Contents

[Agile Release Train 5](#_Toc11847981)

[Details 5](#_Toc11847982)

[Organization 7](#_Toc11847983)

[Agile Teams Power the Train 8](#_Toc11847984)

[Define the ART 10](#_Toc11847985)

[Develop on Cadence 11](#_Toc11847986)

[ART Execution, DevOps, and Continuous Delivery 13](#_Toc11847987)

[ARTs Deliver All or Part of a Value Stream 15](#_Toc11847988)

[Lean-Agile Mindset 16](#_Toc11847989)

[Details 17](#_Toc11847990)

[Thinking Lean with the SAFe House of Lean 17](#_Toc11847991)

[The Goal – Value 18](#_Toc11847992)

[Pillar 1 – Respect for People and Culture 18](#_Toc11847993)

[Pillar 2 – Flow 19](#_Toc11847994)

[Pillar 3 – Innovation 19](#_Toc11847995)

[Pillar 4 – Relentless Improvement 20](#_Toc11847996)

[Foundation – Leadership 20](#_Toc11847997)

[Embracing Agility with the Agile Manifesto 20](#_Toc11847998)

[Product Owner 23](#_Toc11847999)

[**Find a Course**: 23](#_Toc11848000)

[Details 24](#_Toc11848001)

[Responsibilities 24](#_Toc11848002)

[Content Authority 26](#_Toc11848003)

[Fan-Out Model of Product Manager, Product Owner, and Agile Teams 27](#_Toc11848004)

[Core Values 27](#_Toc11848005)

[Details 27](#_Toc11848006)

[Alignment 28](#_Toc11848007)

[Built-in Quality 29](#_Toc11848008)

[Transparency 30](#_Toc11848009)

[Program Execution 31](#_Toc11848010)

[Leadership is Required 31](#_Toc11848011)

[Core Values **Error! Bookmark not defined.**](#_Toc11848012)

[Details **Error! Bookmark not defined.**](#_Toc11848013)

[Alignment **Error! Bookmark not defined.**](#_Toc11848014)

[Built-in Quality **Error! Bookmark not defined.**](#_Toc11848015)

[Transparency **Error! Bookmark not defined.**](#_Toc11848016)

[Program Execution **Error! Bookmark not defined.**](#_Toc11848017)

[Leadership is Required **Error! Bookmark not defined.**](#_Toc11848018)

[ScrumXP **Error! Bookmark not defined.**](#_Toc11848019)

[**Find a Course**: **Error! Bookmark not defined.**](#_Toc11848020)

[Details 32](#_Toc11848021)

[Product Owner 33](#_Toc11848022)

[Scrum Master 33](#_Toc11848023)

[The Scrum Process 33](#_Toc11848024)

[Building Quality In 36](#_Toc11848025)

[Agile Teams Using ScrumXP Are on the Train 36](#_Toc11848026)

[Leadership of Agile Teams 37](#_Toc11848027)

[Learn More 37](#_Toc11848028)

[Principle #8 – Unlock the intrinsic motivation of knowledge workers 38](#_Toc11848029)

[Leverage the Systems View 38](#_Toc11848030)

[Understand the Role of Compensation 39](#_Toc11848031)

[Provide Autonomy with Purpose, Mission, and Minimum Possible Constraints 39](#_Toc11848032)

[Create an Environment of Mutual Influence 40](#_Toc11848033)

[Principle #6 – Visualize and limit WIP, reduce batch sizes, and manage queue lengths 41](#_Toc11848034)

[Visualize and Limit WIP 41](#_Toc11848035)

[Reduce Batch Size 42](#_Toc11848036)

[Manage Queue Lengths 43](#_Toc11848037)

[Summary 44](#_Toc11848038)

[Section 2 44](#_Toc11848039)

[Sections copied above 45](#_Toc11848040)

[Section 3 45](#_Toc11848041)

[Story 45](#_Toc11848042)

[Details 45](#_Toc11848043)

[Sources of Stories 46](#_Toc11848044)

[User Stories 47](#_Toc11848045)

[Enabler Stories 48](#_Toc11848046)

[Writing Good Stories 49](#_Toc11848047)

[Estimating Stories 51](#_Toc11848048)

[Splitting Stories 54](#_Toc11848049)

[Stories in the SAFe Requirements Model 55](#_Toc11848050)

[Section 4 56](#_Toc11848051)

[Continuous Deployment 57](#_Toc11848052)

[Details 58](#_Toc11848053)

[The Four Sub-dimensions of Continuous Deployment 58](#_Toc11848054)

[Deploy to Production 59](#_Toc11848055)

[Section 5 63](#_Toc11848056)

[PI Planning 63](#_Toc11848057)

[**Find a Course**: 63](#_Toc11848058)

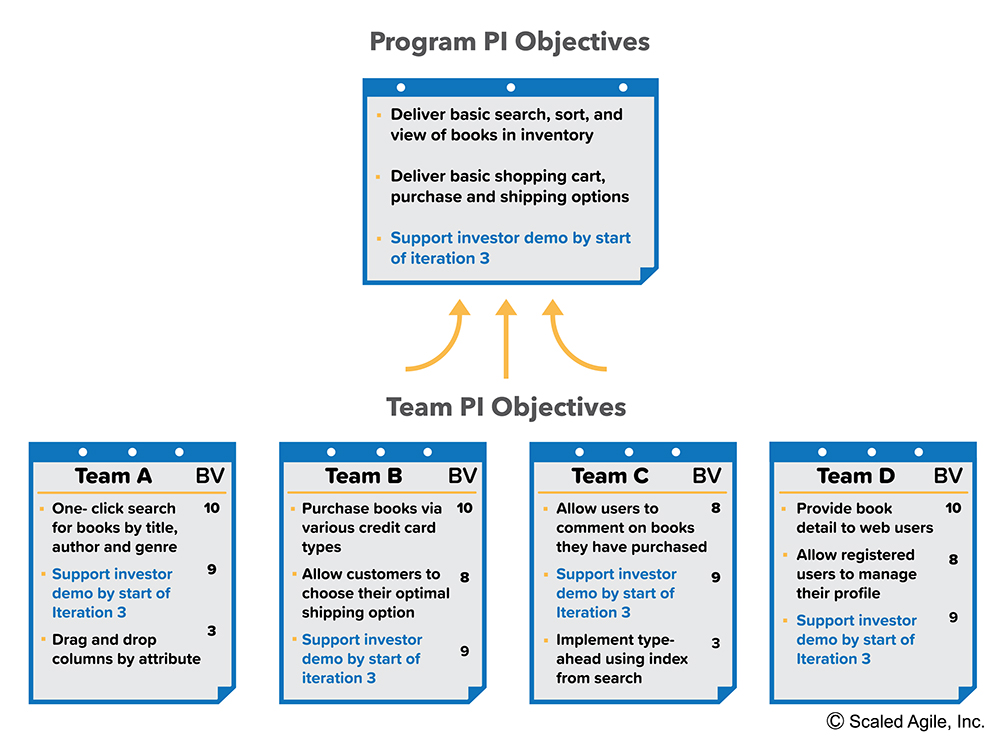
[Details 64](#_Toc11848059)

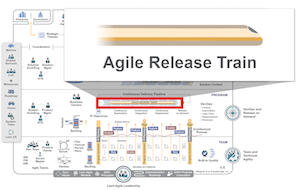
[Business Benefits of PI Planning 65](#_Toc11848060)

[Inputs and Outputs of PI Planning 65](#_Toc11848061)

[Preparation 66](#_Toc11848062)

[Standard Agenda 67](#_Toc11848063)

[Solution Train PI Planning 74](#_Toc11848064)

[](https://www.scaledagileframework.com/)

The more alignment you have, the more autonomy you can grant. The one enables the other.

—Stephen Bungay, Author and Strategy Consultant

Agile Release Train

The *Agile Release Train (ART)* is a long-lived team of [Agile teams](https://www.scaledagileframework.com/agile-teams/), which, along with other stakeholders, incrementally develops, delivers, and where applicable operates, one or more solutions in a value stream.

Details

Agile Release Trains align teams to a common business and technology mission. Each is a virtual organization (typically 50 – 125 people) that plans, commits, develops and deploys together. ARTs are organized around the enterprise’s significant [Value Streams](https://www.scaledagileframework.com/value-streams/) and exist solely to realize the promise of that value by building [Solutions](https://www.scaledagileframework.com/solution/) that deliver benefit to the end user.

ARTs are cross-functional and have all the capabilities—software, hardware, firmware, and other—needed to define, implement, test, deploy, release, and where applicable, operate solutions. An ART delivers a *continuous flow of value*, as shown in Figure 1.

Figure 1. The long-lived Agile Release Train

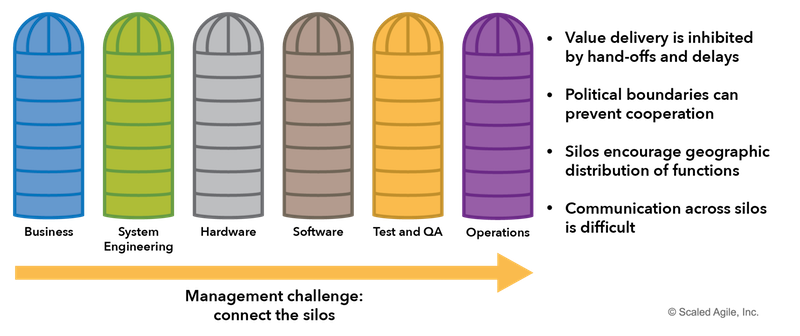
ARTs operate on a set of common principles:

* **The schedule is fixed**– The train departs the station on a known, reliable schedule, as determined by the chosen PI cadence. If a [Feature](https://www.scaledagileframework.com/features-and-capabilities/) misses a timed departure, it can catch the next one.
* **A new system increment every two weeks**– Each train delivers a new system increment every two weeks. The [System Demo](https://www.scaledagileframework.com/system-demo/) provides a mechanism for evaluating the working system, which is an integrated increment from all the teams.
* **The PI timebox is fixed**– All teams on the train are synchronized to the same PI length (typically 8 – 12 weeks) and have common iteration start/end dates and duration.
* **The train has a known velocity**– Each train can reliably estimate how much cargo (new features) can be delivered in a PI.
* **Agile Teams**– Agile Teams embrace the ‘Agile Manifesto’ and the SAFe values and principles. They apply Scrum, Extreme Programming (XP), Kanban, and other [Built-In Quality](https://www.scaledagileframework.com/built-in-quality/) practices.
* **Dedicated people**– Most people needed by the ART are dedicated full time to the train, regardless of their functional reporting structure.
* **Face-to-face PI Planning**– The ART plans its work at periodic, largely face-to-face [PI Planning](https://www.scaledagileframework.com/pi-planning/) events.
* **Innovation and Planning (IP)**– IP iterations provide a guard band (buffer) for estimating and a dedicated time for PI planning, innovation, continuing education, and infrastructure work.
* **Inspect and Adapt (I&A)**– An [I&A](https://www.scaledagileframework.com/inspect-and-adapt/) event is held at the end of every PI. The current state of the solution is demonstrated and evaluated. Teams and management then identify improvement backlog items via a structured, problem-solving workshop.
* **Develop on Cadence. Release on Demand**– ARTs apply cadence and synchronization to help manage the inherent variability of research and development. However, [releasing](https://www.scaledagileframework.com/release-on-demand/) is typically decoupled from the development cadence. ARTs can release a solution, or elements of a solution, at any time, subject to governance and release criteria.

Additionally, in larger value streams, multiple ARTs collaborate to build larger solutions via a [Solution Train](https://www.scaledagileframework.com/solution-train/). Some ART stakeholders participate in Solution Train events, including the [Solution Demo](https://www.scaledagileframework.com/solution-demo/) and [Pre- and Post-PI Planning](https://www.scaledagileframework.com/pre-and-post-pi-planning/).

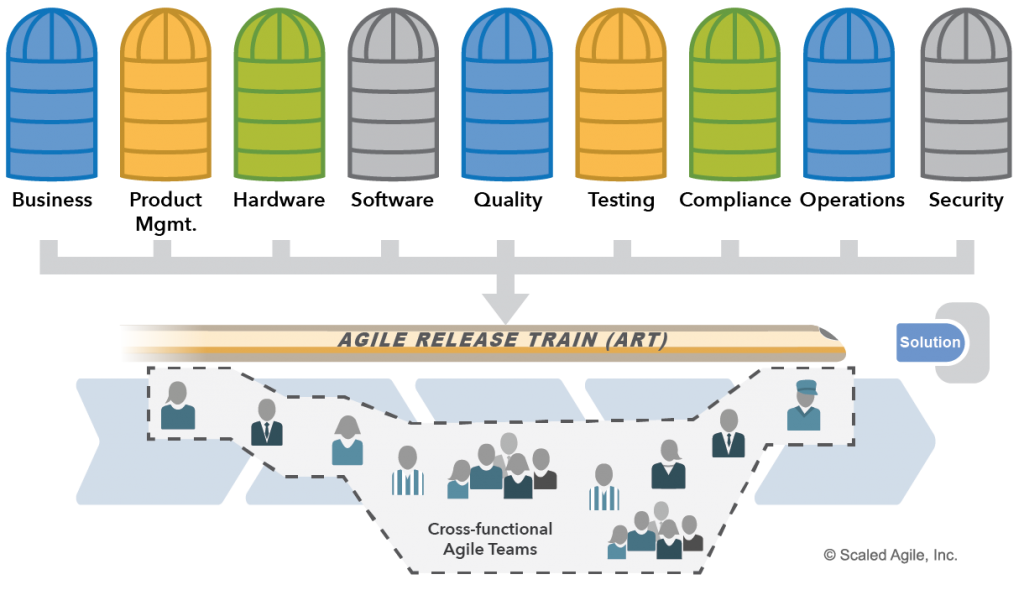
Organization

ARTs are typically virtual organizations that have all the people needed to define, deliver, and operate the solution. This new organization breaks down the traditional functional silos that may exist, as shown in Figure 2.

Figure 2. Traditional functional organization

In such a functional organization, developers work with developers, and testers work with other testers, architects and systems engineers work with each other, and operations work by themselves. While there are reasons why organizations have evolved that way, the value doesn’t flow quickly, as it must cross all the silos. The daily involvement of managers and project managers is necessary to move the work across. As a result, progress is slow, and handoffs and delays rule.

Instead, the ART applies [systems thinking](https://www.scaledagileframework.com/assume-variability-preserve-options/) (SAFe Principle #2)and builds a cross-functional organization that is optimized to facilitate the flow of value from ideation through deployment and release, and into operations, as Figure 3 illustrates.

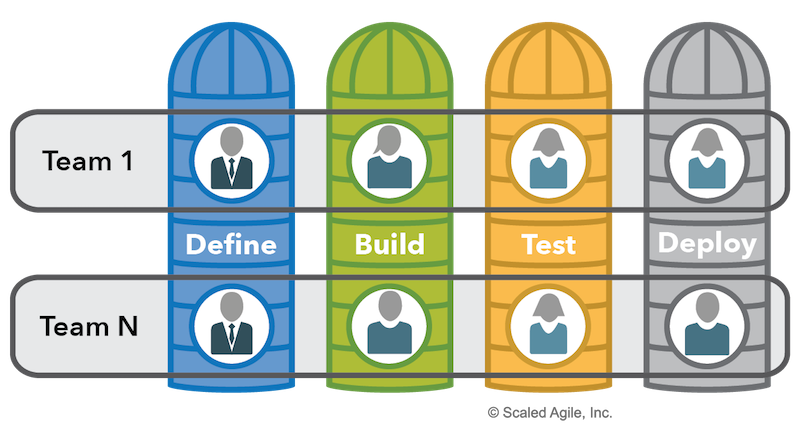
Figure 3. Agile Release Trains are fully cross-functional

Together, this fully cross-functional organization—whether physical (direct organizational reporting) or virtual (line of reporting is unchanged)—has everyone and everything it needs to define, deliver, and operate solutions. It is self-organizing and self-managing. This creates a far leaner organization; one where traditional daily task and project management is no longer required. Value flows more quickly, with a minimum of overhead.

