I want to (do some action) in order to [outcome]**

- Contains:
  - User
  - User Goals & Benefit
  - Acceptance Criteria
  - Testable
- Estimation
  - Story Points
  - Planning Poker
- Prioritization
  - Stack-ranked Ordered next iteration, in the Team Backlog
- Basic States
  - Not started
  - In Progress
  - Completed
- Target Scope
  - ~5 days of development effort
  - Deliverable in one iteration

*Title:* Tweets on Website

*Description:* As a visitor to the Tailspin Toys website, I want to see recent tweets about the company and its product so that I am better informed.

Confidence- High

*Effort-* 5 points

- Tweets must be shown on the home page
- Use the company's Twitter account
- New tweet should appear < 15 minutes after posting
- Filter out spam and profanity



# User Story Fields (Example)

Field Name	Required?	Notes
Title	Y	<b>Succinct title for the User Story</b>
State	Y	State of the User Story
Owner	Y	<b>Product Owner accountable for the User Story</b>
Team	N	The team delivering the User Story
Assigned To	Y	<b>Who is currently working on the User Story</b>
Stack Rank	Y	<b>Numerical sequence identifier (allows sorting)</b>
Effort	Y	<b>Estimate of amount of effort (could be story points, or hours); could also use original estimate/remaining work idea as well</b>
Narrative	Y	<b>"In order to &lt;achieve my goal&gt;, as a (role) I want to (do some action)"</b>
Acceptance Criteria	Y	<b>How the success of the story will be measured. E.g. <i>Given</i> I am a valid user of the banking system, <i>When</i> I login to the web site, <i>Then</i> I see my account summary</b>
Scope	Y	<b>The general product scope at which the story applies</b>
Iteration	N	Sprint/iteration that the user story is targeted to be worked on
Attachments	N	Any attached files, such as documents or UI mock-ups
Links	N	Linked Epic, linked test cases that validate this user story, linked tasks that implement the story



# Writing Good User Stories

As a <role> == Specific User / Role  
I want to <action> == Problem  
In order to <outcome> == Value

## + Acceptance Criteria

REMEMBER: User stories contain basic information. The real value comes from the conversation

**I***ndependent* – Self contained, no inherent dependency on other stories (stand-alone slice)

**N***egotiable* – Statements of intent – not implementation; can change until committed to in a sprint

**V***aluable* – Provides value to the customer

**E***stimable* - Always should be able to estimate the size or complexity

**S***mall* – Small stories make it easier to satisfy the other INVEST requirements

**T***estable* – Has the necessary info to facilitate testing

## Agile Alliance: INVEST

# Bad Story

---

- I want the brochure to be colorful.
- Fix exception handling.
- Let users make reservations.
- Users want to see photos.
- Show room size options.
- *As a user, I can provide best support service to my customer.*

# Good Story

---

*As a <type of user>, I want <some goal> so that <some reason>.*

Epic: As a user, I can backup my entire hard drive.

- As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.
- As a power user, I can specify files or folders to backup based on file size, date created and date modified.

# Acceptance Criteria

Acceptance criteria helps tell you when you are done with a story.

What are “Acceptance Criteria”?

**Conditions that a software product must satisfy to be accepted by a user, customer or other stakeholder.**

- Written quickly
- Just enough documentation to define the “doneness” of a user story
- The result of a conversation to clarify the user story
- Documented *along with* the user story (i.e. story not complete without ACs)
- Driven by PO/PM and written by the team
- Can capture non-functional requirements and other system behavior
- Can provide story splitting guidance
- A pre-cursor to more structured acceptance criteria (Gherkin; more later)

# Anatomy of Acceptance Criteria

---

- Based on a language called Gherkin

***Given*** <pre-condition>  
          ***[And*** <condition>***]***  
***When*** <actor + action>  
          ***[And*** <condition>***]***  
***Then*** <observable result>

Using a standard language makes tests consistent and easy to understand

# Acceptance Criteria Example

---

- Example User Story:
  - As a registered user of the web site, I want to login so that I can access my account summary page
- What are some example Acceptance Criteria:

*Given* I am a registered and valid user of the banking system

*When* I login to the web site

*Then* I see my account summary

*Given* I am an unregistered or invalid user of the banking system

*When* I login to the web site

*Then* my login is rejected

# Acceptance Criteria Example

---

- **As a Consumer, I want to be able to see my daily energy usage so that I can start to understand how to lower my costs over time**
- **Acceptance Criteria:**
- Read DecaWatt meter data every 10 seconds and display on portal in 15 minute increments and display on in-home display every read
- Read KiloWatt meters for new data as available and display on the portal every hour and on the in-home display after every read
- No multi-day trending for now (another story).
- ...



# Write two user stories



## INVEST in your Stories:

1. Practice writing 2 user stories (potentially in your backlog). Make them:
  - a) Independent
  - b) Negotiable
  - c) Valuable
  - d) Estimable
  - e) Sizable
  - f) Testable
2. Ensure they have good acceptance criteria?
3. When everyone on your team is done, compare notes – read out loud.

