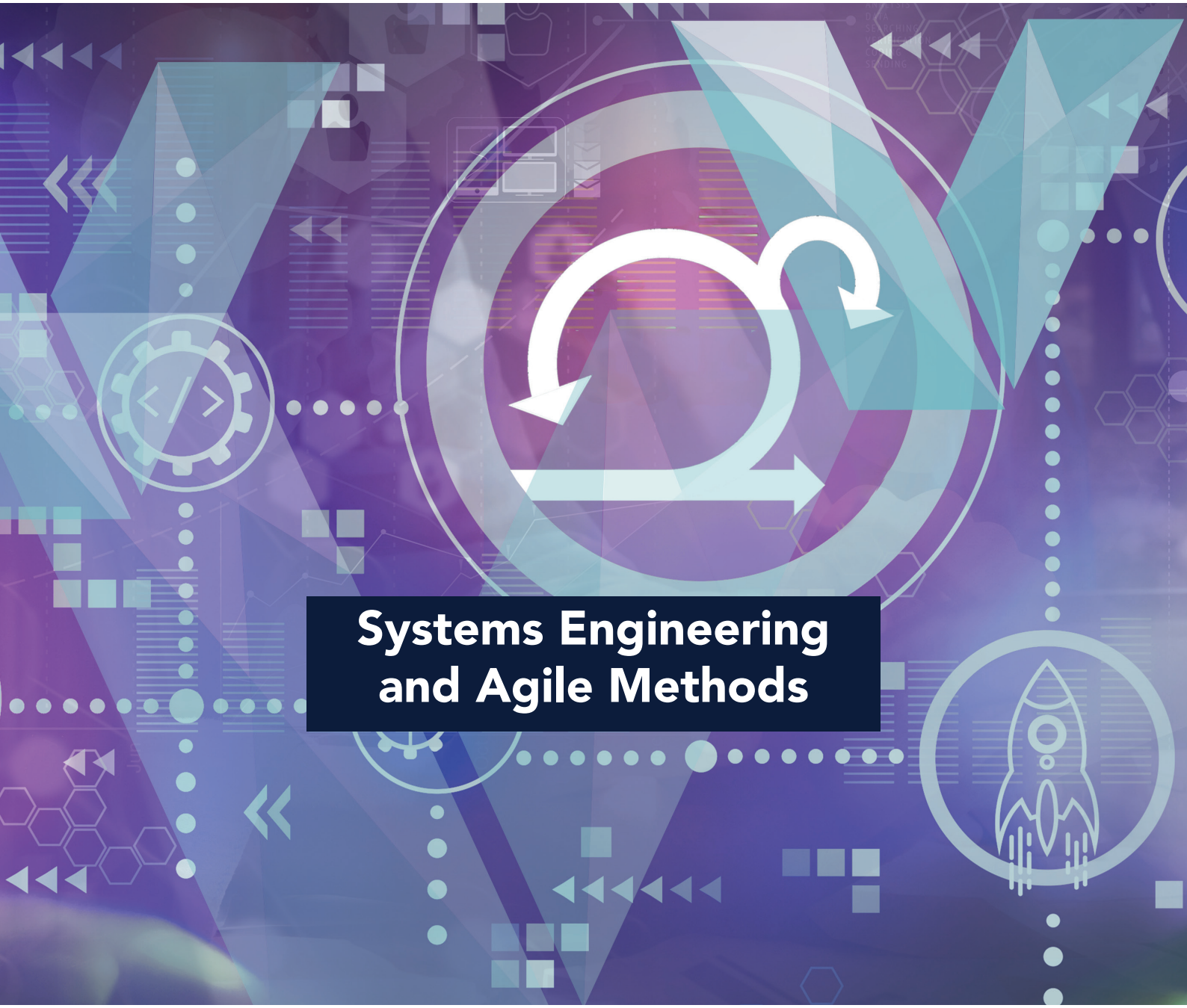
prostep IVIP

White Paper



prostep ivip White Paper

Systems Engineering and Agile Methods

How to Combine Agility with Systems Engineering

Version 1, March 2024

Abstract

This white paper focusses on how systems engineering and agile approaches could complement each other. It describes a framework that can be used to embed agile development into a systems engineering driven product development process. Such a framework must consider constraints given by large projects and large, historically grown organizations. Therefore, this white paper makes suggestions on how to adapt the framework to the different contexts and projects found in real world. It also explains how the framework can be embedded into a classic Product Development Process (PDP).