Agile Teams Power the Train

ARTs include the teams that define, build, and test features and components, as well as those that deploy, release, and operate the solution. Individual teams have a choice of Agile practices, based primarily on Scrum, XP, and Kanban. Software quality practices include architecture & design quality, code quality, systems quality, and release quality practices (see [Built-In Quality](https://www.scaledagileframework.com/built-in-quality/)). Hardware quality is supported by exploratory early iterations, frequent system-level integration, design verification, modeling, and [Set-Based Design](https://www.scaledagileframework.com/set-based-design/). Agile architecture supports software and hardware quality.

Each Agile team has five to eleven dedicated individual contributors, covering all the roles necessary to build a quality increment of value for an iteration. Teams can deliver software, hardware, and any combination. And of course, Agile teams within the ART are themselves cross-functional, as shown in Figure 4.

Figure 4. Agile teams are cross-functional

Critical Team Roles

Each Agile team has dedicated individual contributors, covering all the roles necessary to build a quality increment of value for an iteration. Most SAFe teams apply a [ScrumXP](https://www.scaledagileframework.com/scrumxp/) and [Kanban](https://www.scaledagileframework.com/team-kanban/) hybrid, with the three primary Scrum roles:

* [**Scrum Master**](https://www.scaledagileframework.com/scrum-master/) – The Scrum Master is the servant leader for the team, facilitating meetings, fostering Agile behavior, removing impediments, and maintaining the team’s focus.
* [**Product Owner**](https://www.scaledagileframework.com/product-owner/) – The Product Owner owns the team backlog, acts as the Customer for developer questions, prioritizes the work, and collaborates with Product Management to plan and deliver solutions.
* [**Development Team**](https://www.scaledagileframework.com/dev-team/) – The Development Team has three to nine dedicated individual contributors, covering all the roles necessary to build a quality increment of value for an iteration.

Critical Program Roles

In addition to the Agile teams, the following program-level roles help ensure successful execution of the ART:

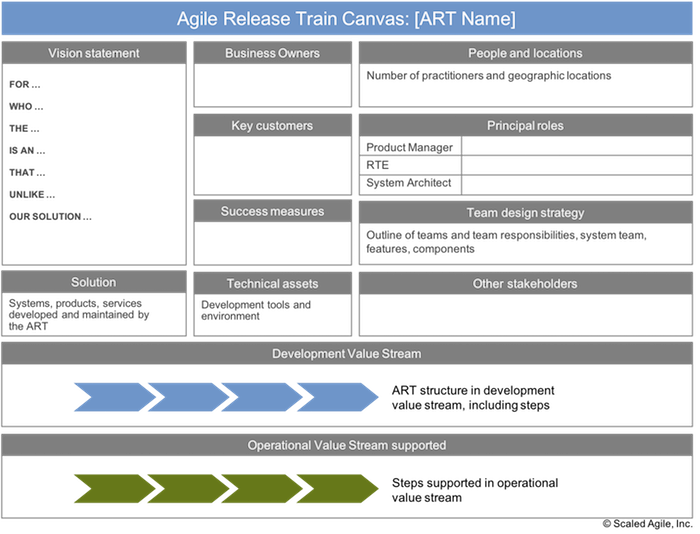
* [Release Train Engineer (RTE)](https://www.scaledagileframework.com/release-train-engineer-and-solution-train-engineer/) is a servant leader who facilitates program-level execution, impediment removal, risk and dependency management, and continuous improvement.
* [Product Management](https://www.scaledagileframework.com/product-and-solution-management/) is responsible for ‘what gets built,’ as defined by the [Vision](https://www.scaledagileframework.com/vision/), [Roadmap](https://www.scaledagileframework.com/roadmap/), and new [Features](https://www.scaledagileframework.com/features-and-capabilities/) in the [Program Backlog](https://www.scaledagileframework.com/program-and-solution-backlogs/). They work with customers and Product Owners to understand and communicate their needs, and also participate in [Solution](https://www.scaledagileframework.com/solution/) validation.
* [System Architect/Engineer](https://www.scaledagileframework.com/system-and-solution-architect-engineering/) is an individual or team that defines the overall architecture of the system. They work at a level of abstraction above the teams and components and define [Nonfunctional Requirements (NFRs)](https://www.scaledagileframework.com/nonfunctional-requirements/), major system elements, subsystems, and interfaces.
* [Business Owners](https://www.scaledagileframework.com/business-owners/) are key stakeholders of the ART and have ultimate responsibility for the business outcomes of the train.
* [Customers](https://www.scaledagileframework.com/customer/) are the ultimate buyers of the solution.

In addition to these critical program roles, the following functions play an essential part in ART success:

* A [System Team](https://www.scaledagileframework.com/system-team/) typically assists in building and maintaining the development, continuous integration, and test environments.
* [Shared Services](https://www.scaledagileframework.com/shared-services/) are specialists—for example, data security, information architects, database administrators (DBAs)—that are necessary for the success of an ART but cannot be dedicated to a specific train.

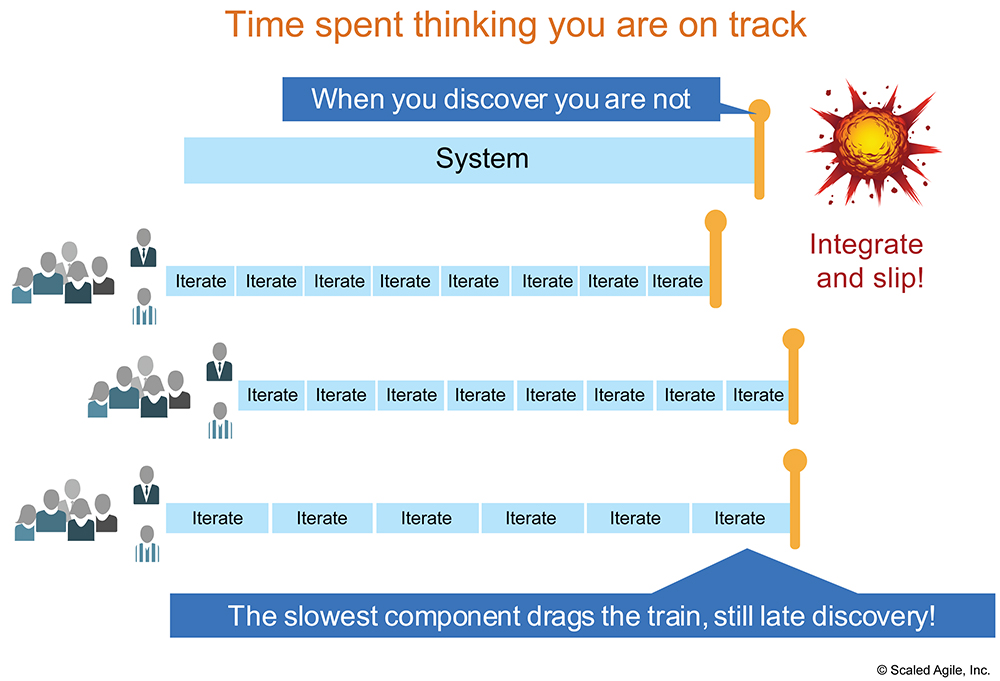
Define the ART

The parameters and boundaries of the ART, as well as its stakeholders, and relations to the value streams can be captured and summarized in the ‘ART canvas’, as Figure 5 shows.

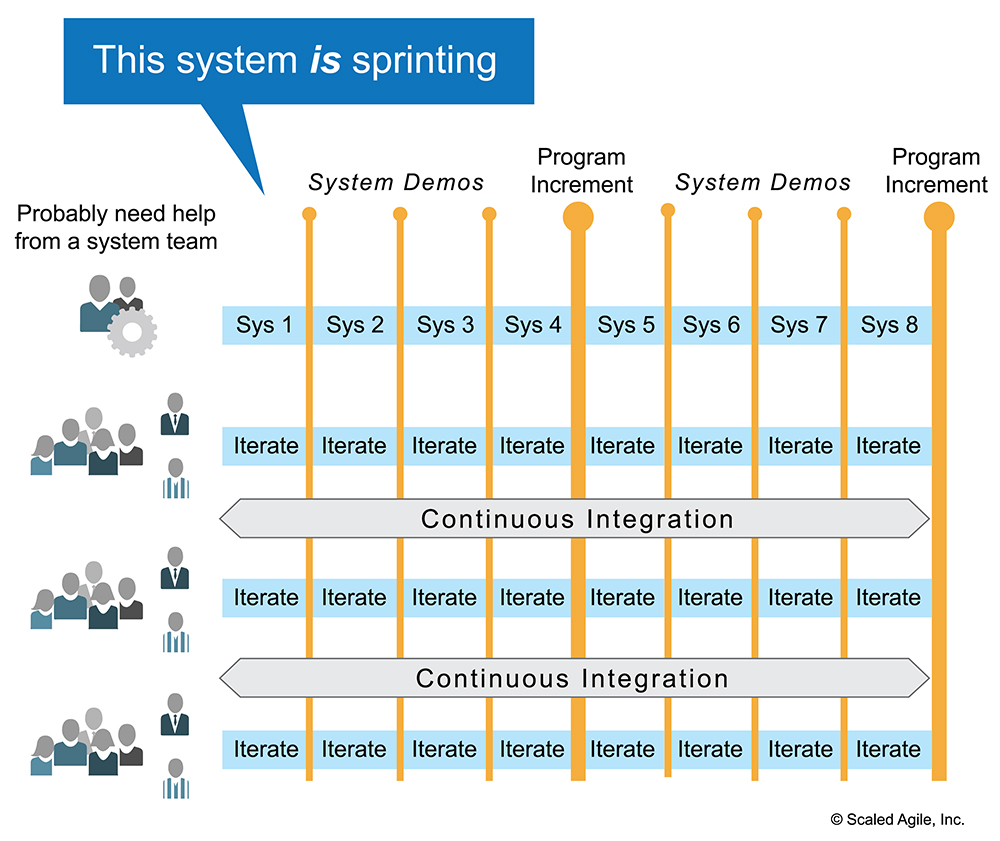
Figure 5. The ART Canvas

Develop on Cadence

ARTs also address one of the most common problems with traditional Agile development: Teams working on the same solution operate independently and asynchronously. That makes it extremely difficult to integrate the full system routinely. In other words, ‘The teams are sprinting, but the system isn’t.’ This increases the risk of late discovery of issues and problems, as shown in Figure 6.

Figure 6. Asynchronous Agile development

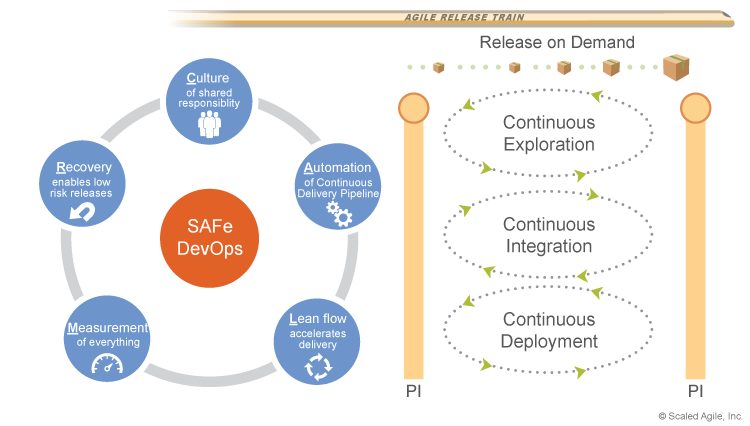
Instead, the ART applies cadence and synchronization to assure that the system is sprinting as a whole, as shown in Figure 7.

Figure 7. Aligned development: this system is sprinting

Cadence and synchronization assure that the focus is constantly on the evolution and objective assessment of the full system, rather than its individual elements. The system demo, which occurs at the end of the iteration, provides the objective evidence that the system is sprinting.

ART Execution, DevOps, and Continuous Delivery

ARTs aim to continuously deliver value to their Customer. This is supported by a [Continuous Delivery Pipeline](https://www.scaledagileframework.com/continuous-delivery-pipeline/), which contains the workflows, activities, and automation needed to provide the availability of new features release. Figure 8 illustrates how these processes run concurrently and continuously, supported by the ART’s [DevOps](https://www.scaledagileframework.com/devops/) capabilities.

Figure 8. Continuous Exploration, Continuous Integration, and Continuous Deployment are continuous, concurrent, and supported by DevOps capabilities

Each ART builds and maintains (or shares) a pipeline with the assets and technologies needed to deliver solution value as independently as possible. The first three elements of the pipeline work together to support the deployment of very of small batches of new functionality, which are released as the market demands.

* [Continuous Exploration](https://www.scaledagileframework.com/continuous-exploration/) is the process of constantly exploring market and user needs, and defining a [Vision](https://www.scaledagileframework.com/vision/), [Roadmap](https://www.scaledagileframework.com/roadmap/), and set of [hypotheses](https://www.scaledagileframework.com/features-and-capabilities/) to address those needs.
* [Continuous Integration](https://www.scaledagileframework.com/continuous-integration/) is the process of taking features from the program backlog and developing, testing, integrating, and validating them in a staging environment where they are ready for deployment and release.
* [Continuous Deployment](https://www.scaledagileframework.com/continuous-deployment/) is the process that takes validated features from continuous integration and deploys them into the production environment, where they’re tested and readied for release.
* [Release on Demand](https://www.scaledagileframework.com/release-on-demand/) is the process of delivering the value to the end user, measuring the results of the hypotheses and learning from them, as well as operating the solutions.

Development and management of the continuous delivery pipeline are supported by DevOps, a capability of every ART. SAFe’s approach to DevOps uses the acronym ‘CALMR’ to reflect the aspects of Culture, Automation, Lean flow, Measurement, and Recovery.

Flow through the system is visualized, managed, and measured by the [Program Kanban](https://www.scaledagileframework.com/program-and-solution-kanbans/).

ARTs Deliver All or Part of a Value Stream

The organization of an ART determines who will plan and work together, as well as what products, services, features, or components the train will deliver. Organizing ARTs is part of the ‘art’ of SAFe. This is covered extensively in the [Implementation Roadmap](https://www.scaledagileframework.com/implementation-roadmap/) article series, particularly in ‘[Identify Value Streams and ARTs](https://www.scaledagileframework.com/identify-value-streams-and-arts/)‘ and ‘[Create the Implementation Plan](https://www.scaledagileframework.com/create-the-implementation-plan/).’

One primary consideration is size. Effective ARTs typically consist of 50 – 125 people. The upper limit is based on Dunbar’s number, which suggests a limit on the number of people with whom one can form effective, stable social relationships. The lower limit is based mostly on empirical observation. However, trains with fewer than 50 people can still be very effective, and provide many advantages over legacy Agile practices for coordinating Agile teams.

Given the size constraints, there are two main patterns of ART design (Figure 9):

* Smaller value streams can be implemented by a single ART
* A larger value stream must be supported by multiple ARTs

Figure 9. ARTs realize all or part of a value stream

In the latter case, enterprises apply the elements and practices of the [Large Solution Level](https://www.scaledagileframework.com/large-solution-level/) and create a Solution Train to help coordinate the contributions of ARTs and [Suppliers](https://www.scaledagileframework.com/supplier/) to deliver some of the world’s largest systems.

[](https://www.scaledagileframework.com/)

It all starts with a Lean-Agile Mindset.

—SAFe authors

Lean-Agile Mindset

The *Lean-Agile Mindset* is the combination of beliefs, assumptions, and actions of SAFe leaders and practitioners who embrace the concepts of the Agile Manifesto and Lean thinking. It’s the personal, intellectual, and leadership foundation for adopting and applying SAFe principles and practices.

SAFe is based on three primary bodies of knowledge: Agile development, Lean product development [1], and systems thinking.