- Celebrate
  - *What did we do well, that if we don't discuss we might forget?*
- Be Critical
  - *What did we not do so well?*
- Change
  - *What should we do differently next time?*

# Retrospectives

# Sprint Retrospective

*Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.*

# Retrospectives

---

- Celebrate
  - *What did we do well, that if we don't discuss we might forget?*
- Be Critical
  - *What did we not do so well?*
- Change
  - *What should we do differently next time?*

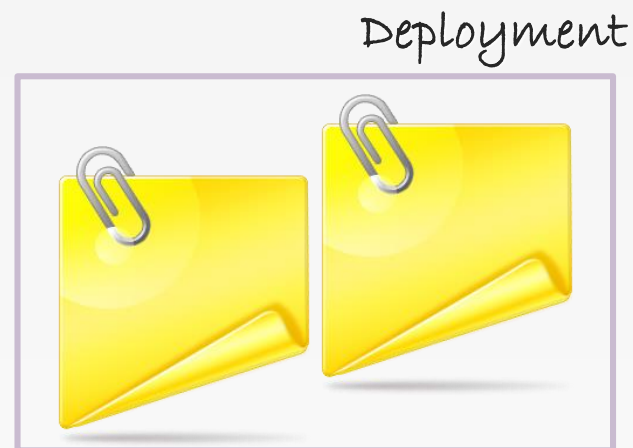
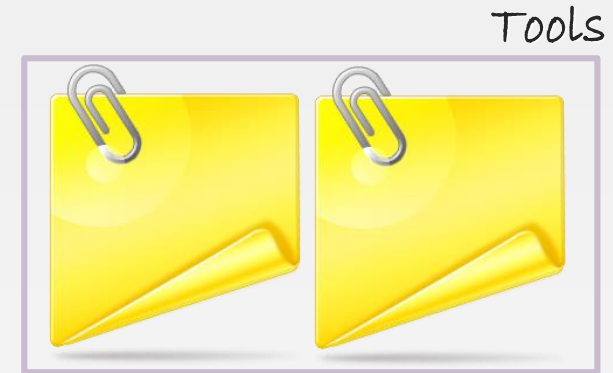


**The 3 C's**

# Retrospectives: Affinity Exercise

1. Individual brainstorming
2. Categorization
3. Clarifications
4. Idea expansion
5. Voting

What went well?  
What could have gone better?  
What concrete actions should we do differently?



# Retrospective Template

- Retrospective Template

## <Team Name> Retrospective

Sprint Goal: <sprint goal>

Sprint #: <sprint #>

Sprint dates: <from> - <to>

### Burndown

<insert a copy of the team's burndown chart here>

### What went well?

- <item 1>
- <item 2>
- :

<Alternatively, paste in a picture of the whiteboard if this was captured during the retrospective>

### What could have gone better?

- <item 1>
- <item 2>
- :

<Alternatively, paste in a picture of the whiteboard if this was captured during the retrospective>

### Review Actions from Last Retrospective

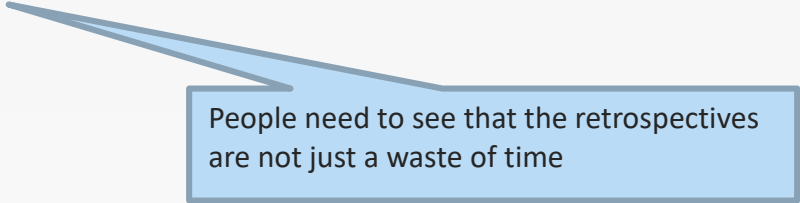
- <item 1>
- <item 2>
- :

# Retrospective Best Practices

---

What are some best practices for a retrospective?

- Facilitated meeting
- NO management
- Never skip as the team gels
- Prepare!
- Time box - ~30min per week of sprint (e.g. 2 week sprint = 1h)
- Everyone contributes (entire team engaged)
- Brainstorm – there are no wrong answers
- Do not find fault (nothing is personal)
- Get to root cause (e.g. “[The 5 Why’s](#)”, [The Six Boxes](#))
- **Improvements should be actionable commitments for next sprint, with dates/assignees**
  - Create a stack-ranked backlog of improvements, and focus on a small # of things to fix; track in your tool



People need to see that the retrospectives are not just a waste of time

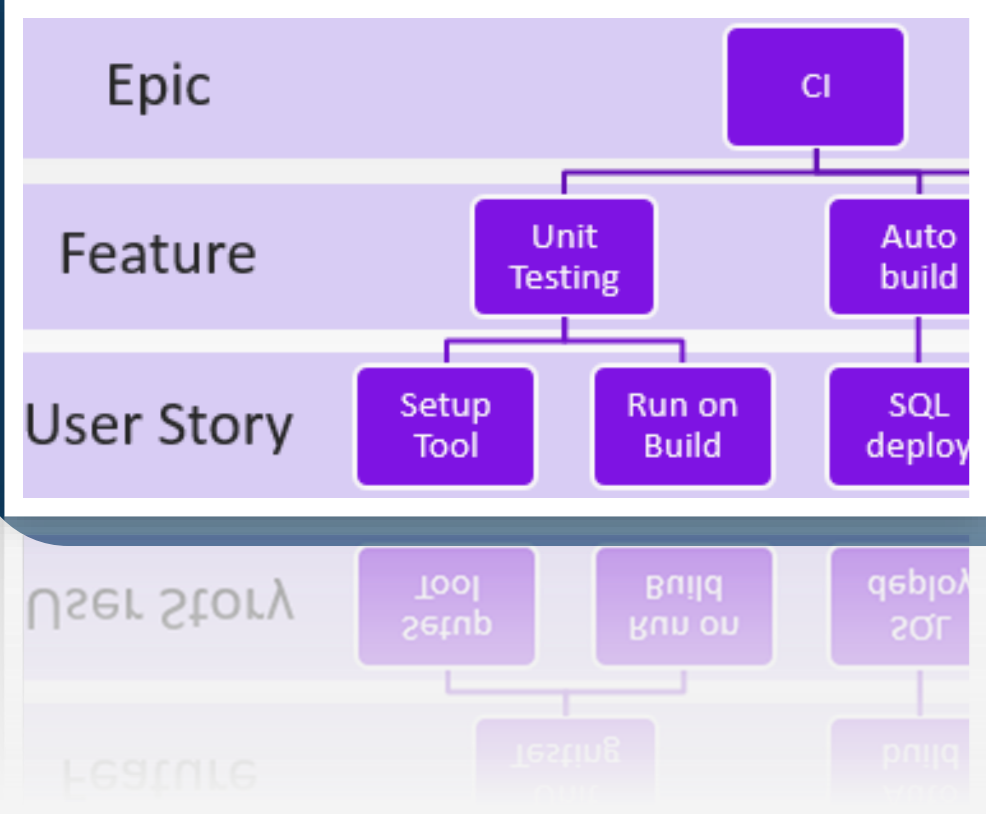
# Sprint Retrospective



**OBJECTIVE: Conduct a retrospective using an affinity exercise.**

1. Get together with your team
2. Decide on doing a retrospective on one of the following:
  - The Mojaloop community workshop
  - A very recent iteration of development
3. Do an affinity exercise
  - a) What went well?
  - b) What did not go so well?
  - c) Actionable improvements
4. Report out top 2 actionable improvements