The “Agile SE” framework described in this whitepaper has many similarities with the Scaled Agile Framework (SAFe). Several ideas from SAFe were adopted in Agile SE. However, Agile SE is not SAFe as it embeds those ideas in a context of systems engineering and a typical Product Development Process (PDP) and describes, how they can coexist.

Content

Table of Content

Figures

Tables

Abbreviations

Abbreviation	Definition	Description
Agile SE	Agile Systems Engineering	A framework for combining agility with systems engineering
MBSE	Model Based Systems Engineering	
PBI	Product Backlog Item	An entry in the product backlog. This can be features but also tasks for providing the architectural foundation or prototypes.
PDP	Product Development Process	The process of developing a product from the first idea until start of production.
PI	Program Increment	The long iteration cycle in SAFe. A typical duration is two or three months.
PIP	Program Increment Planning	An event at the beginning of each PI, to plan the objectives and identify the dependencies for the next PI.
PM	Project Management	
SAFe	Scaled Agile Framework	A framework for a scaled agile approach, provided by Scaled Agile inc. See also: (Scaled Agile, 2023)
SE	Systems Engineering	
Sprint	Sprint	An iteration cycle in Scrum. Also, the short iteration cycle in SAFe. A typical duration is two or three weeks.

1 Intoduction

1.1 Smart Systems Engineering

For more than 12 years the prostep ivip Smart Systems Engineering (Smart SE) project group has focused on collaborative simulation-based engineering. In five different phases the Smart SE group has handled various aspects of collaborative simulation-based engineering. The current project phase focuses on collaborative development:

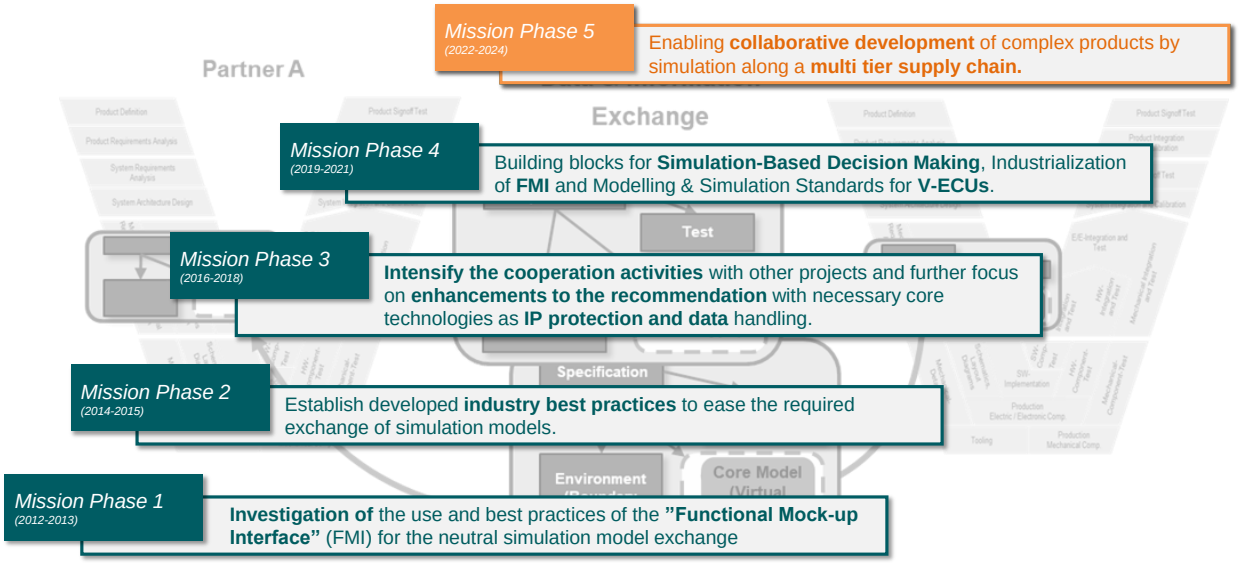


Figure 1: Missions of the Smart Systems Engineering Group

There are five work packages in Phase 5. One of the work packages, the work package “Systems Engineering and Agile Methods”, deals with the combination of systems engineering and agile methods.