Agile development provides the tools needed to empower and engage teams to achieve unprecedented levels of productivity, quality, and engagement. Nevertheless, a broader and deeper Lean-Agile mindset is required to support Lean and Agile development at scale across the entire enterprise. Therefore, there are two primary aspects of a Lean-Agile Mindset:

* **Thinking Lean** – Lean thinking is illustrated by the SAFe House of Lean (Figure 1). In this graphical depiction, the roof of the house represents the goal of delivering value. The pillars embody respect for people and culture, flow, innovation, and relentless improvement to support the goal. Lean leadership provides the foundation on which everything stands.
* **Embracing agility** – SAFe is built entirely on the skills, aptitude, and capabilities of Agile teams and their leaders. The Agile Manifesto provides a value system and set of principles that are fundamental to the mindset for successful Agile development.

Understanding and applying this knowledge helps create the Lean-Agile mindset, part of a new management approach and an enhanced company culture. It provides the leadership needed to drive a successful SAFe transformation, helping individuals and businesses achieve their goals.

Details

Thinking Lean with the SAFe House of Lean

Initially derived from Lean manufacturing, the principles and practices of Lean thinking as applied to software, product, and systems development are now deep and extensive [2]. For example, Ward [3], Reinertsen [4], Poppendieck,[5], Leffingwell [6], and others have described aspects of Lean thinking, placing many of the core principles and practices in a product development context. Along with these, we present the SAFe House of Lean, as illustrated in Figure 1, inspired by houses of Lean from Toyota and others.

Figure 1. The SAFe House of Lean

The Goal – Value

The goal of Lean is to deliver the maximum customer value in the shortest sustainable lead time while providing the highest possible quality to [Customers](https://www.scaledagileframework.com/customer/) and society as a whole. High morale, safety, and customer delight are additional goals and benefits.

Pillar 1 – Respect for People and Culture

A Lean-Agile approach doesn’t implement itself or perform any real work—people do. Respect for people and culture is a basic human need. When treated with respect, people are empowered to evolve their practices and improve*.* Management challenges people to change and may steer them toward better ways of working. However, it’s the teams and individuals who learn problem-solving and reflection skills, and are accountable for making the appropriate improvements [1].

The driving force behind this new behavior is culture, which requires the enterprise and its leaders to change first. The principle of respect for people and culture also extends to relationships with [Suppliers](https://www.scaledagileframework.com/supplier/), partners, customers, and the broader community that supports the [Enterprise](https://www.scaledagileframework.com/enterprise/).

Where there is an urgency for positive change, improvement of culture is possible. First, understand and implement the SAFe values and principles. Second, deliver winning results. Eventually, the culture will change naturally.

Pillar 2 – Flow

The key to successfully executing SAFe is to establish a continuous flow of work that supports incremental value delivery, based on constant feedback and adjustment. Continuous flow enables faster value delivery, effective [Built-In Quality](https://www.scaledagileframework.com/built-In-quality/) practices, relentless improvement, and evidence-based governance.

The principles of flow are an essential part of the Lean-Agile mindset. These include understanding the full [Value Stream](https://www.scaledagileframework.com/value-streams/), visualizing and limiting Work in Process (WIP), and reducing batch sizes and managing queue lengths. Additionally,  Lean focus on identifying and continuously removing delays and waste (non-value-added activities).

Lean-Agile principles provide a better understanding of the system development process, incorporating new thinking, tools, and techniques that leaders and teams can use to move from a phase-gated approach to DevOps and a Continuous Delivery Pipeline that extends flow to the entire value delivery process.

# Pillar 3 – Innovation

Flow builds a solid foundation for value delivery. But without innovation, both product and process will steadily decline. To support this critical part of the SAFe House of Lean, [Lean-Agile Leaders](https://www.scaledagileframework.com/lean-agile-leadership/) engage in the following practices:

* Get out of the office and into the actual workplace where the value is produced, and products are created and used (known as *gemba*). As Taiichi Ohno put it, “No useful improvement was ever invented at a desk.”
* Provide time and space for people to be creative, enabling purposeful innovation, which can rarely occur in the presence of 100 percent utilization and daily firefighting. SAFe’s [Innovation and Planning Iteration](https://www.scaledagileframework.com/innovation-and-planning-iteration/) is one such opportunity.
* Apply [Continuous Exploration](https://www.scaledagileframework.com/continuous-exploration/), the process of constantly exploring the market and user needs, and defining a Vision, Roadmap, and set of Features that address those needs.
* Apply [Innovation Accounting](https://www.scaledagileframework.com/guidance-applied-innovation-accounting-in-safe/) [7]. Establish nonfinancial, non-vanity [Metrics](https://www.scaledagileframework.com/metrics/) that provide fast feedback for innovation.
* Validate the innovation with customers, then pivot without mercy or guilt when the hypothesis needs to change.

Pillar 4 – Relentless Improvement

The fourth pillar, relentless improvement, encourages learning and growth through continuous reflection and process enhancements. A constant sense of competitive danger drives the company to pursue improvement opportunities aggressively. Leaders and teams do the following:

* Optimize the whole, not the parts, of both the organization and the development process
* Consider facts carefully, then act quickly
* Apply Lean tools and techniques to determine the root cause of inefficiencies and apply effective countermeasures rapidly
* Reflect at key milestones to openly identify and address the shortcomings of the process at all levels

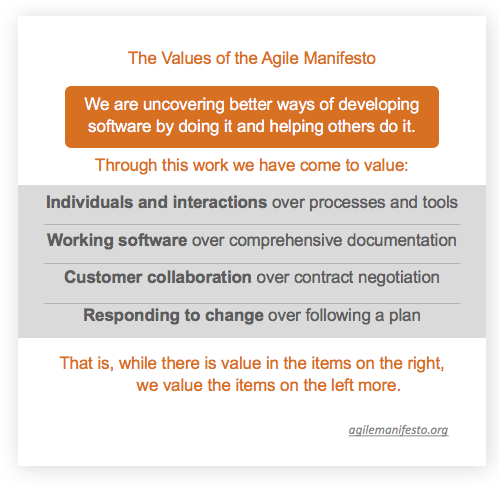
Foundation – Leadership

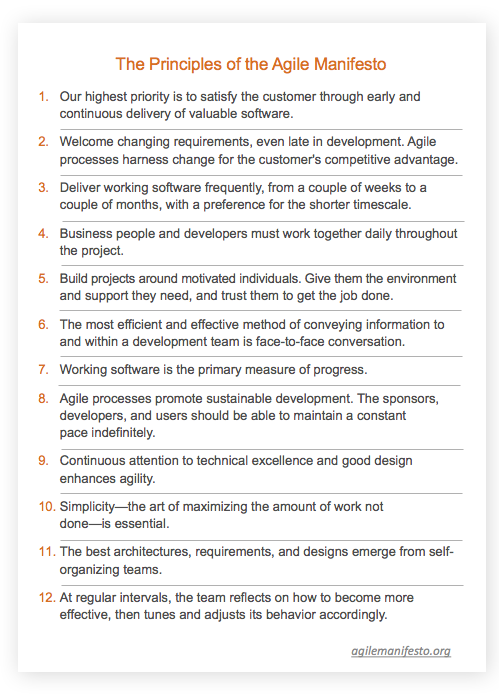
The foundation of Lean is leadership, a key enabler for team success. The ultimate responsibility for the successful adoption of the Lean-Agile approach lies with the enterprise’s managers, leaders, and executives. According to Deming, “Such a responsibility cannot be delegated” to Lean-Agile champions, working groups, a Program Management Office (PMO), process teams, outside consultants, or any other party [8]. Therefore, Leaders must be trained in these new and innovative ways of thinking and exhibit the principles and behaviors of Lean-Agile leadership.

Lean thinking is similar to, but somewhat different than, Agile. It was initially introduced as a team-based process that tended to exclude managers. Unfortunately, that doesn’t scale. In Lean-Agile development, by contrast, managers become leaders who embrace the values of Lean, are competent in the basic practices, proactively eliminates impediments, and take an active role in driving organizational change and facilitating unrelenting improvement.

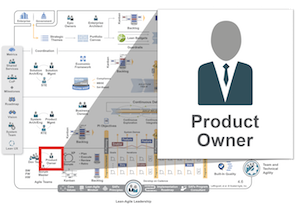
Embracing Agility with the Agile Manifesto

In the 1990s, responding to the many challenges of waterfall processes, some lighter-weight and more iterative development methods emerged. In 2001, many of the leaders of these frameworks came together in Snowbird, Utah. While there were differences of opinion on the specific merits of one method over another, the attendees agreed that their shared values and beliefs dwarfed the differences. The result was a Manifesto for Agile Software Development—a turning point that clarified the new approach and started to bring the benefits of these innovative methods to the whole development industry [9]. The Manifesto consists of the value statement shown in Figure 2 and a set of principles shown in Figure 3.

[](http://agilemanifesto.org/)Figure 2. Values of the Agile Manifesto

[](http://agilemanifesto.org/)Figure 3. Principles of the Agile Manifesto

Along with the various practices, the Agile Manifesto provides the foundation for empowered, self-organizing teams. SAFe extends this to teams-of-Agile-teams, applying Lean thinking to understand and relentlessly improve the systems that support their critical work.

*[](https://www.scaledagileframework.com/)*

Business people and developers must work together daily throughout the project.

—Agile Manifesto

Product Owner

**Find a Course**:

Top of Form

  
  
Go

Bottom of Form

[](https://www.scaledagile.com/training/calendar/?three_months=true&course_id=SAFe+Product+Owner/Product+Manager)

The *Product Owner (PO)* is a member of the [Agile Team](https://www.scaledagileframework.com/agile-teams/) responsible for defining [Stories](https://www.scaledagileframework.com/story/) and prioritizing the [Team Backlog](https://www.scaledagileframework.com/team-backlog/) to streamline the execution of program priorities while maintaining the conceptual and technical integrity of the [Features](https://www.scaledagileframework.com/features-and-capabilities/) or components for the team.

The PO has a significant role in quality control and is the only team member empowered to accept stories as done. For most enterprises moving to Agile, this is a new and critical role, typically translating into a full-time job, requiring one PO to support each Agile team (or, at most, two teams).

This role has significant relationships and responsibilities outside the local team, including working with [Product Management](https://www.scaledagileframework.com/product-and-solution-management/), who is responsible for the [Program Backlog](https://www.scaledagileframework.com/program-and-solution-backlogs/), to prepare for the [Program Increment (PI) Planning](https://www.scaledagileframework.com/pi-planning/) meeting.

Details

The PO is the member of the Agile team who serves as the [Customer](https://www.scaledagileframework.com/customer/) proxy responsible for working with Product Management and other stakeholders—including other POs—to define and prioritize stories in the team backlog. This allows the [Solution](https://www.scaledagileframework.com/solution/) to effectively address program priorities (features and [Enablers](https://www.scaledagileframework.com/enablers/)) while maintaining technical integrity. Ideally, the PO is collocated with the rest of the team, where they typically share management, incentives, and culture. But the PO also attends most relevant Product Management meetings about planning and [Program Backlog](https://www.scaledagileframework.com/program-and-solution-backlogs/)/[Vision](https://www.scaledagileframework.com/vision/) refinement.

Responsibilities

The PO fulfills the following duties:

Preparation and Participation in PI Planning

* As a member of the extended Product Management team, the PO is heavily involved in program backlog refinement and prep for PI planning and also plays a significant role in the planning event itself. Before the planning event, the PO updates the team backlog and typically reviews and contributes to the program vision, [Roadmap](https://www.scaledagileframework.com/roadmap/), and content presentations.
* During the event, the PO is involved with story definition, providing the clarifications necessary to assist the team with their story estimates and sequencing. The PO also drafts the team’s specific objectives for the upcoming PI.

Iteration Execution

* **Maintaining the team backlog** – With input from [System Architect/Engineering](https://www.scaledagileframework.com/system-and-solution-architect-engineering/) and other stakeholders, the PO has the primary responsibility for building, editing, and maintaining the team backlog. Consisting mostly of user stories, it also includes defects and enablers. Backlog items are prioritized based on user value, time, and other team dependencies determined in the PI planning meeting and refined during the PI.
* **Iteration Planning** – The PO reviews and reprioritizes the backlog as part of the prep work for [Iteration Planning](https://www.scaledagileframework.com/iteration-planning/), including coordination of dependencies with other POs. During the iteration planning meeting, the PO is the primary source for story detail and priorities and has the responsibility of accepting the final iteration plan.
* **Just-in-time story elaboration** – Most backlog items are elaborated into user stories for implementation. This may happen before the iteration, during iteration planning, or during the iteration. While any team member can write stories and acceptance criteria, the PO has the primary responsibility for maintaining the flow. It’s usually good to have approximately two iterations’ worth of stories ready in the team backlog at all times. More would create a queue, while less might inhibit flow.
* **Apply Behavior-Driven Development (BDD)** – POs collaborate with their team to details stories with acceptance criteria and acceptance tests. See the [BDD](https://www.scaledagileframework.com/behavior-driven-development/) article for more details.
* **Accepting stories** – The PO is the only team member who can accept stories as done. This includes validation that the story meets acceptance criteria and has the appropriate, persistent acceptance tests, and that it otherwise complies its Definition of Done (DoD). In so doing, the PO also assures a level of quality, focusing primarily on fitness for use.
* **Understand enabler work** – While POs are not expected to drive technological decisions, they are supposed to understand the scope of the upcoming enabler work and to collaborate with [System and Solution Architect/Engineering](https://www.scaledagileframework.com/system-and-solution-architect-engineering/) to assist with decision-making and sequencing of the critical technological infrastructures that will host the new business functionality. This can often be best accomplished by establishing a capacity allocation, as described in the team backlog article.
* **Participate in team demo and retrospective** – As the person responsible for requirements, POs have an essential role in the team demo, reviewing and accepting stories. They also participate in the [Iteration Retrospective](https://www.scaledagileframework.com/iteration-retrospective/), where the teams gather to improve their processes and are active in the [Agile Release Train’s (ART’s)](https://www.scaledagileframework.com/agile-release-train/) [Inspect and Adapt (I&A)](https://www.scaledagileframework.com/inspect-and-adapt/) workshop.

Program Execution

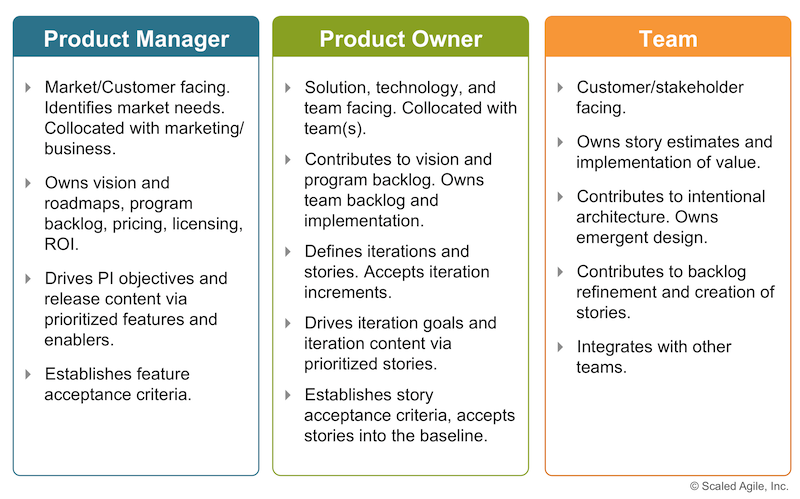
* Iterations and Agile teams serve a larger purpose; the frequent, reliable, and continuous release of value-added solutions. During each PI, the PO coordinates dependencies with other POs. This often occurs in weekly PO sync meetings (see the PI article for more information).
* The PO also has an instrumental role in producing the [System Demo](https://www.scaledagileframework.com/system-demo/) for program and [Value Stream](https://www.scaledagileframework.com/value-streams/) stakeholders.

Inspect and Adapt

* Teams address their larger impediments in the I&A workshop. There, the PO works across teams to define and implement improvement stories that will increase the velocity and quality of the program.
* The PI system demo is part of the I&A workshop. The PO has an instrumental role in producing the PI system demo for program stakeholders.
* To ensure that they will be able to show the most critical aspects of the solution to the stakeholders, POs also participate in the preparation of the PI system demo.