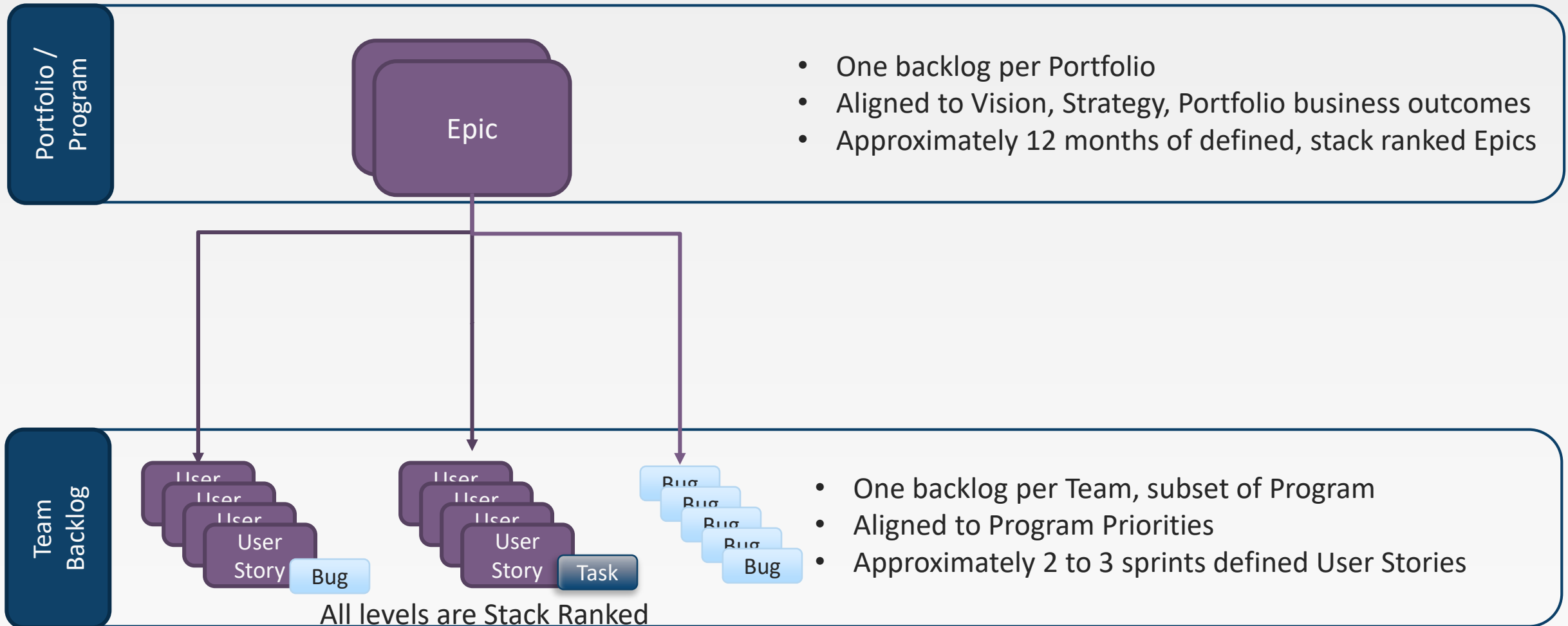




# Backlog Refinement

Splitting and Ranking Epics, Features, and Stories

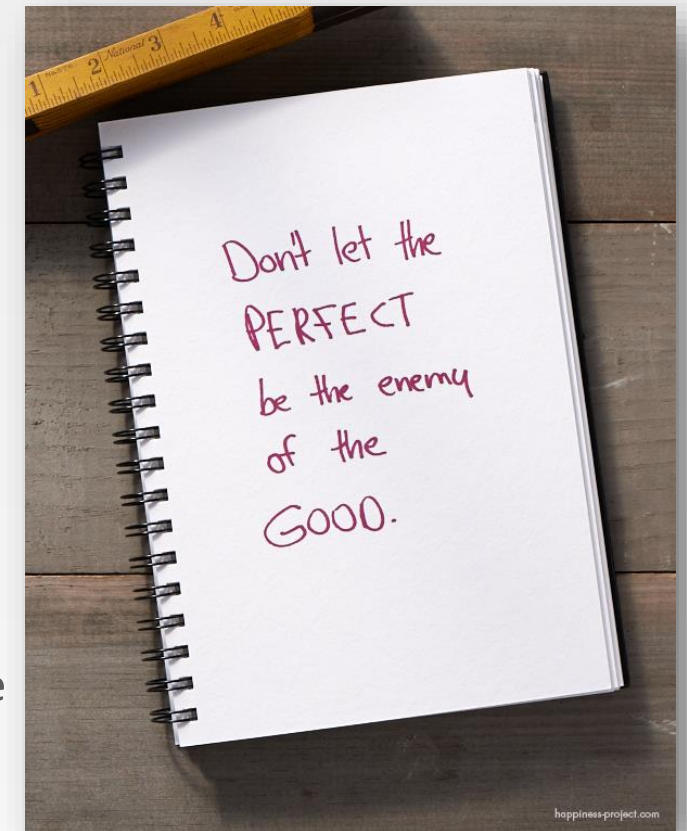
# Backlogs: Align priorities and requirements across streams



# Team Backlog Refinement

---

- Previously called “Grooming”
- Ongoing, but main event done just prior to next sprint planning
- Ensure backlog is stack-ranked accordingly
- Create user stories for the top-ranked epics
  - Break larger stories into smaller ones
  - Understand the “what” of the stories
- Entry Criteria before Sprint Planning
  - Stack ranked list of User Stories
  - Enough basic information to effectively describe the stories and size the user story (extra credit – user stories are estimated before sprint planning)
  - Bullet-point acceptance criteria for each user story being considered for the sprint



# Ranking the Backlog

Ultimately, who's butt is on the line?