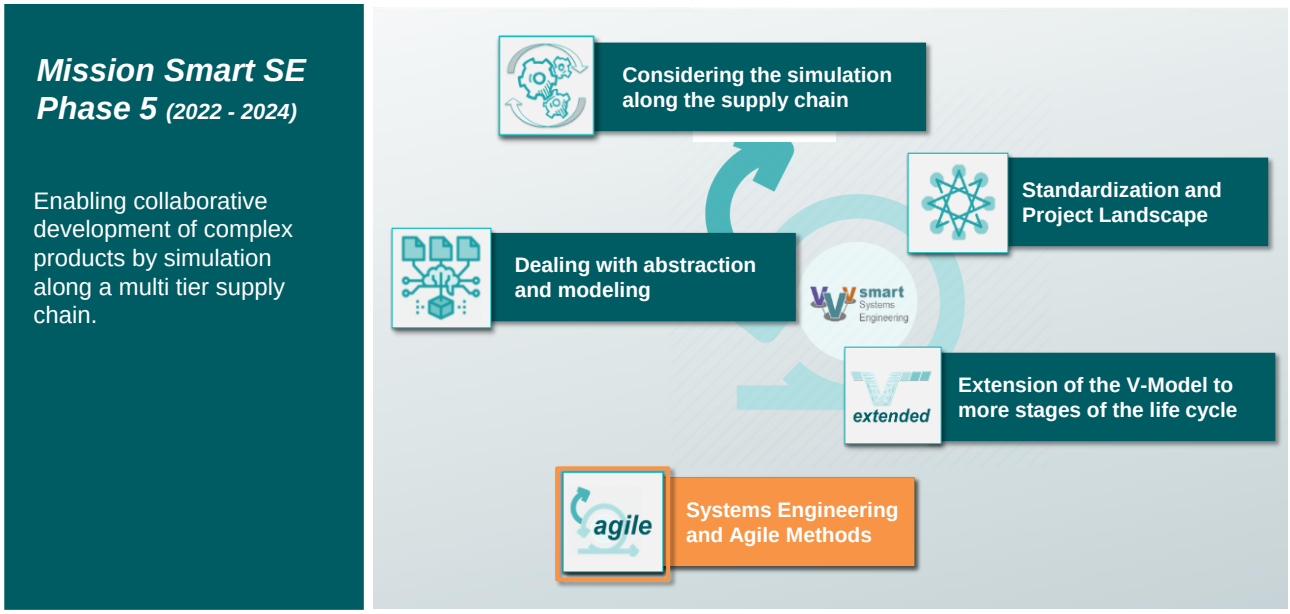


Figure 2: Work Packages in Phase 5

This white paper is the result of this work package. It was created based on expert interviews, expert workshops, and close collaboration within the Smart SE group.

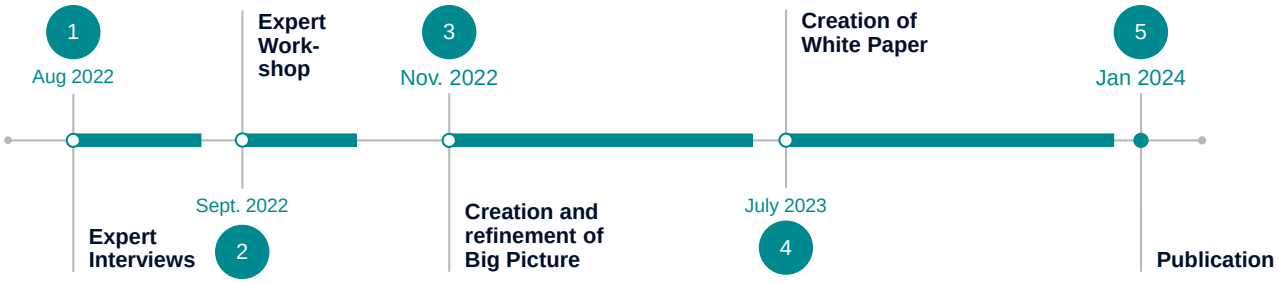


Figure 3: Our approach

1.2 Motivation

Agility becomes more and more important in modern development processes. Time to market and the pressure to innovate require changes until late in the development process. Agility promises to deliver in shorter time frames, less cost and with better business outcome.

On the other hand, many complex systems require complex architectures, approaches like MBSE or a strict traceability from top-level requirements down to implementation. Large, complex and expensive products cannot be easily refactored to implement missing features or correct architectural decisions. A thorough planning in advance is often required to avoid expensive refactoring and loss of time. That's where systems engineering has its advantages.

In real world, both approaches often exist in parallel. You can frequently observe that mechanical and electronic development follow a traditional V-Model approach managed by MBSE and systems engineering, while software is often developed incrementally and agile. Quite often, those two approaches exist without clear interfaces. Agile developers complain about their environment "not providing requirements fast enough". System engineers complain about agile teams "being unreliable" and "not doing what they should". They are "too agile".

This white paper introduces a new Agile Systems Engineering (Agile SE) approach, which aims to bridge between those two worlds. Agile SE is a framework, that can be used to embed agile development into a system engineering driven product development process.