Content Authority

At scale, a single person cannot handle product and market strategy while also being dedicated to an Agile team. Since Product Management and the PO share the content authority for the program, it’s important to have a clear delineation of roles and responsibilities, as illustrated in Figure 1.

Figure 1. Release content governance

Fan-Out Model of Product Manager, Product Owner, and Agile Teams

Successful development is, in part, a game of numbers in the [Enterprise](https://www.scaledagileframework.com/enterprise/). Without the right number of people in the right roles, bottlenecks will severely limit velocity. Therefore, the number of Product Managers, POs, and Agile teams must be roughly in balance to steer the ART. Otherwise, the whole system will spend much of its time waiting for definition, clarification, and acceptance. Instead, SAFe recommends a fan-out model, as illustrated in Figure 2.

Figure 2. Fan-out model for Product Manager, PO, and Agile teams

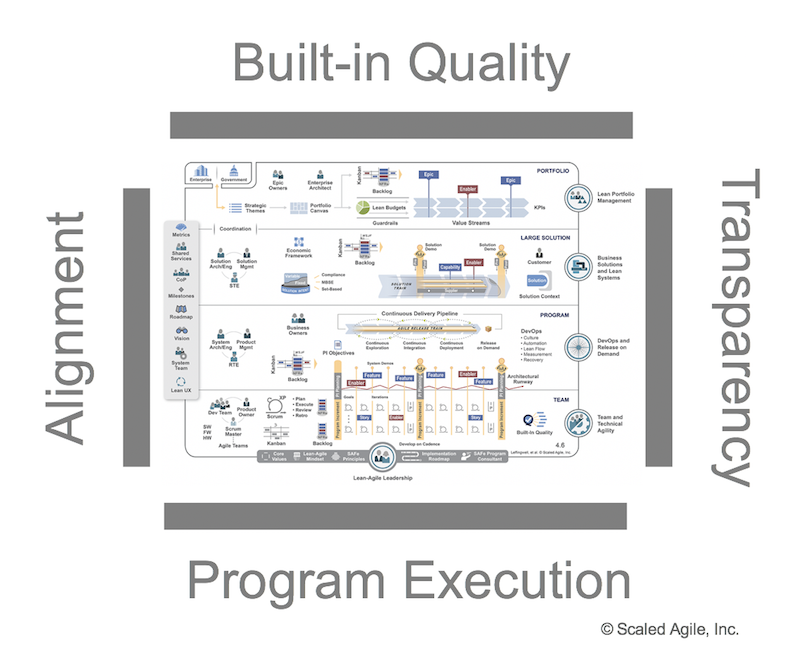
Each Product Manager can usually support up to four POs, each of whom can be responsible for the backlog of one or two Agile teams

Core Values

The four *Core Values* of *alignment*, *built-in quality*, *transparency*, and *program execution* represent the fundamental beliefs that are key to SAFe’s effectiveness. These guiding principles help dictate behavior and action for everyone who participates in a SAFe portfolio.

Details

SAFe is based on three primary bodies of knowledge: Agile Development, Lean Product Development, and Systems Thinking. That makes SAFe broad, deep and scaleable. But at its core, SAFe places the highest value on four things:  *alignment*,*built-in quality*, *transparency*, and*program execution.* These are illustrated in Figure 1 and described in the following sections.

Figure 1. SAFe’s four core values

Alignment

Like cars out of alignment, misaligned companies can develop serious problems. They are hard to steer, and they don’t respond well to changes in direction [1]. Even if it’s clear where everyone thinks they’re headed, the vehicle is unlikely to get them there.

Alignment is needed to keep pace with fast change, disruptive competitive forces, and geographically distributed teams. While empowered, [Agile Teams](https://www.scaledagileframework.com/agile-teams/) are good (even great), but the responsibility for strategy and alignment cannot rest with the combined opinions of the teams, no matter how good they are. Instead, alignment must rely on the [Enterprise](https://www.scaledagileframework.com/enterprise/) business objectives. Here are some of the ways how SAFe supports alignment:

* Alignment starts with the strategy and investment decisions at the [Portfolio](https://www.scaledagileframework.com/portfolio-level/) level and is reflected in [Strategic Themes](https://www.scaledagileframework.com/strategic-themes/) and the [Portfolio Backlog](https://www.scaledagileframework.com/portfolio-backlog/). In turn, this informs the [Vision](https://www.scaledagileframework.com/vision/), [Roadmap](https://www.scaledagileframework.com/roadmap/), and the backlogs at all level of SAFe. [Continuous Exploration](https://www.scaledagileframework.com/continuous-exploration/) gathers the inputs and perspectives from a diverse group of stakeholders and information sources to ensure that the items in the backlogs contain economically prioritized and refined work, ready for teams to implement. All work is visible, debated, resolved and transparent.
* Alignment is supported by clear lines of content authority, starting with the portfolio and then resting primarily with the [Product and Solution Management](https://www.scaledagileframework.com/product-and-solution-management/) roles, and extending to the [Product Owner](https://www.scaledagileframework.com/product-owner/) role.
* [PI Objectives](https://www.scaledagileframework.com/pi-objectives) and [Iteration Goals](https://www.scaledagileframework.com/iteration-goals/) are used to communicate expectations and commitments.
* Cadence and synchronization are applied to ensure that things stay in alignment, or that they drift only within reasonable economic and time boundaries.
* Architectures and user experience guidance and governance help ensure that the [Solution](https://www.scaledagileframework.com/solution/) is technologically sound, robust, and scalable.
* Economic prioritization keeps stakeholders engaged in a continuous, agreed-to, rolling-wave prioritization, based on the current context and evolving facts.

Alignment, however, does not imply or encourage top-down command and control. Alignment occurs when everyone is working towards a common direction. Indeed, Alignment enables empowerment, autonomy, and [Decentralized Decision-making](https://www.scaledagileframework.com/decentralize-decision-making/), allowing those who implement value to make better local decisions.

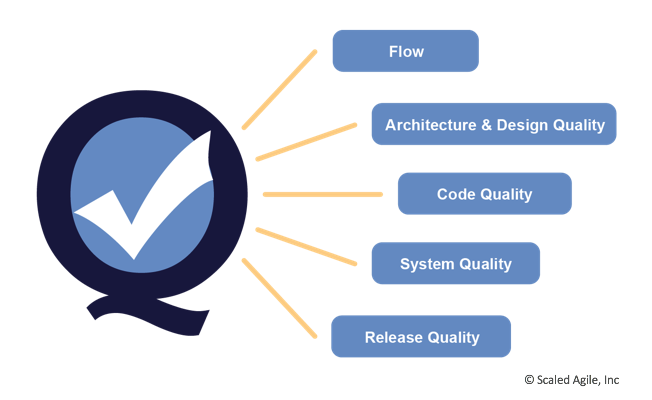
Built-in Quality

*“Inspection does not improve the quality, nor guarantee quality. Inspection is too late. The quality, good or bad, is already in the product. Quality cannot be inspected into a product or service; it must be built into it.”*

—W. Edwards Deming

Built-in Quality ensures that every element and every increment of the solution reflects quality standards throughout the development lifecycle. Quality is not “added later.” Building quality in is a prerequisite of Lean and flow—without it, the organization will likely operate with large batches of unverified, unvalidated work. Excessive rework and slower velocities are the likely results.

Also the bigger the system, the more important endemic quality is, so there can be no ambiguity about the importance of built-in quality in large-scale systems. It is mandatory. Built-in quality is a major foundational element of SAFe, and since it’s impossible to localize the quality discussion to every specific activity or artifact that affects quality, the SAFe [Built-in Quality](https://www.scaledagileframework.com/built-in-quality/) article organizes quality thinking around five specific aspects—Flow, Architecture and Design Quality, Code Quality, System Quality and Release Quality, as Figure 2 illustrates.

Figure 2. Five primary dimensions of Built-In Quality

In addition, this article is augmented by three Advanced Topic articles, [Agile Testing](https://www.scaledagileframework.com/agile-testing/), [Behavior-Driven Development](https://www.scaledagileframework.com/behavior-driven-development/) (BDD) and [Test-Driven Development](https://www.scaledagileframework.com/test-driven-development/). Together, this set of articles provides a fairly comprehensive starting point for achieving built-in quality in large systems.

Transparency

Solution development is hard. Things go wrong or do not work out as planned. Without openness, facts are obscure and decision-making is based on speculative assumptions and lack of data. No one can fix a secret.

To ensure openess—*trust* is needed. Trust exists when the business and development can confidently rely on another to act with integrity, particularly in times of difficulty. Without trust no one can build high-performance teams and programs, nor build (or rebuild) the confidence needed to make and meet reasonable commitments.  And without trust, working environments are a lot less fun and motivating.

Building trust takes time. Transparency is an enabler of trust, provided through several SAFe practices:

* Executives, [Lean Portfolio Management](https://www.scaledagileframework.com/lean-portfolio-management/), and other stakeholders can see the [Portfolio Kanban](https://www.scaledagileframework.com/portfolio-kanban/) and program backlogs, and they have a clear understanding of the [PI Objectives](https://www.scaledagileframework.com/pi-objectives/) for each [Agile Release Train](https://www.scaledagileframework.com/agile-release-train/) or [Solution Train](https://www.scaledagileframework.com/solution-train/).
* ARTs have visibility into the team’s backlogs, as well as other [Program Backlogs.](https://www.scaledagileframework.com/program-and-solution-backlogs/)
* Teams and programs commit to short-term, visible commitments that they routinely meet.
* [Inspect and Adapt](https://www.scaledagileframework.com/inspect-and-adapt/) occurs with all relevant stakeholders and creates backlog improvement items from lessons learned.
* Teams and [Agile Release Trains (ARTs)](https://www.scaledagileframework.com/agile-release-train/) can see portfolio business and enabler [Epics](https://www.scaledagileframework.com/epic/). They have visibility into new initiatives.
* Progress is based on objective measures of working solutions. ([Principle #5](https://www.scaledagileframework.com/base-milestones-on-objective-evaluation-of-working-systems/))
* Everyone can understand the velocity and WIP of the teams and programs; strategy and the ability to execute are visibly aligned.
* Programs execute reliably, as noted below.

Program Execution

Of course, none of the rest of SAFe matters if teams can’t execute and continuously deliver value. Therefore, SAFe places an intense focus on working systems and business outcomes. History shows us that while many enterprises start the transformation with individual Agile teams, they often become frustrated as even those teams struggle to deliver more substantial amounts of solution value, reliably and efficiently.

That is the purpose of the ART, and that is why SAFe focuses implementation initially at the [Program Level](https://www.scaledagileframework.com/program-level/). In turn, the ability of [Value Streams](https://www.scaledagileframework.com/value-streams/) to deliver value depends on the ability of the ARTs and Solution Trains.

But with *alignment*, *transparency*, and *built-in quality* on the team’s side, they have a little ‘wind at their back.’ That enables a focus on *execution*. And if they struggle—and they will, because complex solution development is *hard—*they have the cornerstone of the Inspect and Adapt workshops. In that way, they close the loop and execute better and better during each [Program Increment](https://www.scaledagileframework.com/program-increment/).

Leadership is Required

Successful scaled Lean-Agile development and these four core values require the active support of [Lean-Agile Leadership.](https://www.scaledagileframework.com/lean-agile-leadership/) Leaders couple these core values with SAFe [Lean-Agile Principles](https://www.scaledagileframework.com/safe-lean-agile-principles/" \t "_blank) and practices and an orientation toward [Customer](https://www.scaledagileframework.com/customer/) results. In turn, that creates a persistent and meaningful culture for the teams and their stakeholders.

This is the way successful teams and programs are doing it, and that’s why they are getting the many benefits—employee engagement, productivity, quality, and time to market—that Lean-Agile enterprises so enjoy.

*ScrumXP* is a lightweight process to deliver value for cross-functional, self-organized teams within SAFe. It combines the power of Scrum project management practices with Extreme Programming (XP) practices*.*

ScrumXP details the two essential characteristics of [Team and Technical Agility](https://www.scaledagileframework.com/team-and-technical-agility), with Scrum providing guidance for team agility and XP for technical. Most [Agile Teams](https://www.scaledagileframework.com/agile-teams/) use Scrum as their primary, team-based project management framework. A lightweight yet disciplined and productive process, Scrum allows cross-functional, self-organized teams to operate within the SAFe construct. It prescribes three roles: Scrum Master, Product Owner (PO), and Development Team [2]. The Scrum Master is a servant leader who helps the team adhere to the rules of Scrum and works inside and outside of the team to remove impediments. The Product Owner is responsible for defining what gets built. When extended by Lean quality practices and Extreme Programming (XP) engineering techniques, the ScrumXPteam provides the basic Agile building block for SAFe.

But, of course, ScrumXP teams do not work in isolation. Each is part of the larger [Agile Release Train (ART)](https://www.scaledagileframework.com/agile-release-train/), where they cooperate in building the larger system.

Details

The Agile Team using ScrumXP is a self-organizing, self-managing, cross-functional group of 5 to 11 people, collocated when possible. The size and structure of the team are optimized for communication, interaction, and the ability to deliver value. Self-organization implies that there is no team leader or manager role that oversees the team members, estimates their work, commits them to specific objectives or determines how exactly they will advance the [Solution](https://www.scaledagileframework.com/solution/). The team is presented with the intent of the [Iteration](https://www.scaledagileframework.com/iterations/). Then it is solely responsible for determining how much of that scope they can commit to, and how they are going to build that increment of value. The team is cross-functional, with all the roles and skills needed to deliver a working solution. The self-organization and cross-functional nature of the team—along with constant communication, constructive conflict, and dynamic interaction—can create a productive and more enjoyable work environment for its members.

Scrum defines two specific roles on the Agile team performed by members who have a unique set of specific responsibilities: the [Product Owner (PO)](https://www.scaledagileframework.com/product-owner/) and [Scrum Master](https://www.scaledagileframework.com/scrum-master/). Each of these roles is further described in a SAFe article by that name. A brief summary of their responsibilities is provided below.

Product Owner

Each Agile team has a Product Owner who is responsible for the [Team Backlog](https://www.scaledagileframework.com/team-backlog/). Focusing intensely on the team’s efforts, the PO interacts with them every day. Therefore, the most effective model is to dedicate a PO to each team or to share one across no more than two teams. This allows the PO to support the team effectively during [Iteration Execution](https://www.scaledagileframework.com/iteration-execution/) by answering questions, providing more detail on the functionality under development, and reviewing and accepting the completed [Stories](https://www.scaledagileframework.com/story/) into the baseline.

Scrum Master

The Scrum Master is the facilitator and Agile coach for the team. Primary responsibilities include:

* Ensuring that the ScrumXP process is followed
* Educating the team in Scrum, XP, and SAFe practices
* Providing the environment for continuous improvement

As a full- or part-time role for a team member, the Scrum Master is also typically charged with removing impediments. Alternately, some dedicated Scrum Masters may support two to three Scrum teams.

The Scrum Process

The Scrum process is a lightweight project management framework that fosters quick, iterative advancement of the solution. Facilitating continuous improvement to support higher quality and productivity, and better outcomes, it employs the iteration—a two-week timebox, during which the team defines, builds, tests, and reviews results. The Scrum process is further described in the sections below. (Note: Scrum uses the term ‘sprint.’ SAFe uses the more general term ‘iteration.’)

Planning the Iteration

The iteration starts with [Iteration Planning](https://www.scaledagileframework.com/iteration-planning/), a time-boxed event of four hours or less in which the PO presents the stories for planning. The team then:

* Reviews the stories
* Defines the acceptance criteria
* Splits larger stories into smaller ones where necessary
* Estimates them in story points
* Distills what they can build, based on their known velocity (story points per iteration), into Iteration Goals
* Commits to the iteration goals

Many teams further divide stories into tasks, estimating them in hours to better refine their understanding of the work ahead.