- Dot voting
- Business Value Game
- ROI Estimates
- Weighted Decision Matrix / Relative Scoring
- Kano Model
- Weighted Shortest Job First
- Relative Weighting
- Customer surveys (large sample)

If you have problems agreeing on a stack rank, there may be other problems.

- Is the team aligned on the vision of the product?
- If you are stack ranking the next release, is there still debate on the goal of the release?
- Are the customer views being factored in enough here with a large enough sample size or are individuals trying to push their way without enough evidence?
- Is there no Product Owner (PO) butt-on-the-line that is responsible for the final call?

# Backlog Review



**OBJECTIVE: Review the backlog and prioritize with the team**

1. Get together with your team
2. Determine a method for prioritization
3. Prioritize this list of every day tasks

# Task List



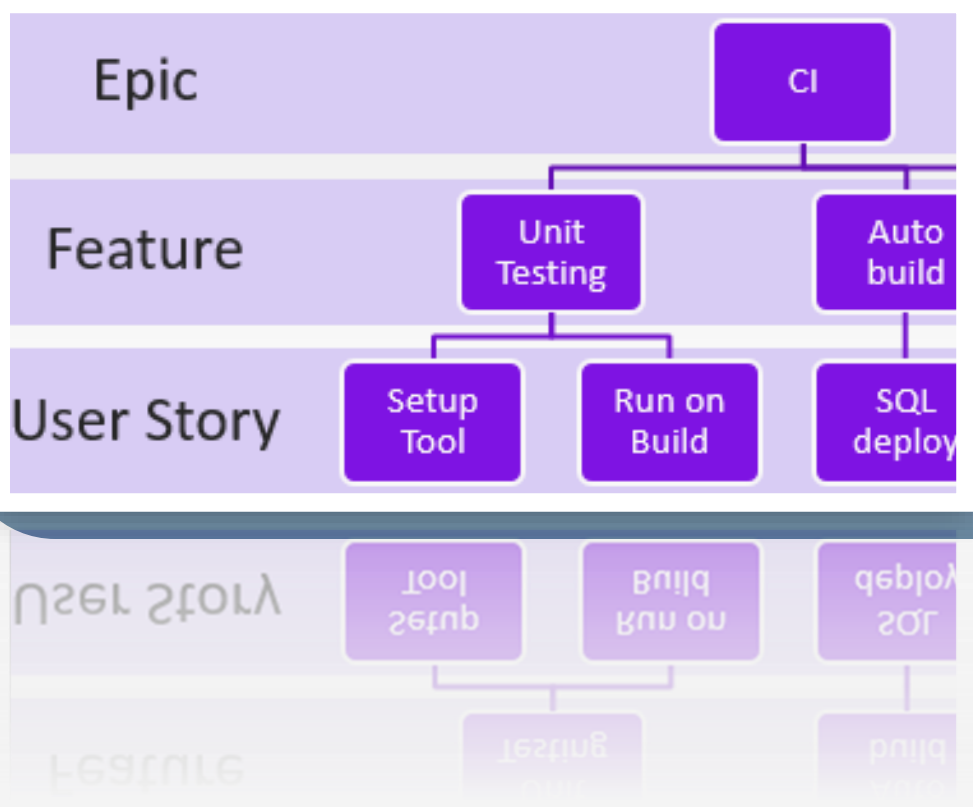
- **Call mom**
- **Buy groceries**
- **Build a shed**
- **Let out the dog**
- **Clean the house**
- **Mop the floor**
- **Plan a trip**
- **Go to post office**
- **Write an email**
- **Take out garbage**