It considers constraints typically observed in large projects or large, historically grown organizations and makes suggestions on how to adapt to the different contexts and projects found in real world. It also explains, how the framework can be aligned with the V-Model and how it can be embedded into a classic Product Development Process (PDP).

2 Typical Situations Observed

Collaboration between agile and systems engineering can be quite complex and is often specific to the individual project or organization. To better understand typical, real-world issues when combining agility within systems engineering, the authors of this white paper conducted interviews with eight experts from agile and systems engineering domains:

- One expert had a mostly stage-gate driven background.
- Two experts had a pure systems engineering background.
- Two experts had a pure agile background.
- Three experts had a combined agile and systems engineering background.

All experts had an industrial background:

- Automotive OEM
- Automotive supplier
- Mobility & Consumer industry

The experts were asked to answer questions based on their personal experience out of the following categories:

- Personal background
- Typical issues in collaboration of software development and systems engineering
- Own experiences with integration of agile methods with system engineering
- Connection to simulation and test
- Own experiences with integration of partners
- Ideas for a green field approach
- Boundaries and success factors

The subsequent chapters elaborate the interview results in more detail.

As only eight experts where interviewed, the interviews are not representative. However, they give ideas about typical issues and typical approaches. Since all experts came from the industrial sector, further investigations should be made which adjustments to the Agile SE approach are needed for other domains like, for example, health care, finance or construction.

2.1 Top-Rated Issues

Based on the expert-interviews the following top-rated issues and challenges were identified:

- Requirements are always too late.
- Different cycle lengths or quality gates. Unaligned sync-points.
- Finding the right degree of architecture.
- Coordination of product-changes.

2.2 Typical Observations

The interview-partners made the following observations:

- **Inappropriate milestones:**

  Milestones are often artefacts to be handed over. But this kind of milestone does not inform about the maturity of the product. It would be better to have milestones describe a certain maturity that must be achieved at a given point of time. Maturity should be measured by availability of important functionality.

- **Unclear responsibilities:**

  Responsibility differs between classical and agile teams: agile teams take full responsibility for their results whereas in a classical line organization responsibility is within the hierarchy.

- **Overlooked synergies:**

  Synergies between systems engineering and software development are not realized: systems engineering and software development are considered as opposite. But systems engineering can be nicely used to fill the product backlog, instead.

- **Misunderstood models:**

  A V-Model describes a logical sequence and not a timeline.

- **Unaligned language:**

  Terms are differently used. Also, the difference between user stories and use cases is often not understood.

- **Transparency:**

  Agility requires transparency - that's not always wanted.

- **Shared understanding:**

  No shared understanding. Does everyone know, what we are doing?

- **Understand limitations of others**

  Make technical limitations of other disciplines visible: what can be achieved within a certain domain and what not. Ideally use models for that.

- **Think in solutions, not in requests:**

  People should talk in options for solutions instead of requests to others. Avoid: „Someone else must solve this".

- **Missing feedback for requirements:**

  Some requirement engineers don't understand a problem and don't accept feedback. Specifications should be created in agile iterations. Customer of a specification is the development team.

- **Acceptance criteria:**

  Missing or badly captured acceptance criteria are a challenge.

- **Missing focus:**

  Developer work on too many functionalities in parallel.

- **Communication:**

  We talk a lot – but that's required!

2.3 Key Findings

Those top-rated issues and typical observations can be boiled down to four key findings:

- Rethink milestones
- Consider SAFe
- Simulations are essential for product maturity.
- Different cultures and culture changes are a major concern.

They are explained in more detail in the next chapters.

2.3.1 Rethink Milestones

Milestones used to be quality gates, that must be fulfilled to 100%. But this requires a detailed planning in advance, which quite often quickly becomes obsolete, due to unforeseen changes or impediments.

Therefore, guardrails, a vision or purpose should be provided instead of fixed quality gates. This leaves space for the development teams to develop the details of the solution themselves and just-in-time.

Milestones should describe a maturity that must be achieved at a given point of time. Maturity should be measured by availability of important functionality.

Work packages should be synchronized frequently. Synchronization only at certain milestones should be avoided.

2.3.2 Consider SAFe

Having a framework for collaboration is essential. SAFe is a good blueprint which can be adjusted to the own needs. It provides good ideas and puts the customer in focus. Three out of eight of the interviewed experts consider SAFe as a suitable approach. (The others didn't mention SAFe.)

SAFe suggests using different cycle length with frequent synchronizations. For Agile SE this could be applied as: agile development in frequent cycles and systems engineering in longer cycles. Synchronization is essential.