Even before the iteration starts, the Agile team is preparing content by refining the team backlog items. Their objective is to understand the work to be delivered in the upcoming iteration.

Visualizing Work

During execution, the team builds and tests stories with the goal of delivering one or two every few days. This limits Work in Process (WIP) and helps avoid ‘waterfalling’ the iteration. Teams use Big Visual Information Radiators (BVIRs) to understand and track progress during iteration execution. The team’s storyboard visualizes the stories and their progress throughout the iteration. In so doing, they often use development steps as the columns, moving stories from left to right over time, as Figure 1 demonstrates.

Figure 1. An example of a team’s storyboard

Some teams also apply [WIP limits](https://www.scaledagileframework.com/visualize-and-limit-wip-reduce-batch-sizes-and-manage-queue-lengths/) to some steps to create a ‘pull’ process within the iteration and to continuously balance the work to increase throughput. Indeed, many teams integrate the best practices of Scrum and Kanban to facilitate the flow of work through the iterations. In this case, the simple storyboard above evolves into a more structured Kanban board. See the [Team Kanban](https://www.scaledagileframework.com/team-kanban/) article for more on the use of Kanban by Agile teams.

Coordinating with Daily Stand-Up Meetings

Each day, the team has a formal event*—*the Daily Stand-up (DSU) meeting*—*to understand where they are, escalate problems, and get help from other team members. During this meeting, each team member describes what they did yesterday to advance iteration goals, what they are going to work on today to achieve the iteration goals, and any blocks they are encountering in delivering iteration goals. As this is a daily coordination meeting, the Scrum Master has to keep it short and to the point. The DSU should take no more than 15 minutes and is done standing up in front of the storyboard.

But team communication does not end there, as team members interact continuously throughout the iteration. Facilitating such communication is the main reason why ScrumXP prefers that the team be collocated whenever possible.

Demonstrating Value and Improving the Process

At the end of each iteration, the team conducts an [Iteration Review](https://www.scaledagileframework.com/iteration-review/) and an [Iteration Retrospective](https://www.scaledagileframework.com/iteration-retrospective/). During the iteration review, the team demonstrates each story accomplished, culminating with the team’s increment of value for that iteration. This is not a formal status report; rather, it’s a review of the tangible outcomes of the iteration. Thereafter, the team conducts a brief retrospective—a time to reflect on the iteration, the process, things that are working well, and current obstacles. Then the team comes up with improvement stories for the next iteration.

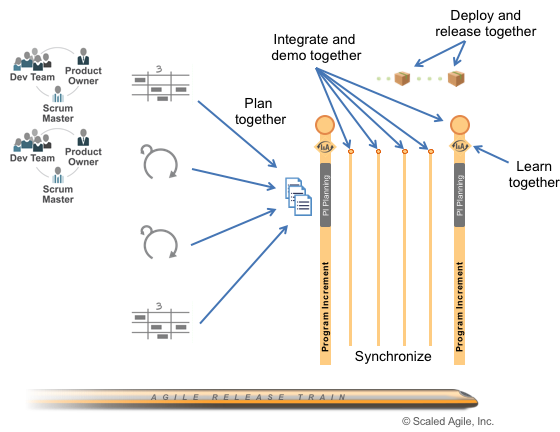
Building Quality In

A tenet of SAFe spells out, “You can’t scale crappy code.”  Therefore, one of the [Core Values](https://www.scaledagileframework.com/safe-core-values/) of SAFe is [Built-In Quality](https://www.scaledagileframework.com/built-in-quality/). Quality begins at the code and component levels with the people creating the solution. Otherwise, it’s difficult (or impossible) to ensure quality later, as the solution integrates and scales from component to system and solution.

To make sure teams build quality-in to code and components, SAFe describes five engineering and quality practices that are inspired by the tenets of XP and that supplement the project management practices of Scrum. They are [Continuous Integration](https://www.scaledagileframework.com/continuous-integration/), [Test-First](https://www.scaledagileframework.com/team-and-technical-agility#testfirst) (including [Test-Driven Development](https://www.scaledagileframework.com/test-driven-development/) and [Behavior-Driven Development](https://www.scaledagileframework.com/behavior-driven-development/)), [Refactoring](https://www.scaledagileframework.com/refactoring/), pair work, and collective ownership. Some teams use other XP practices, such as a pair programming, and metaphor [3].

Agile Teams Using ScrumXP Are on the Train

Although the teams are cross-functional, it isn’t always realistic for a team of seven or eight to deliver end-user value when a large system includes different technology platforms and a spectrum of disciplines. It may include hardware, software, and systems engineering. Typically, many more teams are required. To address this, SAFe Agile teams operate within an ART, which provides mission alignment and a collaborative environment in which teams can cooperate with other teams to build the larger solution capabilities. As part of the ART, all Agile Teams plan together, integrate and demo together, release and deploy together, and learn together, as illustrated in Figure 2.

Figure 2. Agile Teams plan together, integrate and demo together, release and deploy together, and learn together

Each team’s participation in this shared responsibility program is further defined in the Agile team’s article.

Leadership of Agile Teams

Managers are typically not part of the cross-functional team. However, the initial organization of people around features, components, and subsystems—and the design and structure of the ART—is typically a management responsibility based on team input. Thereafter, team management undergoes a shift from ‘manager as an expert,’ directing the team to specific technical achievements, to ‘manager as a developer of people’ and [Lean-Agile Leader.](https://www.scaledagileframework.com/lean-agile-leadership/)

Learn More



It appears that the performance of the task provides its own intrinsic reward…this drive…may be as basic as the others….

—Daniel Pink, *Drive: The Surprising Truth About What Motivates Us*

Principle #8 – Unlock the intrinsic motivation of knowledge workers

Lean-Agile Leaders accept a relatively new, game-changing truth: the ‘management of knowledge workers’ is an oxymoron. As Peter Drucker points out, “knowledge workers are individuals who know more about the work that they perform than their bosses [2].” In that context, how can any manager seriously attempt to oversee or even coordinate the technical activities of people who are infinitely more capable than they are of defining the tasks necessary to accomplish their mission?

Indeed, they cannot. What managers *can* do instead is unlock the intrinsic motivation of knowledge workers. Some guidelines are provided in the following sections.

Leverage the Systems View

Before delving into additional motivational constructs, we note a significant insight: the Lean-Agile principles of SAFe are themselves a system, too. Moreover, the elements of this system collaborate to create a new and empowering management paradigm. With SAFe, knowledge workers are now able to:

* Communicate across functional boundaries
* Make decisions based on an understanding of the economics
* Receive fast feedback about the efficacy of their solution
* Participate in continuous, incremental learning and mastery
* Participate in a more productive and fulfilling solution development process —one of the most powerful motivations of all

Understand the Role of Compensation

*“Many organizations still operate from assumptions about human potential and individual performance that are outdated, rooted more in folklore than in science. They continue to pursue practices such as short-term incentive plans and pay-for-performance schemes in the face of mounting evidence that such measures usually don’t work and often do harm.”*

—Daniel Pink, *Drive: The Surprising Truth About What Motivates Us*

Pink, Drucker, and others have pointed out the fundamental paradox of compensation as a motivational factor for knowledge workers [1, 2]:

* If you don’t pay enough, people won’t be motivated.
* But after a point, money is no longer a motivator. That is the goal of intellectual freedom and self-actualization. When that state is achieved, the knowledge worker’s mind is free to focus on the work, not the money.

After this point, adding incentive compensation elements can shift the focus to the money, rather than the work, resulting in poorer employee performance.

Lean-Agile Leaders understand that ideation, innovation, and deep workplace engagement aren’t motivated by money, or even by the reverse—threats, intimidation, or fear. Such incentive-based compensation, often determined by individual management by objectives (MBO), causes internal competition and the potential destruction of the cooperation necessary to achieve the larger aim. In that competition, the enterprise is the loser.

Provide Autonomy with Purpose, Mission, and Minimum Possible Constraints

Pink asserts that knowledge workers have a need for autonomy—the ability to self-direct and to manage their own lives. Providing autonomy, while harnessing it to the larger aim of the enterprise, is an important leadership responsibility [1].

Managers and workers also know that the motivation of self-direction must occur within the context of the larger business objective. To this end, leaders must provide some larger purpose—some connection between the aim of the enterprise and a worker’s daily work activities.

When building systems, knowledge workers collaborate as a team. Being part of a high-performing group is yet another critical motivation. Leaders can inspire teams to do their best by providing guidance [4]:

* The mission, a general goal and strategic direction, and a strong vision
* Little, minimal, or even no specific work or project plans
* Challenging requirements, along with the minimum possible constraints as to how teams meet these requirements

Create an Environment of Mutual Influence

*“To effectively lead, the workers must be heard and respected”* [2] in the context of an environment of mutual influence [4]. Leaders create this kind of environment by giving tough feedback supportively, by showing a willingness to become more vulnerable, and by encouraging others to engage in positive ways:

* Disagree where appropriate
* Advocate for the positions they believe in
* Make their needs clear and push to achieve them
* Enter into joint problem-solving with management and peers
* Negotiate, compromise, agree, and commit

We live in a new age, where the workers are smarter and have more local context than management can ever have. Unlocking this raw potential can significantly improve the lives of those doing the work, as well as provide better outcomes for customers and the enterprise.

Operating a product development process near full utilization is an economic disaster.

—Donald Reinertsen

Principle #6 – Visualize and limit WIP, reduce batch sizes, and manage queue lengths

To achieve the shortest sustainable lead time, Lean enterprises strive for a state of continuous flow, which allows them to move new system features quickly from ‘concept to cash’. Accomplishing flow requires eliminating the traditional start-stop-start project initiation and development process, along with the incumbent phase gates that hinder flow (see [Principle #5](https://www.scaledagileframework.com/base-milestones-on-objective-evaluation-of-working-systems/) and [Lean Budgets](https://www.scaledagileframework.com/lean-budgets/)).

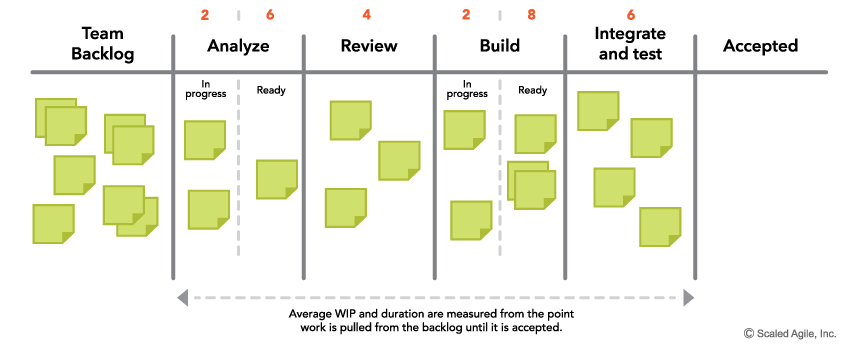
Three primary keys to achieving flow are:

* Visualize and limit work in process (WIP)
* Reduce the batch sizes of work items
* Manage queue lengths

Visualize and Limit WIP

Overloading teams and programs with more work than can be reasonably accomplished is a common and pernicious problem. Having too much WIP confuses priorities, causes frequent context switching, and increases overhead. It overloads people, scatters focus on immediate tasks, reduces productivity and throughput, and increases wait times for new functionality. Like a highway at rush hour, there is simply no upside to having more work in a system than the system can handle.

The first step to correct the problem is to make the current WIP visible to all stakeholders. A Kanban board is a simple way of doing that. Figure 1 illustrates an example.

Figure 1. An example Kanban board showing the steps in a process and WIP limits for each step

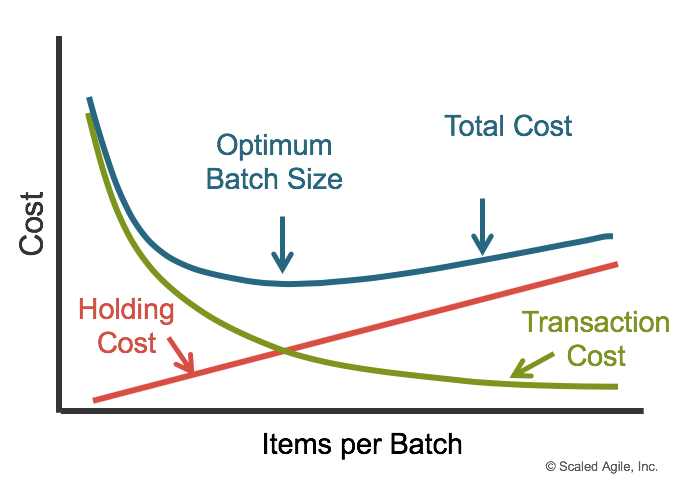
The Kanban board illustrates the total amount of work in process and the state of each work item. It also serves as an initial process diagnostic, showing the current bottlenecks. Often, simply visualizing the current volume of work is the wake-up call that causes practitioners to start addressing the systemic problems of too much work and too little flow.

The next step is to start balancing the amount of WIP against the available development capacity. This is done by establishing—and continually adjusting—WIP limits for the relevant states. Simply, when any workflow state reaches its WIP limit, no new work is taken on. This matches demand to capacity and increases flow through the system.

Limiting WIP, however, requires knowledge, discipline, and commitment. It may even seem counterintuitive to those who believe that the more work you put into the system, the more you get out. That can be true up to the point of nearly full capacity, but thereafter, the system becomes turbulent and throughput decreases. There is no substitute for effectively managing WIP.

Reduce Batch Size

Another way to reduce WIP and improve flow is to decrease the batch sizes of the work—the requirements, designs, code, tests, and other work items that move through the system. Small batches go through the system more quickly and with less variability, which fosters faster learning. The reason for the faster speed is obvious. The reduced variability results from the smaller number of items in the batch. Since each item has some variability, the accumulation of a large number of items has more variability.

Figure 2. U-curve optimization for batch size

The economically optimal batch size depends on both the holding cost (the cost for delayed feedback, inventory decay, and delayed value delivery) and the transaction cost (the cost of preparing and implementing the batch). Figure 2 illustrates the u-curve optimization for batch size [1].

To improve the economics of handling smaller batches—and thus increase throughput—teams must focus on reducing the transaction costs of any batch. This typically involves increasing the attention to and investment in infrastructure and automation, including things such as continuous integration, automating the build environment, automated regression testing and more. (This is a primary purpose of implementing [DevOps](https://www.scaledagileframework.com/devops-and-release-on-demand/).)

Manage Queue Lengths

The third method of achieving flow is to manage queue lengths. Little’s Law—the seminal law of queuing theory—tells us that the average wait time for service from a system equals the ratio of the average queue length divided by the average processing rate. (While this might sound complicated, even the line at Starbucks illustrates that.) Therefore, assuming any average processing rate, the longer the queue, the longer the wait. Simply, reducing queue length decreases delays, reduces waste, increases flow and improves predictability of outcomes.

For solution development, this means that the longer the queue of committed work awaiting implementation by the team, the longer the wait time, no matter how efficient the team. So, for faster service, the length of the queue must be reduced or the processing rate must be increased. While increasing the processing rate (more efficient development and test practices, automation, etc.) is a consistent and common goal, the fastest method to reduce wait time is to reduce the queue length. As illustrated in Figure 1, visualizing the work helps immensely.

Here are some additional tips:

* **Keep team and program backlogs short and largely uncommitted.** This allows new, higher priority work to enter and leave the system with less wait time.
* **Establish WIP limits for each process step.** This means that the length of the queue in front of any one state is limited to the WIP limit.
* **Be especially careful of large, long-term commitments.** It’s always tempting to look into the future and make a long-range commitment which seems to satisfy the need for certainty about the future. But very long-term commitment increases the response time for new opportunities. In other words, every long-term commitment *decreases* the agility of the enterprise.