# Backlog Review



**OBJECTIVE: Review the backlog and prioritize with the team**

1. What method did you use?
2. How did the team work together?
3. What were the challenges?



# Scaled Agile Framework



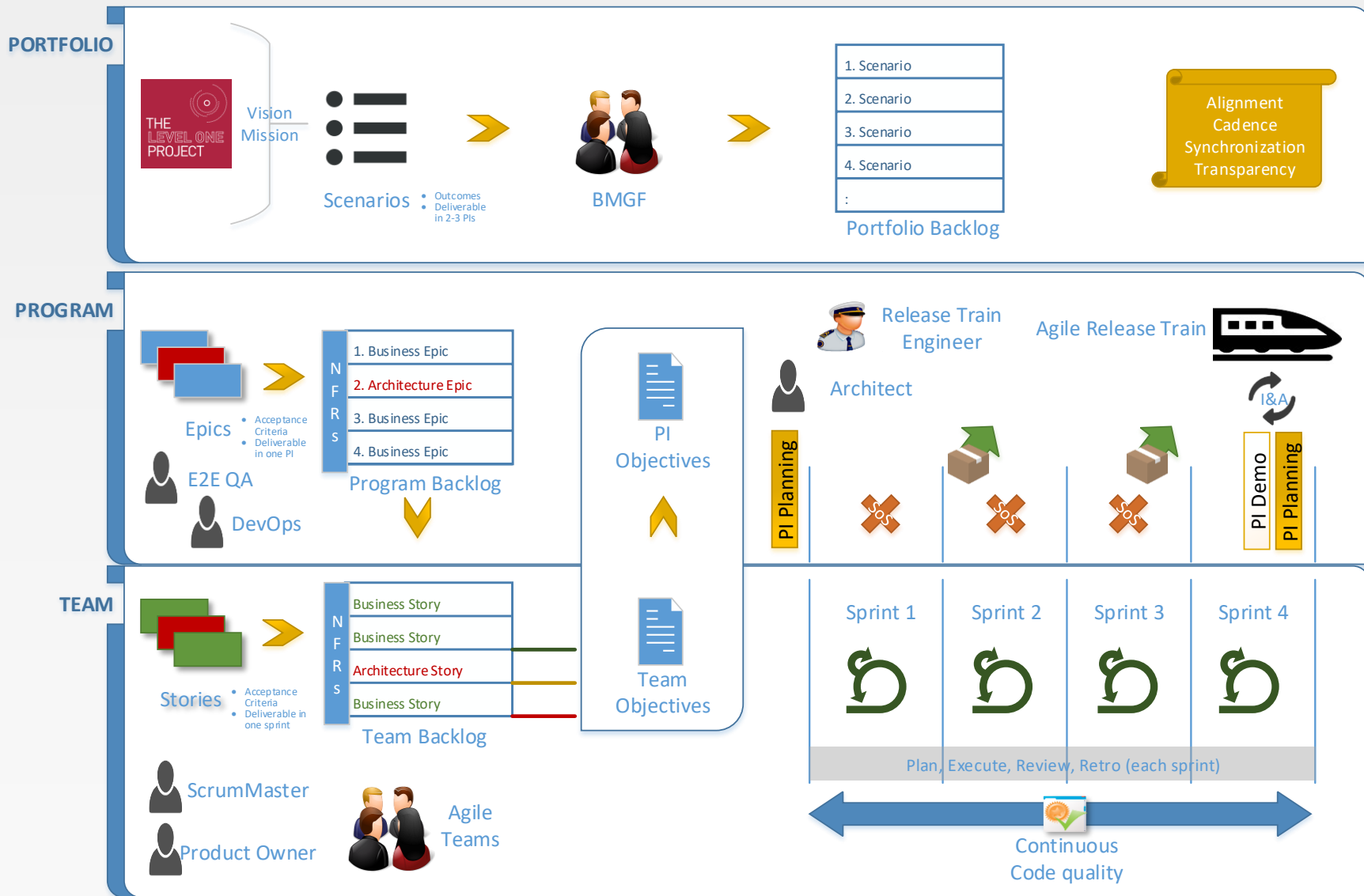
# Why SAFe (Scaled Agile Framework)?

---

- Relatively lightweight framework that creates efficiency in software development
- Handle a coordinated strategy for large-scale and complex projects
- Work across teams, as its centralization makes multi-team coordination possible.
- Standardizes processes across teams and helps avoid obstacles and delays that may pop up when different teams need to work together.
- Ability to help teams maintain alignment with business goals.
  - This alignment can often get lost in agile environments that take more of a bottom-up approach, as developers and testers can sometimes lose sight of bigger picture business objectives.
- Top-down alignment and centralized decision-making helps ensure strategic objectives remain top of mind and that all decisions get made in support of those objectives.

# Overall Approach to Mojaloop

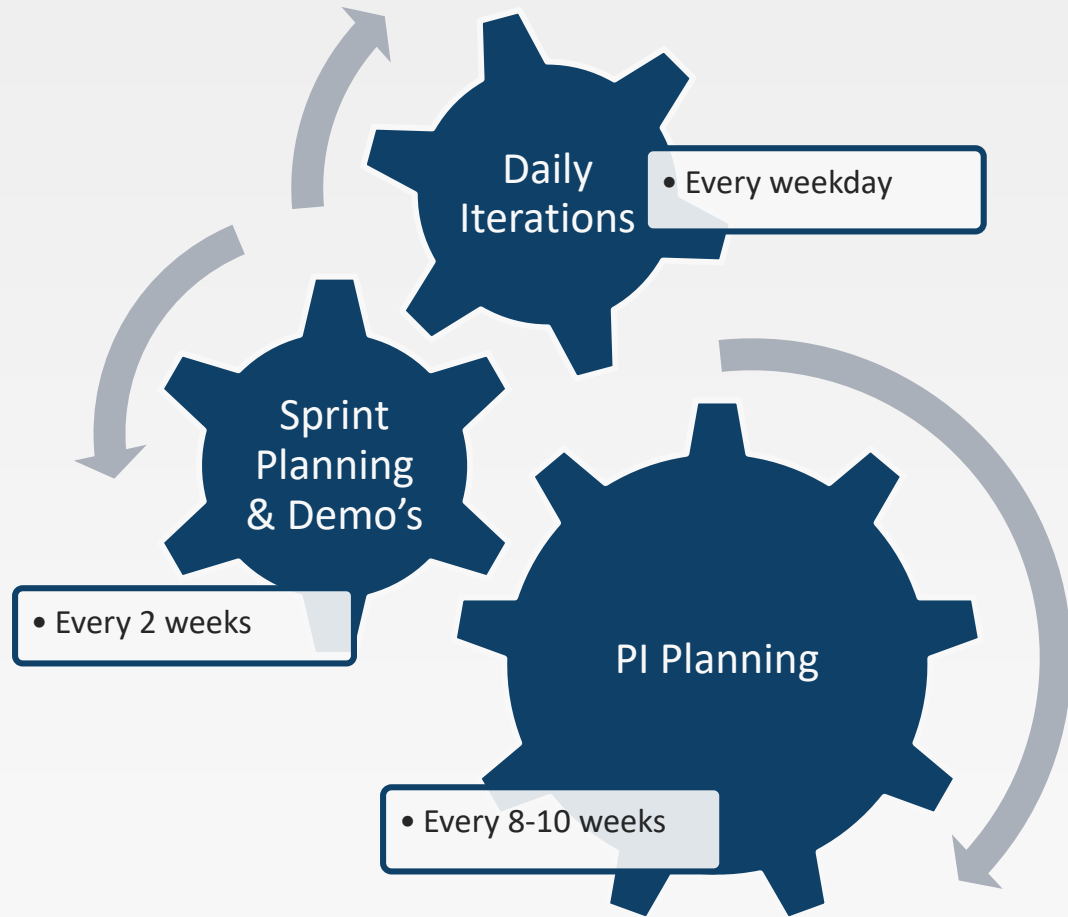
(Simplified) Scaled Agile Framework



# Project Management Overview

---

- Program Increments
  - Sprint Planning
    - Daily Stand-ups
  - Sprint Demo
  - Sprint Retrospective
- Release Integration
- Release Demo
- Release Retrospective
- (Repeat PI – N)