Any agile approach should aim for Minimum Viable Products (MVPs) to provide value as early as possible.

2.3.3 Simulations are Essential for Product Maturity.

Most experts consider simulation and tests as essential to foster product maturity.

2.3.4 Different Cultures and Culture Changes are a Major Concern.

Different cultures can significantly impede communication and conflicts can arise from that. Early adopters might quickly adopt new agile approaches whereas others need every single step explained. It is quite common to observe so called "agile teams" that are not willing to think agile. Cultural changes are difficult and require a good leadership.

3 The Agile SE Framework

3.1 Summary

The conducted interviews highlighted significant challenges in the collaboration between agile approaches and systems engineering. Based on these observations, four key findings motivated the development of the Agile SE framework – **(1) Milestones need to be rethought**, **(2) SAFe should be considered as a reference framework**, **(3) Simulations are essential for product maturity**, and **(4) Different cultures and culture changes are a major concern** and need to be addressed.

Incorporating these findings, the Agile SE framework was developed. The overall framework is based on:

- A big picture,
- Fifteen key principles,
- An Agile SE organization
- Team- and program-level events, and
- Enablers

They all will be explained in detail in the following chapters.

Figure 4 shows the **big picture** of Agile SE.

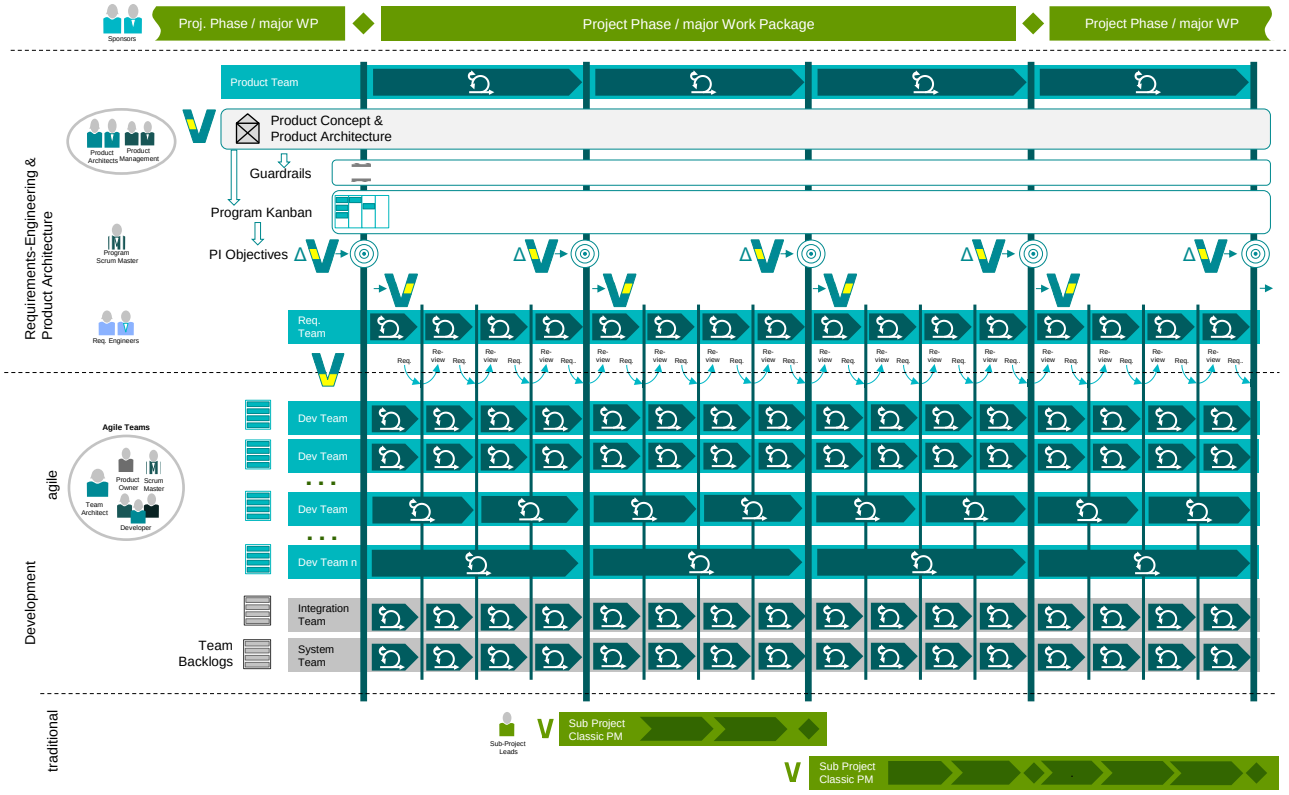


Figure 4. Agile SE - Big Picture

A core element of Agile SE are the **15 key principles**, which are explained in chapter 0. A fundamental principle out of those is the **alignment of Agile SE with the V-Model**. This is explained in detail in chapter 3.2.1.