Summary

These three primary mechanisms for implementing flow—visualizing and limiting WIP, reducing the batch sizes of work, and managing queue lengths—increase throughput and accelerate value delivery*.* Implementing them gives people a sense of control over the process, and triggers fast and measurable improvements in customer satisfaction and employee engagement

Section 2

* *SAFe for Teams Student Workbook* (available only from taking the course): materials and exercises from lesson 2
* • www.scaledagileframework.com/agile-release-train/
* • www.scaledagileframework.com/release-train-engineer-and-value-stream-engineer/

Sections copied above

Section 3

Stories act as a ‘pidgin language,’ where both sides (users and developers) can agree enough to work together effectively.

—Bill Wake, co-inventor of Extreme Programming

Story

Stories are short descriptions of a small piece of desired functionality, written in the user’s language.  [Agile Teams](https://www.scaledagileframework.com/agile-teams/) implement small, vertical slices of system functionality and are sized so they can be completed in a single [Iteration](https://www.scaledagileframework.com/iterations/).

Stories are the primary artifact used to define system behavior in Agile. They’re not requirements. Instead, they’re short, simple descriptions of functionality usually told from the user’s perspective and written in their language. Each one is intended to enable the implementation of a small, vertical slice of system behavior that supports incremental development.

Stories provide just enough information for both business and technical people to understand the intent. Details are deferred until the story is ready to be implemented. Through acceptance criteria and acceptance tests, stories get more specific, helping to ensure system quality.

User stories deliver functionality directly to the end user. [Enabler](https://www.scaledagileframework.com/enablers/) stories bring visibility to the work items needed to support exploration, architecture, infrastructure, and compliance.

Details

SAFe describes a four-tier hierarchy of artifacts that outline functional system behavior: [Epic](https://www.scaledagileframework.com/epic/), [Capability](https://www.scaledagileframework.com/features-and-capabilities/), [Feature](https://www.scaledagileframework.com/features-and-capabilities/), and story. Along with [Nonfunctional Requirements (NFRs)](https://www.scaledagileframework.com/nonfunctional-requirements/), these Agile backlog items define the system and [Solution Intent](https://www.scaledagileframework.com/solution-intent/), model system behavior, and build up the [Architectural Runway](https://www.scaledagileframework.com/architectural-runway/).

Epics, Capabilities, Features, and enablers are used to describe the larger intended behavior. But the detailed implementation work is described through stories, which make up the [Team Backlog](https://www.scaledagileframework.com/team-backlog/). Most stories emerge from business and enabler features in the [Program Backlog](https://www.scaledagileframework.com/program-and-solution-backlogs/), but others come from the team’s local context.

Each story is a small, independent behavior that can be implemented incrementally and provides some value to the user or the [Solution](https://www.scaledagileframework.com/solution/). It’s a vertical slice of functionality to ensure that every iteration delivers new value. Stories are split into smaller ones so they can be completed in a single iteration (see the splitting stories section).

Often, stories are first written on an index card or sticky note. The physical nature of the card creates a tangible relationship between the team, the story, and the user: it helps engage the entire team in story writing. Sticky notes offer other benefits as well: they help visualize work and can be readily placed on a wall or table, rearranged in sequence, and even passed off when necessary. Stories allow improved understanding of the scope and progress:

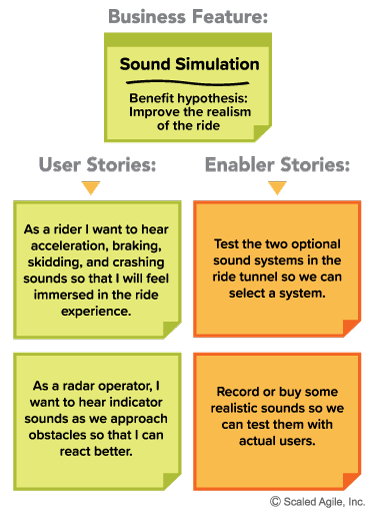
* “Wow, look at all these stories I’m about to sign up for” (scope)
* “Look at all the stories we accomplished in this iteration” (progress)

While anyone can write stories, approving them into the team backlog and accepting them into the system baseline are the responsibility of the [Product Owner](https://www.scaledagileframework.com/product-owner/). Of course, stickies don’t scale well across the [Enterprise](https://www.scaledagileframework.com/enterprise/), so stories often move quickly into Agile lifecycle management (ALM) tooling.

There are two types of stories in SAFe, user stories and enabler stories, as described below.

Sources of Stories

Stories are typically driven by splitting business and enabler features, as Figure 1 illustrates.

Figure 1. Example of a business feature split into stories

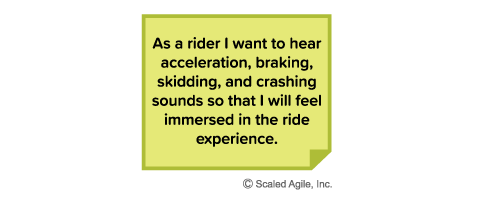
User Stories

User stories are the primary means of expressing needed functionality. They largely replace the traditional requirements specification. (In some cases, however, they serve to explain and develop system behavior that’s later recorded to support compliance, traceability, or other needs.)

Because they focus on the user as the subject of interest, and not the system, user stories are value-centric. To support this, the recommended form of expression is the user-voice form, as follows:

As a (user role), I want (activity) to, so that (business value)

By using this format, the teams are guided to understand who is using the system, what they are doing with it, and why they are doing it. Applying the ‘user voice’ format routinely tends to increase the team’s domain competence; they come to better understand the real business needs of their user. Figure 2 provides an example.

Figure 2. Example user story in user voice form

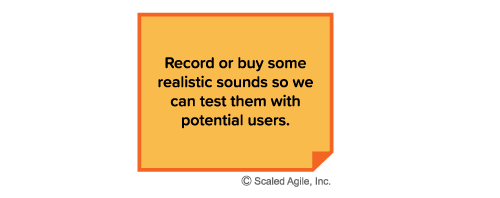
‘Personas’ describe specific characteristics of representative users that help teams better understand their end user.  Example personas for the rider in Figure 2 could be a thrill-seeker ‘Jane’ and a timid rider ‘Bob’. Stories descriptions then reference these personas (As Jane I want…).

While the user story voice is the common case, not every system interacts with an end user. Sometimes the ‘user’ is a device (e.g., printer) or a system (e.g., transaction server). In these cases, the story can take on the form illustrated in Figure 3.

Figure 3. Example of a user story with a ‘system’ as a user

Enabler Stories

Teams may need to develop the architecture or infrastructure to implement some user stories or support components of the system. In this case, the story may not directly touch any end user. These are enabler stories and they support exploration, architecture, or infrastructure. Enabler stories can be expressed in technical rather than user-centric language, as Figure 4 illustrates.

Figure 4. Example enabler story

Enabler stories may include any of the following:

* [Refactoring](https://www.scaledagileframework.com/refactoring/) and [Spikes](https://www.scaledagileframework.com/spikes/) (as traditionally defined in XP)
* Building or improving development/deployment infrastructure
* Running jobs that require human interaction (e.g., index 1 million web pages)
* Creating required product or component configurations for different purposes
* Verification of system qualities (e.g., performance and vulnerability testing)

And, of course, enabler stories are demonstrated just like user stories, typically by showing the artifacts produced or via the user interface, stub, or mock-up.

Writing Good Stories

Good stories require multiple perspectives.  In agile, the entire team – Product Owner, developers, and testers – create a shared understanding of what to build to reduce rework and increase throughput. Teams collaborate using [Behavior-Driven Development](https://www.scaledagileframework.com/behavior-driven-development) (BDD) to define detailed acceptance tests that definitively describe each story.

The 3Cs: Card, Conversation, Confirmation

Ron Jeffries, one of the inventors of XP, is credited with describing the 3Cs of a story:

* **Card** – Represents capturing the statement of intent of the user story on an index card, sticky note, or tool. The use of index cards provides a physical relationship between the team and the story. The card size physically limits story length and premature suggestions for the specificity of system behavior. Cards also help the team ‘feel’ upcoming scope, as there is something materially different about holding ten cards in one’s hand versus looking at ten lines on a spreadsheet.
* **Conversation** – Represents a “promise for a conversation” about the story between the team, [Customer](https://www.scaledagileframework.com/customer/)/user, PO, and other stakeholders. The discussion is necessary to determine more detailed behavior required to implement the intent. The conversation may spawn additional specificity in the form of attachments to the user story (e.g., mock-up, prototype, spreadsheet, algorithm, timing diagram). The conversation spans all steps in the story life cycle:
  + Backlog refinement
  + Planning
  + Implementation
  + Demo

These discussions provide a shared understanding of scope that formal documentation does not provide. Specification by example replaced overly detailed documentation of functionality. Conversations also help uncover gaps in user scenarios and NFRs.

* **Confirmation** – The acceptance criteria provide the information needed to ensure that the story is implemented correctly and covers the relevant functional and NFRs. Figure 5 provides an example. Some teams often use the confirmation section of the story card to write down what they will demo.

Figure 5. Story acceptance criteria

Agile teams automate acceptance tests wherever possible, often in business-readable, domain-specific language. Automation creates an executable specification to validate and verify the solution. Automation also provides the ability to quickly regression-test the system, enhancing [Continuous Integration](https://www.scaledagileframework.com/continuous-integration/), refactoring, and maintenance.

Invest in Good Stories

To remind themselves of the elements of a good story, people often use the [INVEST](http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/) model, developed by [Bill Wake](https://www.industriallogic.com/people/Bill) [1, 2]:

* **I** – Independent (among other stories)
* **N** – Negotiable (a flexible statement of intent, not a contract)
* **V** – Valuable (providing a valuable vertical slice to the customer)
* **E** – Estimable (small and negotiable)
* **S** – Small (fits within an iteration)
* **T** – Testable (understood enough to know how to test it)

Estimating Stories

Agile teams use story points and ‘estimating poker’ to value their work [2, 3]. A story point is a singular number that represents a combination of qualities:

* **Volume** – How much is there?
* **Complexity** – How hard is it?
* **Knowledge** – What’s known?
* **Uncertainty** – What’s unknown?

Story points are relative, without a connection to any specific unit of measure. The size (effort) of each story is estimated relative to the smallest story, which is assigned a size of ‘one.’  A modified Fibonacci sequence (1, 2, 3, 5, 8, 13, 20, 40, 100) is applied that reflects the inherent uncertainty in estimating, especially large numbers (e.g., 20, 40, 100) [2].

Estimating Poker

Agile teams often use ‘*estimating poker*,’ which combines expert opinion, analogy, and disaggregation to create quick but reliable estimates. Disaggregation refers to splitting a story or features into smaller, easier to estimate pieces.

(Note that there are a number of other methods used as well.) The rules of estimating poker are:

* Participants include all team members.
* Each estimator is given a deck of cards with 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞, and,?
* The PO participates but does not estimate.
* The [Scrum Master](https://www.scaledagileframework.com/scrum-master/) participates but does not estimate, unless they are doing actual development work.
* For each backlog item to be estimated, the PO reads the description of the story.
* Questions are asked and answered.
* Each estimator privately selects an estimating card representing his or her estimate.
* All cards are turned over at the same time to avoid bias and to make all estimates visible.
* High and low estimators explain their estimates.
* After a discussion, each estimator re-estimates by selecting a card.
* The estimates will likely converge. If not, the process is repeated.

Some amount of preliminary design discussion is appropriate. However, spending too much time on design discussions is often wasted effort. The real value of estimating poker is to come to an agreement on the scope of a story. It’s also fun!

Velocity

The team’s velocity for an iteration is equal to the sum of the points for all the completed stories that met their Definition of Done (DoD). As the team works together over time, their historical trend of average completed story points per iteration builds a reliable picture of the team’s velocity. Knowing the velocity assists with planning and helps limit Work in Process (WIP), as teams don’t take on more stories than their historical velocity would allow. This measure is also used to estimate how long it takes to deliver epics, features, capabilities, and enablers, which are also forecasted using story points.

Capacity

Capacity is the portion of the team’s velocity that is actually available for any given iteration. Vacations, training, and other events can make team members unavailable to contribute to an iteration’s goals for some portion of the iteration. This decreases the maximum potential velocity for that team for that iteration. For example, a team that averages 40 points delivered per iteration would have to adjust that maximum velocity to some lower number (for example to 36 points – the team’s actual capacity for that iteration) if a team member is going to be on vacation for one week of the two-week iteration. Knowing this in advance, the team only pulls a maximum of 36 points of stories into the iteration during iteration planning. This also helps during PI Planning to forecast the actual available capacity for each iteration in the PI so the team doesn’t over commit in building their PI Objectives.

Starting Baseline for Estimation

In standard Scrum, each team’s story point estimating—and the resulting velocity—is a local and independent concern. In SAFe, however, story points must share the same starting baseline so that estimates for features or epics that require the support of many teams can be understood.

SAFe uses a starting baseline where one story point is defined roughly the same across all teams. This means that work can be prioritized based on converting story points to cost. Of course, adjustments may be needed to account for the different average labor cost across geographies (e.g. United States, China, India, Europe). After all, there’s no way to determine the potential Return on Investment (ROI) if there is no common ‘currency.’  Normalized story points provide a method for getting to an agreed starting baseline for stories and velocity as follows:

* Give every developer-tester on the team eight points (adjust for part-timers).
* Subtract one point for every team member vacation day and holiday.
* Find a small story that would take about a half-day to code and a half-day to test and validate. Call it a ‘one.’
* Estimate every other story relative to that ‘one.’

Example: Assuming a six-person team composed of three developers, two testers, and one PO, with no vacations or holidays, the estimated initial velocity = 5 × 8 points = 40 points/iteration. (Note: Adjusting a bit lower may be necessary if one of the developers and testers is also the Scrum Master.)

In this way, story points are somewhat comparable to an ideal developer day, and all teams estimate using a common method. Management can better understand the cost for a story point and more accurately determine the cost for an upcoming feature or epic.

Note: There is no need to recalibrate team estimation or velocity after that point. It is just a starting baseline.

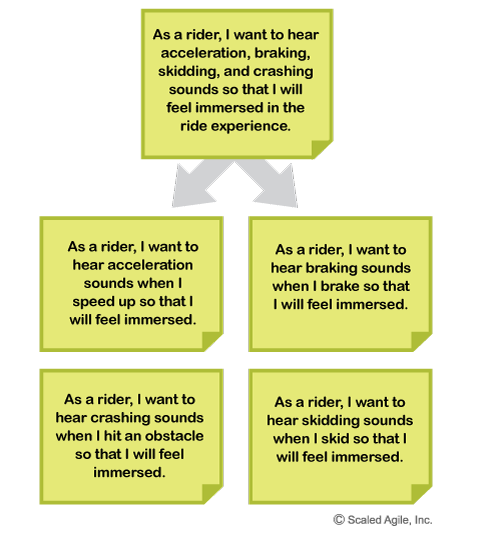
While teams will tend to increase their velocity over time—and that’s a good thing— in reality, the number tends to remain stable. A team’s velocity is far more affected by changing team size and technical context than by productivity variations. If necessary, financial planners can adjust the cost per story point a bit. Experience shows that this is a minor concern, versus the wildly differing velocities that teams of comparable size may have if they don’t set a common starting baseline. That simply doesn’t work at enterprise scale, making it difficult to make economic decisions.

Splitting Stories

Smaller stories allow faster, more reliable implementation, since small things go through a system faster, reducing variability and managing risk. Splitting bigger stories into smaller ones is, thus, a mandatory survival skill for every Agile team. It’s both the art and the science of incremental development. Ten ways to split stories are described in Leffingwell’s Agile Software Requirements [1]. A summary of these techniques follows:

* Workflow steps
* Business rule variations
* Major effort
* Simple/complex
* Variations in data
* Data entry methods
* Deferred system qualities
* Operations (ex., Create, Read, Update, Delete [CRUD])
* Use-case scenarios
* Break-out spike

Figure 6 illustrates an example of splitting by use-case scenarios.

Figure 6. An example of splitting a big Story into smaller stories

Stories in the SAFe Requirements Model

As described in the [SAFe Requirements Model](https://www.scaledagileframework.com/safe-requirements-model/) article, the Framework applies an extensive set of artifacts and relationships to manage the definition and testing of complex systems in a Lean and Agile fashion. Figure 7 illustrates the role of stories in this larger picture.