# Program Increment (PI) Planning Objectives & Outputs

## Objectives

- Establish further communication and collaboration across the teams/community
- Identify dependencies and foster cross-team coordination
- Ensure we have 'just the right' amount of architecture and epic guidance for the next sprint
- Address open parking lot items and accelerate decision making
- Allow time for innovation and exploration

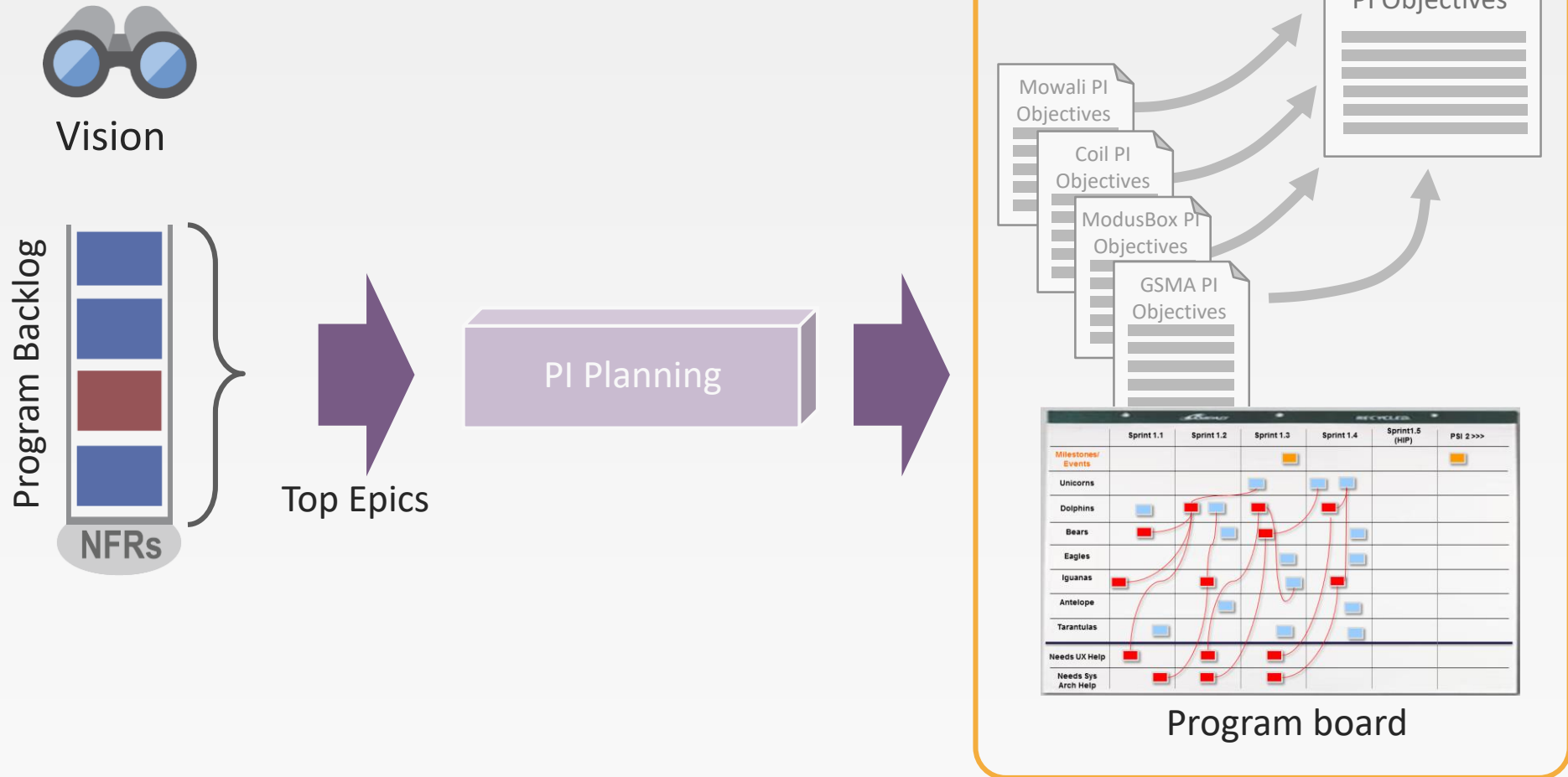
## Outputs

- All PI-5 Epics and acceptance criteria are captured, reviewed and understood by all the teams
  - Program board highlights completion date for all epics and dependencies
- Clear set of PI-5 objectives for each team (created by the team)
- Confidence and **commitment** around the PI-5 objectives
- Plan (Program Board) created for team commitments on what will be accomplished over the next 10 weeks.