To ensure the facilitation of the key principles and practices a corresponding **Agile SE organization** needs to be set up. Thus, the framework includes descriptions of the required teams – product-, requirement-, development-, integration-, and system-team. Further, specific roles and stakeholders are identified and described. Read more in chapter 3.3.

The progress of a project within the Agile SE framework is driven by **team and program-level events**, supporting continuous improvement through iterative practices, aligning goals, and encouraging seamless coordination. They are described in more detail in 0.

The framework unfolds its full potential when the five identified **enablers** are taken into account – the presence of an **agile culture**, **use of modelling and simulation** as well as **rapid prototyping**, a **modular and change-friendly product architecture** and **frequent and automated testing**. Read chapter 0 for a detailed description.

Most often, organizations already have **product development processes** in place. Therefore, Agile SE is designed to be integrable into existing processes. To emphasize this aspect, Agile SE provides guidelines on how to set up independent organizations, how these organizations should be handled during ramp-up, full working mode and ramp-down phases. Chapter 4 describes, how Agile SE can be embedded into a product development process and explains how it evolves over the phases of the PDP.

Agile SE is inspired by the **SAFe** framework. Many concepts of the SAFe framework can be found in Agile SE. However, Agile SE is not SAFe, as it introduces additional elements and ignores others. Instead, Agile SE should be considered as a different view on SAFe. In general, Agile SE is compatible with SAFe.

In conclusion, Agile SE provides a novel industry and product agnostic framework combining agility and systems engineering in alignment with the SAFe framework and the Smart SE V-Model.

3.2 Key Principles

The Agile SE big picture incorporates fifteen key principles:

1. Align agile SE layers to V-Model layers appropriately.
2. Use nested iterations.
3. Apply cadence.
4. Choose appropriate sprint length for domains.
5. Provide PI objectives and PI guardrails instead of fixed deliveries at quality gates.
6. Define PI objectives incrementally.
7. Align requirements frequently.
8. Foster communication across the teams.
9. Integrate frequently.
10. Define a high-level roadmap.
11. Align major milestones with program increments.
12. Handle roadmap and concept with appropriate flexibility.
13. Allow coexistence of traditional and agile teams.
14. Synchronize after each program increment.
15. Balance between agile and traditional teams.

They are described in the subsequent chapters.

3.2.1 Align Agile SE Layers to V-Model

One of the fundamental principles of Agile SE is its alignment with the V-Model:

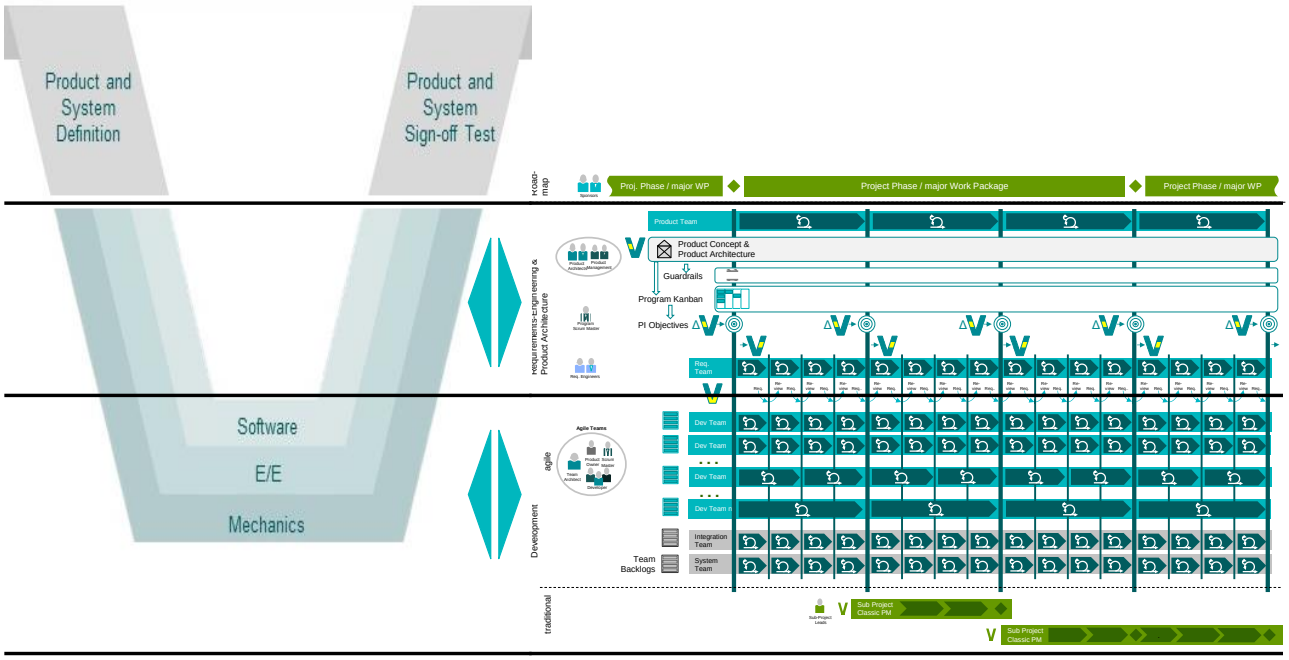


Figure 5: Alignment with the V-Model

Each level of the Agile SE framework level has different roles, tasks and events assigned. Depending on the project, different requirement levels of the V-Model can be matched to the Agile SE levels.

More traditional projects may choose to define requirements with a higher level of detail by traditional systems engineering. In those environments, agile teams will focus mainly on implementation and only few decisions will be made by the agile teams.

More agile organizations will rely on the power and collective intelligence of the agile development teams and provide requirements only to a medium level of detail, allowing the teams to figure out the concrete solution themselves.

Agile organizations with a high level of agility will establish self-organizing teams, that figure out themselves the best way to deliver even complex features. In such environments, the role of systems engineering changes more to coaching and supporting the agile teams in finding the best solution and systems engineering focuses on the high- and medium level-architecture, with tendency to the higher levels.

Figure 6 visualizes how handling of lower and medium level requirements can be flexibly balanced between systems engineering and agile development teams, depending on the context of the organization or project.

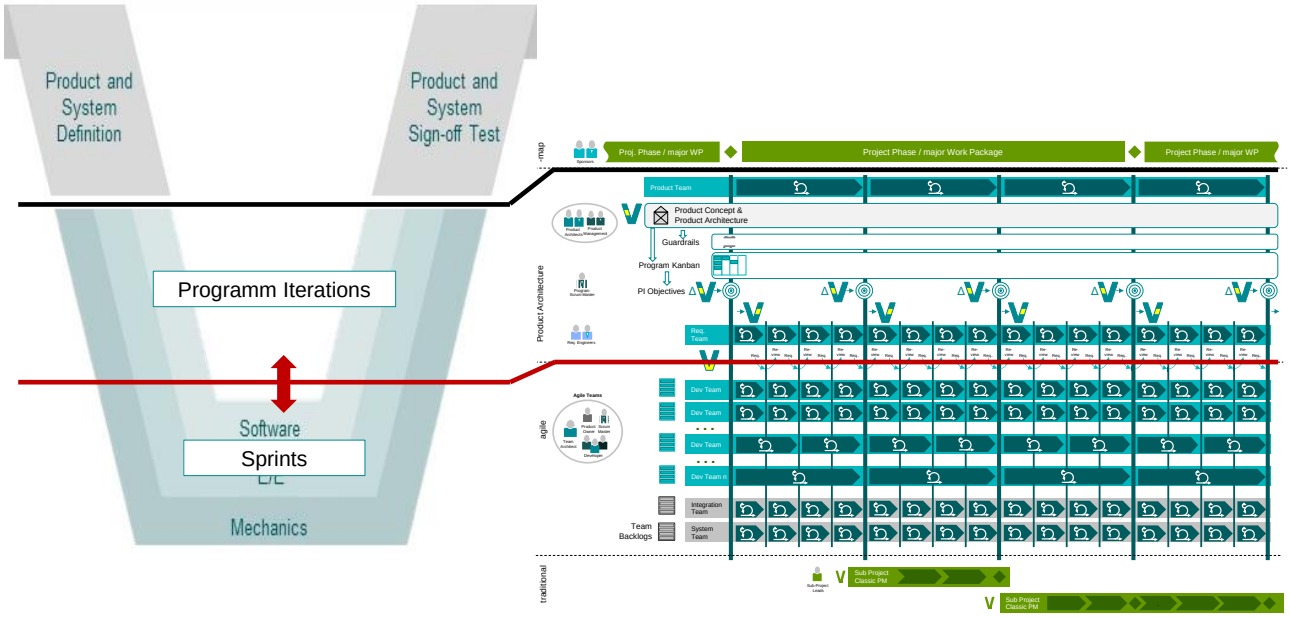


Figure 6: Balancing Requirements between Sprints and PIs

3.2.2 Use Nested Iterations

Another key element of Agile SE are nested iterations: program increments and sprints.