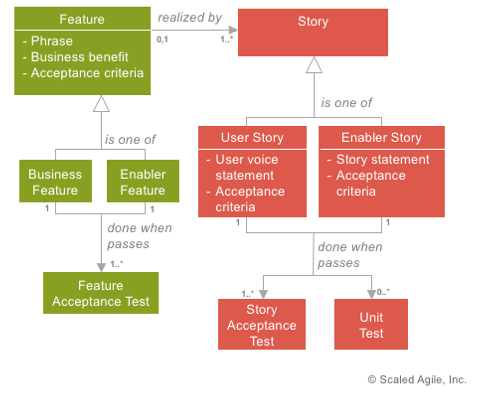
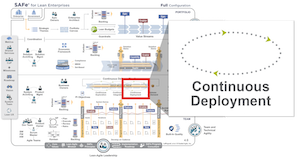
Figure 7. Stories in the SAFe Requirements Model

Figure 7 illustrates how stories are often (but not always) created by new features and how each has a story acceptance test. Further, each story should have a unit test. Unit tests primarily serve to ensure that the technical implementation of the story is correct. Also, this is a critical starting point for test automation, as unit tests are readily automated, as described in the [Test-Driven Development](https://www.scaledagileframework.com/test-driven-development) (TDD) article.

Note:  Figure 7 uses Unified Modeling Language (UML) notation to represent the relationships between the objects: zero to many (0..\*), one to many (1..\*), one to one (1), and so on.

Section 4

[](https://www.scaledagileframework.com/)

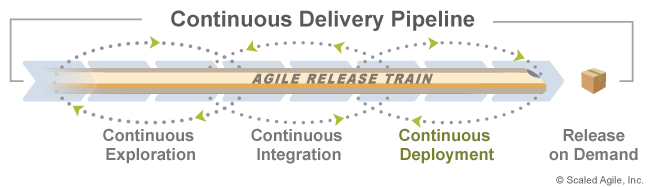
In order for you to keep up with customer demand, you need to create a deployment pipeline. You need to get everything in version control. You need to automate the entire environment creation process. You need a deployment pipeline where you can create test and production environments, and then deploy code into them, entirely on demand.

—Erik to Grasshopper, The Phoenix Project [1]

Continuous Deployment

Continuous Deployment (CD) is the process that takes validated Features from a staging environment and deploys them into the production environment, where they are readied for release.

It is the third element in the four-part [Continuous Delivery Pipeline](https://www.scaledagileframework.com/continuous-delivery-pipeline/) of [Continuous Exploration](https://www.scaledagileframework.com/continuous-exploration/) (CE), [Continuous Integration](https://www.scaledagileframework.com/continuous-integration/) (CI), Continuous Deployment, and [Release on Demand](https://www.scaledagileframework.com/release-on-demand/), as Figure 1 shows.

Figure 1. Continuous Deployment in the context of the Continuous Delivery Pipeline

The ability to Release on Demand is a critical competency for each Agile Release Train (ART) and Solution Train. It allows businesses to respond to market opportunities with the highest value solutions in the shortest sustainable lead times, and to do so at a rate that allows customers to absorb the new functionality.

To enable a business to release on demand, the features must be waiting and verified in production before the business needs them. Therefore, the deployment process is separated from the release process—where deployed changes are moved into the production environment in a manner that does not affect the behavior of the current system. This gives the teams the ability to make smaller, incremental changes which can be deployed to production on a continuous basis but are not released to end users until the time is right.

This article, Continuous Deployment, details the skills a Lean Enterprise needs to continuously deploy potential end-user value to production.

Details

[DevOps and Release on Demand](https://www.scaledagileframework.com/devops-and-release-on-demand/) is one of the five core competencies of the Lean Enterprise. It provides the enterprise with the ability to deliver increasingly valuable solutions to end users with optimal frequency. By systematically reducing time in the development cycle with Built-In Quality approaches, SAFe’s leaner and more Agile approach to the delivery process helps establish faster development flow.

Currently, however, many development teams still deploy solutions to production in large batches. This means the actual deployment and release of the new solution are likely to be manual, error-prone, and unpredictable, adversely affecting go-to-market commitments and solution quality.

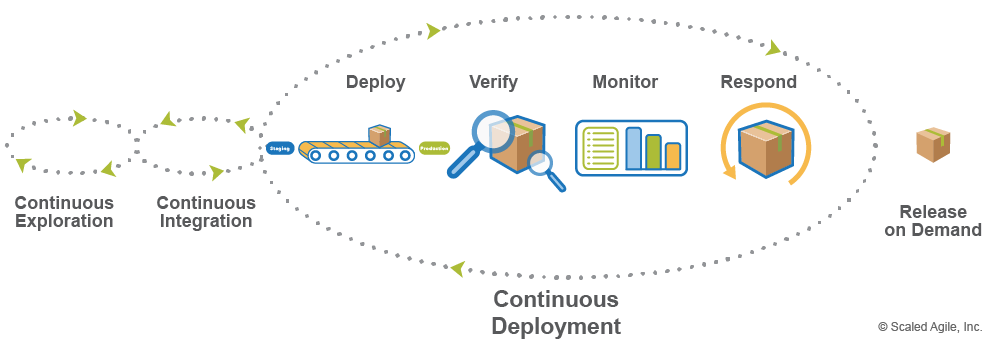
For most organizations, this happens because deployment and release are the same activity. Once value is deployed to production, it is immediately in the hands of the users. This drives organizations to deploy only when the business needs the value, and only when it’s possible to do a full release regression-testing cycle, which leads to large batches.

To address this, ARTs focus their attention collectively on reducing the transaction cost and risk of production deployments by implementing continuous deployment.  By working together to reduce the deployment process to a routine, predictable, “non-event,” teams enable their organizations to achieve a continuous deployment capability that permits value to be released on demand. Facilitating low-cost migration of high-quality changes to production on a regular cadence builds predictability, responsiveness, and competitive advantage.

The Four Sub-dimensions of Continuous Deployment

SAFe describes the four basic sub-dimensions of Continuous Deployment, as Figure 2 shows: deploy to production, verify the solution, monitor for problems, and respond and recover:

* Deploy to production covers the skills necessary to deploy a solution to a production environment
* Verify the solution covers the skills needed to make sure the changes operate in production as intended before they are released to customers
* Monitor for problems covers the skills to monitor and report on solutions
* Respond and recover encompasses the skills to quickly address any problems that happen during deployment

Figure 2. Four sub-dimensions of Continuous Deployment

Deploy to Production

Deployment is the actual migration of changes into a production environment. In the [Continuous Delivery Pipeline](https://www.scaledagileframework.com/continuous-delivery-pipeline/), changes are deployed continuously, regardless of whether the business is ready to release them to end users, and without waiting for complete features (or even stories) to be ready. ARTs need to be able to deploy features in “dark” mode—using feature toggles, for example—so that they can be validated, monitored and queued in a bona fide production setting until customers are ready to receive them.

The actual deployment process needs to be quick, painless, and highly reliable. This is achieved by automating the entire deployment process, from server provisioning and infrastructure configuration to database scripting and code migration. Therefore, it’s imperative to maintain all deployable assets in version control and script all deployment steps in a deployment automation tool.

Ideally, the deployment process is triggered automatically by the deployment pipeline upon successful build, integration, and validation. This makes the entire workflow, from code-commit to production-deploy, a fully-automated “one-click” process. Additionally, organizations should be able to deploy reliably any time of day, any day of the week, and any week of the year—even during peak periods.

The result is a rhythmic, reliable deployment of features to a bona fide production environment, where teams prepare for release to end users.

Seven skills contribute to the ability to deploy:

* Dark launches – the ability to deploy to a production environment without releasing the functionality to end users
* Feature toggles – a technique to facilitate dark launches by implementing toggles in the code, which enables switching between old and new functionality
* Deployment automation – the ability to deploy a tested solution automatically from check-in to production
* Selective deployment – the ability to deploy to specific production environments and not others based on criteria such as geography, user role, etc.
* Self-service deployment – when automation deployment is not fully implemented, self-service deployment allows a single command to take solutions from staging to production
* Version control – maintaining environments under version control enables fast deployment and fast recovery
* Blue/green deployment – a technique that permits automatic switching between two environments, one for deployment and one that is live

Verify the Solution

Before being released to end users, deployments must be verified for functional integrity and robustness. When they’re coupled, deployment and release have to happen almost instantaneously, as decisions must be made immediately about whether to rollback or not. When they’re decoupled, however, there’s room to test new functionality extensively in production before marking it as ready for release. Immediately following the migration to production, solutions undergo a final round of testing, typically in the form of smoke testing and/or light user acceptance testing, but also stress and performance testing that can only be done in production. This provides a critical sanity check that tests the behavior of the solution in an actual production environment.

[Continuous Integration](https://www.scaledagileframework.com/continuous-integration/) will have already provided assurance upstream that the solution will behave as expected in production; however, surprises do occur. When verification reveals critical defects, deployments must either be rolled back or fixed quickly to prevent them from contaminating the production environment or disrupting the flow of business.

Once the deployed changes are verified and operable as intended in the production environment, they are one step closer to being able to release. Four skills help drive verification:

* Production testing – the ability to test solutions in production when they are still ‘dark’
* Test automation – the ability to test repeatedly via automation
* Test data management – managing the test data in version control to create consistency in automatic testing
* Testing nonfunctional requirements (NFRs) – system attributes such as security, reliability, performance, maintainability, scalability, and usability must also be thoroughly tested before release

Monitor for Problems

Verifying that deployed features didn’t break on their way into production is an important pre-release quality check. However, teams also need to ensure they can measure a feature’s performance and value over its entire useful life. Teams cannot stop checking for quality now—before a feature has been used. That would undermine the whole intent of the Continuous Delivery Pipeline, which is to test solution ideas (hypotheses) in a real business setting to make quick and informed judgments about how best to enable strategic business outcomes. The insights that drive this critical feedback loop come primarily from robust monitoring capabilities, which must be in place prior to release.

Effective monitoring requires that full-stack telemetry is active for all features deployed through the Continuous Delivery Pipeline. This ensures that system performance, end-user behavior, incidents and business value can be determined quickly and accurately in production. That information allows tracking and monitoring of each feature, which increases the fidelity of the assertions about business value delivered, as well as increased responsiveness to production issues.

While some business-value metrics cannot be collected until release, teams need to make sure they know how to measure them once the decision to release occurs. Three skills help support this:

* Full-stack telemetry – the ability to monitor for problems across the full stack that a system covers
* Visual displays – tools that display automated measurements
* Federated monitoring – consolidated monitoring across applications in the solution that creates a holistic view of problems and performance

Respond and Recover

The ability to respond and recover from unforeseen production issues is critical to enabling continuous deployment and streamlining the Continuous Delivery Pipeline. The reasons are obvious:

* Production issues impact business customers and end users directly, so the value of deployed solutions can erode quickly when issues occur.
* Production issues spawn rework in the form of fixes, patches, redevelopment, retesting, redeployment, etc. That disrupts the normal flow of value through the pipeline.

Because of the severe impact production issues can have on delivery efficiency and delivered value, teams must ensure that they can proactively detect problems and quickly recover from them at all times. Also, the ability to rapidly resolve issues found when verifying and monitoring increases the ability to continuously deploy with confidence.

In fact, fast recovery is among the most reliable leading indicators of high DevOps maturity, as measured by Mean Time to Restore (MTTR). [5] Recovery also appears in CALMR as one of SAFe’s five core [DevOps](https://www.scaledagileframework.com/devops/) principles.

The goal of respond and recover is to identify potential issues before they turn into incidents and to prevent them from impacting business operations. This requires the ability to detect difficulties before end users do, quickly identify root causes, and restore services with well-rehearsed procedures. In contrast, making hasty, reactive changes directly to production systems—‘just to keep the lights on’—invites configuration drift, unverified changes, and long-term risk.

Six skills support this sub-dimension:

* Proactive detection – a practice for proactively creating faults in the solution to identify potential problems and situations before they occur
* Cross-team collaboration – a mindset of cooperation across the Value Stream to identify and solve problems as they arise
* Session replay – the ability to replay end-user sessions to research incidents and identify problems
* Rollback and fix forward – the ability to both rollback a solution quickly to a previous environment, or to fix a problem quickly through the pipeline without the need to rollback
* Immutable infrastructure – never change the elements of the production environment itself but instead, flow the changes through the Continuous Delivery Pipeline
* Version control – environments should be maintained under version control in order to rollback quickly

When teams have demonstrated that features have been deployed successfully to production and that the necessary monitoring and recovery capabilities are in place to track and manage ongoing value to the business, they have completed the continuous deployment stage of the Continuous Delivery Pipeline. In turn, this gives the enterprise the ability to release whenever warranted.

Section 5

Future product development tasks can’t be predetermined. Distribute planning and control to those who can understand and react to the end results.

—Michael Kennedy, Product Development for the Lean Enterprise

There is no magic in SAFe . . . except maybe for PI Planning.

—Authors

PI Planning

**Find a Course**:

Top of Form

  
Go

Bottom of Form

[](https://www.scaledagile.com/certification/courses/safe-release-train-engineer/)

Program Increment (PI) Planning is a cadence-based, face-to-face event that serves as the heartbeat of the [Agile Release Train (ART)](https://www.scaledagileframework.com/agile-release-train/), aligning all the teams on the ART to a shared mission and [Vision](https://www.scaledagileframework.com/vision/).

For geographically distributed ARTs, the event may occur at multiple locations simultaneously by maintaining constant audio and video communication between the sites.

Details

The Agile Manifesto states, “The most efficient and effective method of conveying information to and within a development team is a face-to-face conversation.”

SAFe takes this to the next level with PI planning, a routine, face-to-face event, with a standard agenda that includes a presentation of business context and vision followed by team planning breakouts—where the teams create their [Iteration](https://www.scaledagileframework.com/iterations/) plans and objectives for the upcoming PI.

Facilitated by the [Release Train Engineer (RTE)](https://www.scaledagileframework.com/release-train-engineer-and-solution-train-engineer/), this event includes all members of the ART, whenever possible. It takes place over two days and occurs within the [Innovation and Planning (IP) Iteration](https://www.scaledagileframework.com/innovation-and-planning-iteration/). Holding the event during the IP iteration avoids affecting the scheduling, or capacity of other iterations in the PI.

PI planning is essential to SAFe: If you are not doing it, you are not doing SAFe. This is quite a significant occasion, as Figure 1 implies.

Figure 1. Face-to-face PI planning. Remote teams are planning at the same time using video conferencing.

Business Benefits of PI Planning

PI planning delivers many business benefits, including:

* Establishing face-to-face communication across all team members and stakeholders
* Building the social network the ART depends upon
* Aligning development to business goals with the business context, [Vision](https://www.scaledagileframework.com/vision/), and [Team and Program PI objectives](https://www.scaledagileframework.com/pi-objectives/)
* Identifying dependencies and fostering cross-team and cross-ART collaboration
* Providing the opportunity for ‘just the right amount’ of architecture and [Lean User Experience (UX)](https://www.scaledagileframework.com/lean-ux/) guidance
* Matching demand to capacity, eliminating excess Work in Process (WIP)
* Fast decision-making

Inputs and Outputs of PI Planning

Inputs to PI planning include:

* Business context (see ‘content readiness’ below)
* Roadmap and vision
* Top 10 [Features](https://www.scaledagileframework.com/features-and-capabilities/) of the Program Backlog

A successful PI planning event delivers two primary outputs:

* **Committed PI objectives** – A set of [SMART](https://www.scaledagileframework.com/pi-objectives/) objectives that are created by each team with the business value assigned by the Business Owners.
* **Program board** – This highlights the new feature delivery dates, feature dependencies among teams and with other ARTs, and relevant [Milestones](https://www.scaledagileframework.com/milestones/).