# PI Planning Process

Input: PI 5 Themes

Outputs: Program PI Objectives, Epic Program board & Risks



# Program Increment Schedule

<b>PI 5 On-Site</b>	1/29/2019	1/31/2019	3 days	Arusha
5.1	2/4/2019	2/15/2019	2 weeks	
5.2	2/18/2019	3/1/2019	2 weeks	
5.3	3/4/2019	3/15/2019	2 weeks	
5.4	3/18/2019	3/29/2019	2 weeks	
5.5	4/1/2019	4/12/2019	2 weeks	Hackathon: 4/8
<b>PI 6 On-Site</b>	4/16/2019	4/18/2019	3 days	Johannesburg
6.1	4/22/2019	5/3/2019	2 weeks	
6.2	5/6/2019	5/17/2019	2 weeks	
6.3	5/20/2019	5/31/2019	2 weeks	
6.4	6/3/2019	6/14/2019	2 weeks	
6.5: QA & Prep	6/17/2019	6/21/2019	1 week	Fixes, QA, Prep
<b>Phase 3 Wrap-Up</b>	6/25/2019	6/27/2019	3 days	TBD
Wrap-up & Release	7/1/2019	7/12/2019	2 weeks	

# Backlog Review



**OBJECTIVE: Review the backlog and prioritize with the team**

1. Get together with your team
2. Determine a method for prioritization
3. Prioritize this list of every day tasks
4. Report back

# SMART Team Objectives

Teams should write their **Team Objectives** in the “SMART” format



- **Specific** States the intended outcome as simply, concisely, and explicitly as possible. (Hint: Try starting with an action verb.)
- **Measurable** It should be clear what a team needs to do to achieve the objective. The measures may be descriptive, yes/no, quantitative, or provide a range.
- **Achievable** Achieving the objective should be within the team’s control and influence.
- **Results-oriented** Measure outcomes not activities
- **Time-bound** The time period for achievement must be within the PI, and therefore all objectives must be scoped appropriately



# Team Planning Procedure

## TEAM BOARDS

### **Negotiate, and gain agreement on PI objectives**

- Discuss with your and other teams
- Make them “SMART” & Write Draft objectives

### **Review Program Backlog Items & Themes**

- Review Epics. Elaborate acceptance criteria
- Review nonfunctional requirements (NFRs)

### **Estimate and record velocity for all Iterations on the ‘Iteration’ team board**

- Based on prior periods (Factor in team size, team changes, holidays, and vacations)

### **Estimate & Load Epics into Iterations on team ‘Iteration’ board**

- Use poker cards to estimate Epics
- Update capacity/load information

## GROUP BOARDS

### **Place Epics on Project Board**

- Epics should be place in the iteration they will be DONE.
- Identify, discuss & address interdependencies and update the program board.
- Epics = **BLUE**, Dependency = **RED**.

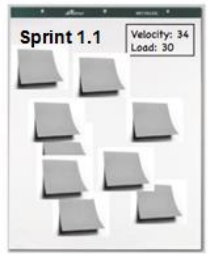
### **Identify impediments and risks**

- If local to the team, resolve
- If Program level, place issues on RISK board and be ready to discuss

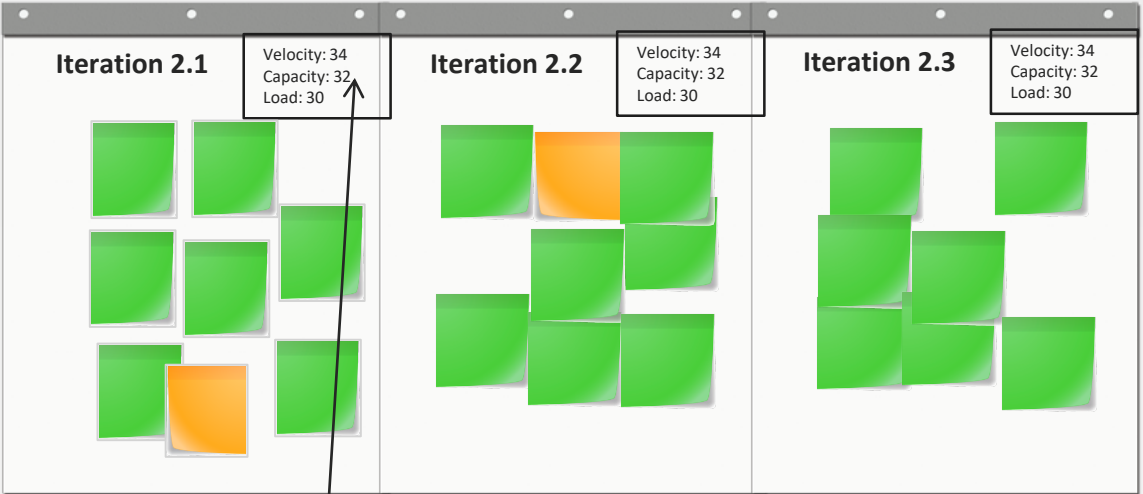
### **Prepare to present your plan**

# Team deliverables – details

## Iterations

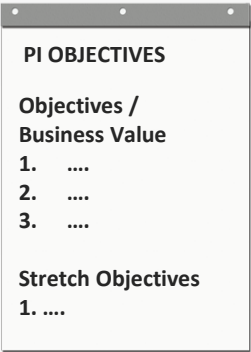


- ▶ If an epic has a dependency, put a red sticky on it describing the dependency. Put a check mark through it once the dependency has been addressed.
- ▶ If a risk is broader in nature, put it on the risk sheet
- ▶ If needed, allocate a percentage of capacity for unplanned activities like maintenance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