Preparation

PI planning is a significant event that requires preparation, coordination, and communication. It is facilitated by the [Release Train Engineer](https://www.scaledagileframework.com/release-train-engineer-and-solution-train-engineer/) (RTE). Event attendees include [Business Owners](https://www.scaledagileframework.com/business-owners/), [Product Management](https://www.scaledagileframework.com/product-and-solution-management/), [Agile Teams](https://www.scaledagileframework.com/agile-teams/), [System and Solution Architect/Engineering](https://www.scaledagileframework.com/system-and-solution-architect-engineering/), the [System Team](https://www.scaledagileframework.com/system-team/), and other stakeholders who must be notified in advance to be well prepared. The active participation of Business Owners in this event provides an important [Guardrail](https://www.scaledagileframework.com/guardrails/) on budgetary spend.

For the event to be successful, preparation is required in three major areas:

* **Organizational readiness** – Strategic alignment and teams and trains setup
* **Content readiness** – Management and development preparedness
* **Facility readiness** – The actual space and logistics for the event

Below are highlights of the ART Readiness Checklist. (The full checklist is provided in the [SAFe Program Increment Toolkit](https://www.scaledagile.com/spc-resources/), available to [SPCs](https://www.scaledagileframework.com/safe-program-consultant/)).

Organizational Readiness

Before planning, [Programs](https://www.scaledagileframework.com/program-level/) must have strategy alignment among participants, stakeholders, and Business Owners. Critical roles are assigned. To address this in advance, however, event organizers must consider the following:

* **Planning scope and context** – Is the scope (product, system, technology domain) of the planning process understood? Do we know which teams need to plan together?
* **Business alignment** – Is there reasonable agreement on priorities among the Business Owners?
* **Agile teams** – Do we have Agile teams? Are there dedicated developer and test resources and an identified [Scrum Master](https://www.scaledagileframework.com/scrum-master/) and [Product Owner](https://www.scaledagileframework.com/product-owner/) for the team?

Content Readiness

It’s equally important to ensure that there are a clear vision and context and that the right stakeholders can participate. Therefore, the PI planning must include:

* **Executive briefing** – A briefing that defines the current business context
* **Product vision briefing(s)** – Briefings prepared by [Product Management](https://www.scaledagileframework.com/product-and-solution-management/), including the top 10 [features](https://www.scaledagileframework.com/features-and-capabilities/) in the [Program Backlog](https://www.scaledagileframework.com/program-and-solution-backlogs/)
* **Architecture vision briefing** – A presentation made by the CTO, [Enterprise Architect](https://www.scaledagileframework.com/enterprise-architect/), or System Architect to communicate new [Enablers](https://www.scaledagileframework.com/enablers/), features, and [Nonfunctional Requirements](https://www.scaledagileframework.com/nonfunctional-requirements/) (NFRs)

Facility Readiness

Securing the physical space and technical infrastructure necessary to support a large number of attendees isn’t trivial either—especially if there are remote participants. Considerations include:

* **Facility** – This must be roomy enough for all attendees, with breakout rooms if necessary
* **Facilities/tech support** – These people need to be identified in advance and reachable during setup and testing, and the event itself
* **Communication channels** – For distributed planning meetings, primary and secondary audio, video, and presentation channels must be available

Standard Agenda

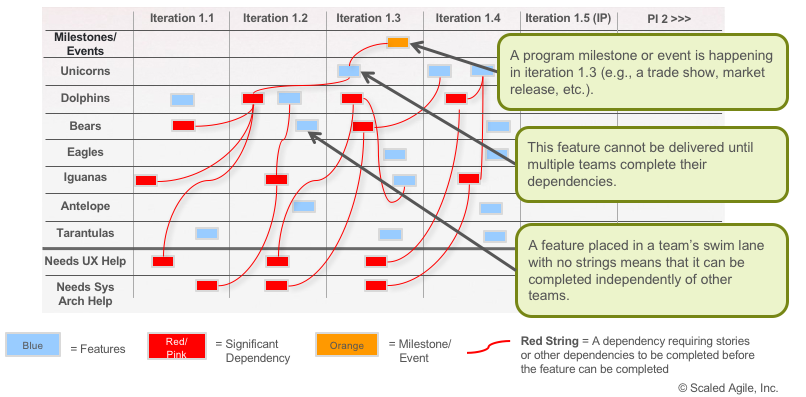
The meeting follows an agenda similar to Figure 2. Descriptions of each item follow.

Figure 2. Standard two-day PI Planning agenda

Day 1 Agenda

* **Business context** – A senior executive/line-of-business owner describes the current state of the business and presents a perspective on how well existing solutions are addressing current [Customer](https://www.scaledagileframework.com/customer/) needs.
* **Product/solution vision** – Product Management presents the current program vision (typically represented by the next top 10 upcoming features) and highlights any changes from the previous PI planning meeting, as well as any forthcoming [Milestones](https://www.scaledagileframework.com/milestones/).
* **Architecture vision and development practices** – System Architect/Engineering presents the architecture vision. Also, a senior development manager may introduce Agile-supportive changes to development practices, such as test automation, [DevOps](https://www.scaledagileframework.com/devops/), [Continuous Integration](https://www.scaledagileframework.com/continuous-integration/), and [Continuous Deployment](https://www.scaledagileframework.com/continuous-deployment), which are being advanced in the upcoming [PI](https://www.scaledagileframework.com/program-increment/).
* **Planning context and lunch** – The Release Train Engineer presents the planning process and expected outcomes of the meeting.
* **Team breakouts #1** – In the breakout, teams estimate their capacity (velocity) for each [Iteration](https://www.scaledagileframework.com/iterations/) and identify the backlog items they will likely need to realize the features. Each team creates their draft plans, visible to all, iteration by iteration.

During this process, teams identify risks and dependencies and draft their initial team PI objectives. The PI objectives typically include ‘stretch objectives,’ which are goals built into the plan (e.g., stories that have been defined and included for these objectives), but are not committed to by the team because of too many unknowns or risks. Stretch objectives are not extra things to do in case there is time. Instead, they increase the reliability of the plan and give management an early warning of goals that the ART may not be able to deliver. The team also adds the features to the program board, as shown in Figure 3.

Figure 3. Program board

* **Draft plan review** – During the tightly timeboxed draft plan review, teams present key planning outputs, including draft objectives, potential risks, and dependencies. Business Owners, Product Management, and other teams and stakeholders review and provide input.
* **Management review and problem-solving** – It’s likely that the draft plans present challenges such as scope, people and resource constraints, and dependencies. During the problem-solving meeting, management may negotiate scope changes and resolve other problems by agreeing to various planning adjustments. The RTE facilitates and keeps the primary stakeholders together for as long as necessary to make the decisions needed to reach achievable objectives.

In multi-ART [Solution Trains](https://www.scaledagileframework.com/solution-train/), a similar meeting may be held after the first day of planning to solve cross-ART issues that have come up. Alternatively, the RTEs of the involved trains may talk with each other to raise issues that are then resolved in the ART’s management problem-solving meetings. The [Solution Train Engineer](https://www.scaledagileframework.com/release-train-engineer-and-solution-train-engineer/) (STE) helps facilitate and resolve issues across the ARTs.

Day 2 Agenda

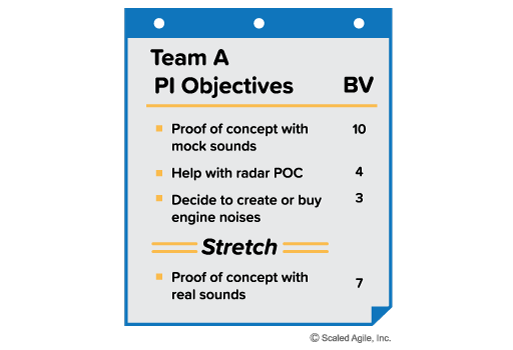
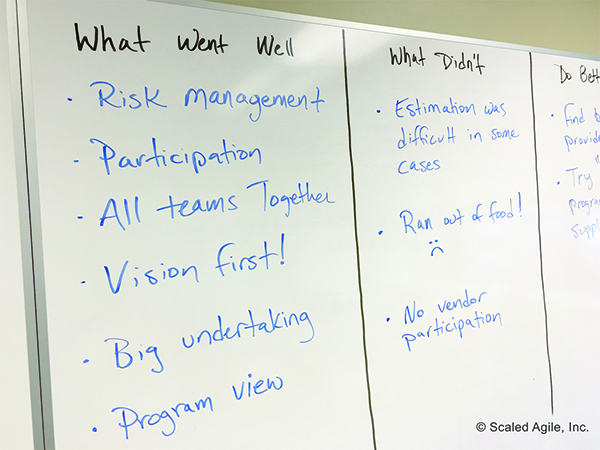
* **Planning adjustments** – The next day, the meeting begins with managers describing any changes to planning scope and resources.
* **Team breakouts #2** – Teams continue planning based on their agenda from the previous day, making the appropriate adjustments. They finalize their objectives for the PI, to which the Business Owners assign business value, as shown in Figure 4. Figure 4. A team’s PI Objective sheet with business value
* **Final plan review and lunch** – During this session, all teams present their plans to the group. At the end of each team’s time slot, the team states their risks and impediments, but there is no attempt to resolve them in this short timebox. If the plan is acceptable to the customers, the team brings their program PI objective sheet and program risk sheet to the front of the room so that all can see the aggregate objectives unfold in real time.
* **Program risks** – During planning, teams have identified program-level risks and impediments that could impact their ability to meet their objectives. These are resolved in a broader management context in front of the whole train. One by one, the risks are addressed with honesty and transparency, and then categorized into one of the following categories:
  + **Resolved** – The teams agree that the issue is no longer a concern.
  + **Owned** – Someone on the train takes ownership of the item since it cannot be resolved at the meeting.
  + **Accepted** – Some risks are just facts or potential problems that must be understood and accepted.
  + **Mitigated** – Teams can identify a plan to reduce the impact of an item.
* **Confidence vote** – Once program risks have been addressed, teams vote on their confidence in meeting their program PI objectives, as illustrated in Figure 5.

Figure 5. Confidence vote for an ART

Each team conducts a ‘fist of five’ vote. If the average is three fingers or above, then management should accept the commitment. If it’s less than three, the team reworks the plan. Any person voting two fingers or fewer should be given an opportunity to voice their concern. This might add to the list of risks, require some replanning, or simply be informative.

* **Plan rework** – If necessary, teams rework their plans until a high confidence level can be reached. This is one occasion where alignment and commitment are valued more highly than adhering to a timebox.
* **Planning retrospective and moving forward** – Finally, the RTE leads a brief retrospective for the PI planning event to capture what went well, what didn’t, and what can be done better next time, as shown in Figure 6.

Figure 6. Planning retrospective

* Typically a discussion about the next steps, along with final instructions to the teams, follows. This might include:
  + Cleaning up the rooms used for planning.
  + Capturing the team PI objectives and user stories in the Agile project management tool.
  + Reviewing team and program calendars.
  + Determining Daily stand-up (DSU) meeting times and locations.
  + Reviewing the locations for the Iteration Planning meetings.

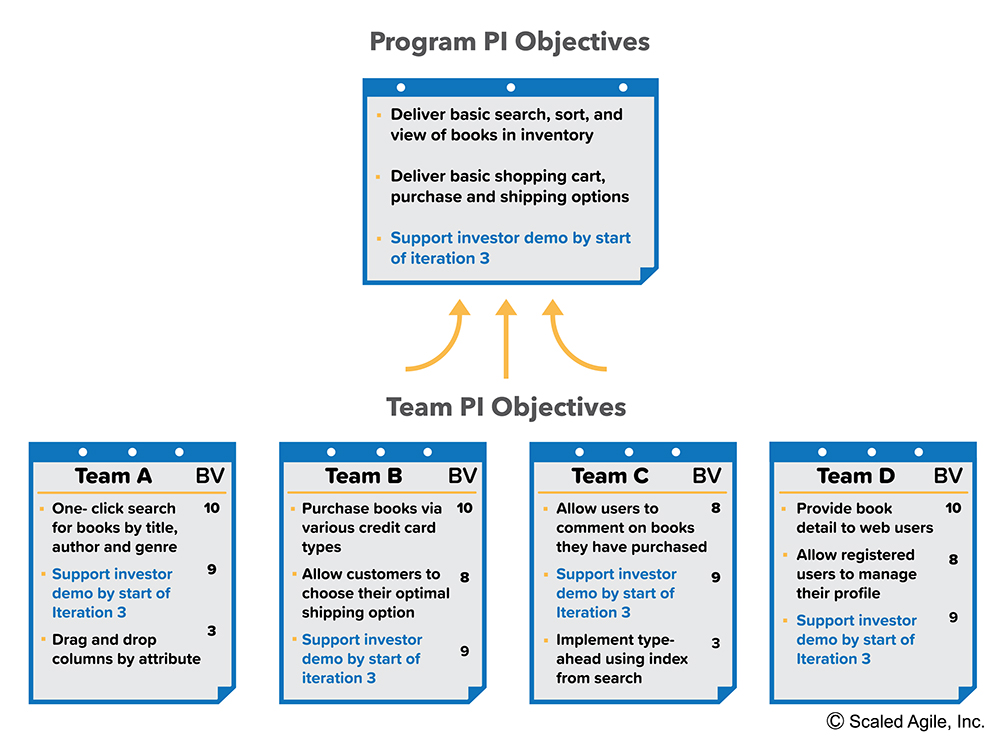
After the planning event is done, the RTE and other ART stakeholders summarize the individual team PI objectives into a set of program [PI objectives](https://www.scaledagileframework.com/pi-objectives/) (Figure 7) and use this to communicate externally and to track progress toward the goals.

Product Management uses the program PI objectives to update the roadmap and will improve the forecast for the next two PIs, based on what was just learned.

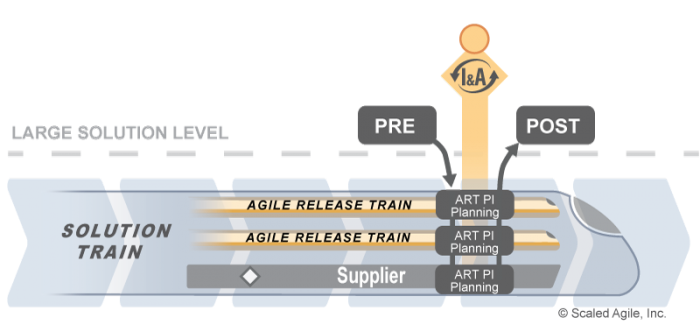
The program board is often used during the Scrum of Scrums meetings to track dependencies, or it may not be maintained (manually) after that time. This depends upon the Agile project management tooling in place and the needs of the ART.

Teams leave the PI planning event with a prepopulated iteration backlog for the upcoming PI. They take their team’s PI objectives, iteration plans, and risks back to their regular work area. Program risks remain with the RTE, who ensures that the people responsible for owning or mitigating a risk have captured the information and are actively managing the risk.

Most important, the program proceeds to execute the PI, tracking progress and adjusting as necessary to the changes that occur as new knowledge emerges. Execution of the PI begins with all the teams conducting planning for the first iteration, using their PI plans as a starting point. This is fresh input for the [Iteration Planning](https://www.scaledagileframework.com/iteration-planning/) processes that follow. Since the iteration plans did not take into account the story acceptance criteria, it’s likely that adjustments will need to be made to the first and subsequent iteration plans.

Solution Train PI Planning

This article focuses on the planning activities of a single ART. However, large [Value Streams](https://www.scaledagileframework.com/value-streams/) may contain multiple ARTs and suppliers. In this case, the Solution Train provides coordination using a [Pre-PI Planning](https://www.scaledagileframework.com/pre-and-post-pi-planning/) meeting, which sets the context and input objectives for the individual ART PI planning sessions. A [Post-PI Planning](https://www.scaledagileframework.com/pre-and-post-pi-planning/) session follows the ART PI planning and is used to integrate the planning results of the ARTs that contribute to the solution.

Figure 8. PI and Post-PI Planning

The [Innovation and Planning Iteration](https://www.scaledagileframework.com/innovation-and-planning-iteration/) article provides an example calendar for the Pre- and Post-PI planning meetings.