Velocity: historical  
Capacity: what can you handle?  
Load: # of story points

## Objectives



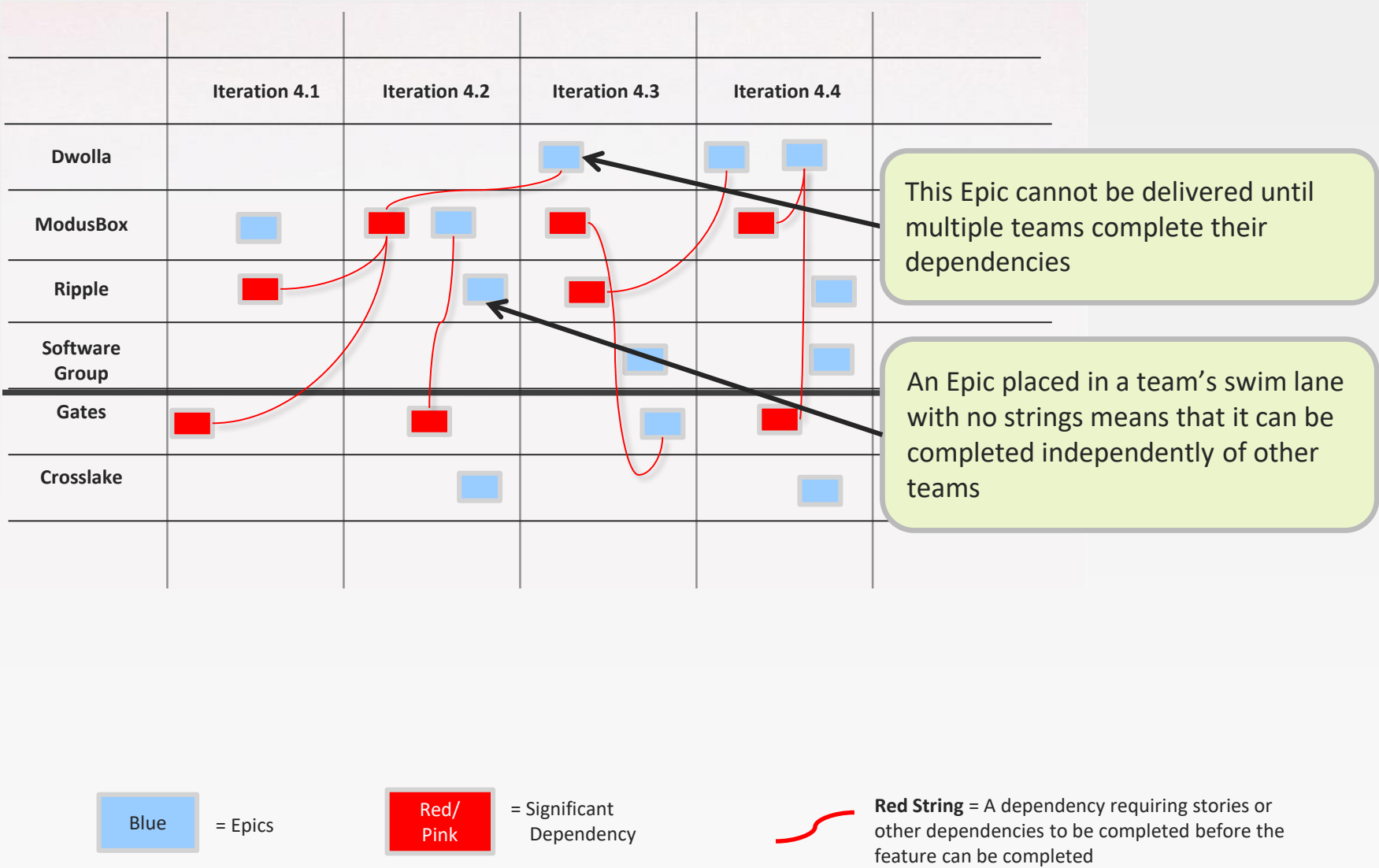
- ▶ PI Objectives should be written as “SMART” objectives
- ▶ Objectives are assigned business value during the second team breakout
- ▶ Stories supporting stretch objectives are included in the load calculation

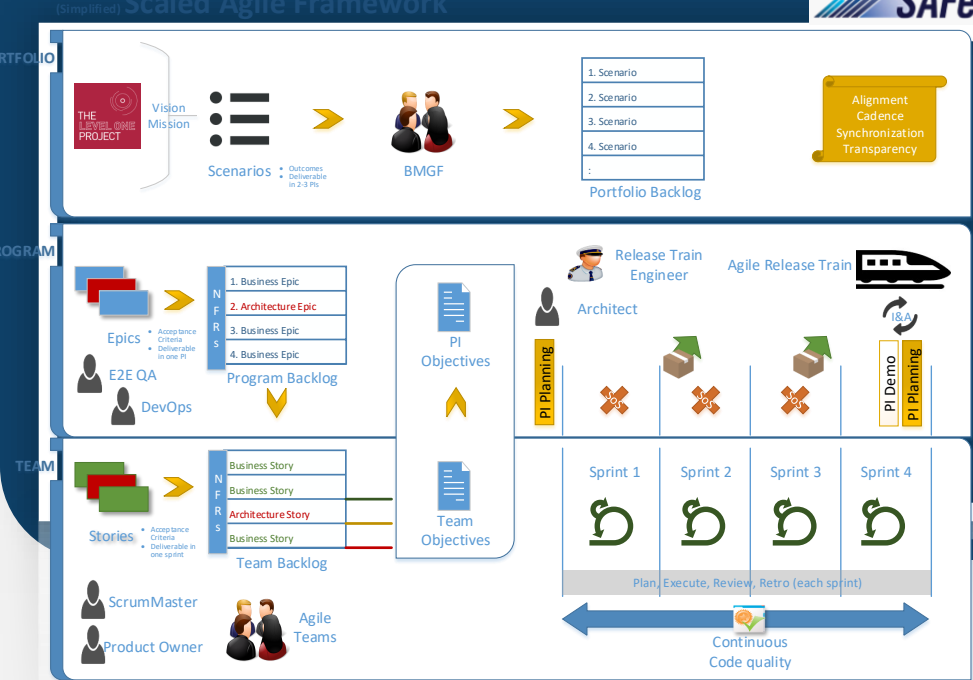
## Risks



- ▶ Program risks are those which need to be escalated to the program level. They will be captured and “ROAMed” after the final plan review.
- ▶ Team risks are those under the team’s control. They won’t be presented.

# Program board - Epic delivery, dependencies and risks





# Putting it into Practice