Program Increments (PIs) are long iterations with a typical duration between two and three months. They are used to plan the next features to be implemented based on the outcomes achieved in the previous iterations. At the start of each new program increment the following activities should be performed:

1. **Review** the **outcomes** of the previous program increments and the project context. Did we achieve, what was expected? Are there impediments that require adjustments of the plan? Are there changes in market or environment, that result in new or modified requirements that must be incorporated in the plan?
2. **Review** the overall **roadmap** and adjust it where necessary.
3. Perform a **retrospective** on the Agile SE way of working. Adjust it, where necessary in the sense of continuous improvement.
4. Identify and **plan** the **features**, that should be implemented in the next program increment. (program increment planning event)
5. **Identify** the **teams**, who should implement each feature.
6. **Identify dependencies** between collaborating teams.

Program increments typically iterate with moderate frequency through the medium and lower levels of the V-Model (Figure 7):

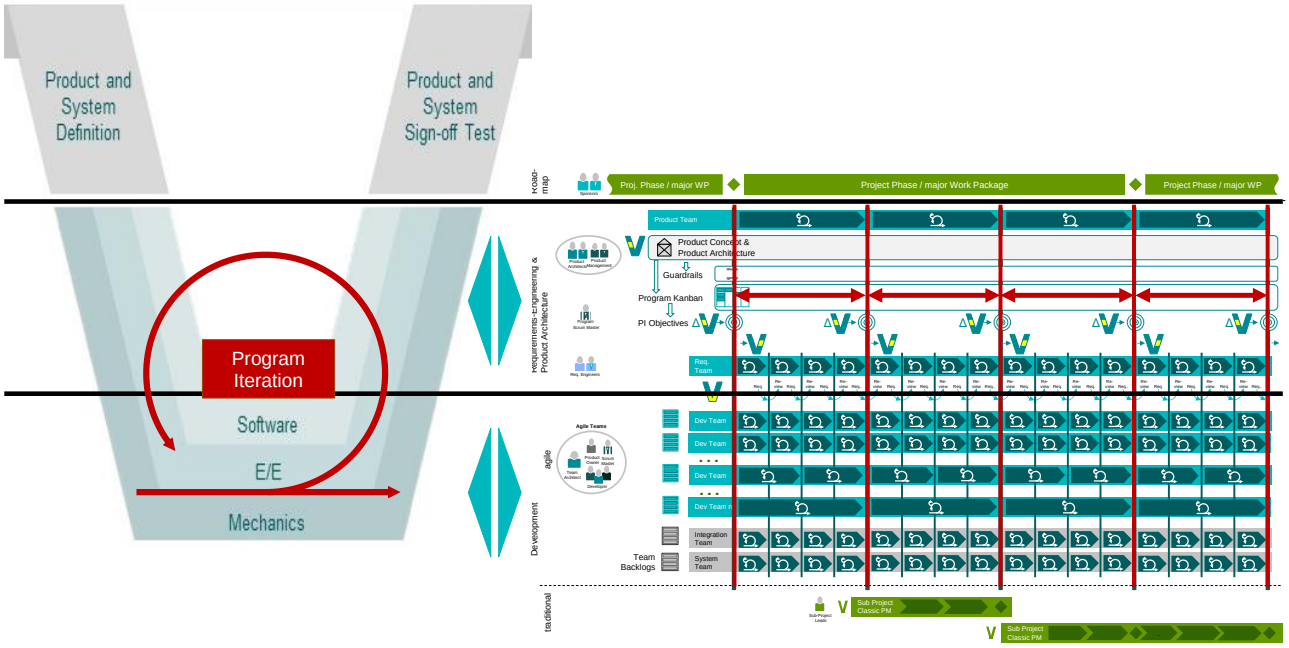


Figure 7: Program Increments on Medium Level of V-Model

The main purpose of program increments is to frequently gather feedback and allow adjustments to the plan and the way of working. Therefore, the duration of program increments should not be too long. On the other hand, the duration should not be too short to avoid significant overhead. It is important to find the right balance. For many projects a duration between two and three month is a good compromise.

Sprints are short iterations, nested into program increments. They are used to implement the features planned for the program increment.

Sprints typically iterate with higher frequency through the lower levels of the V-Models (Figure 8):

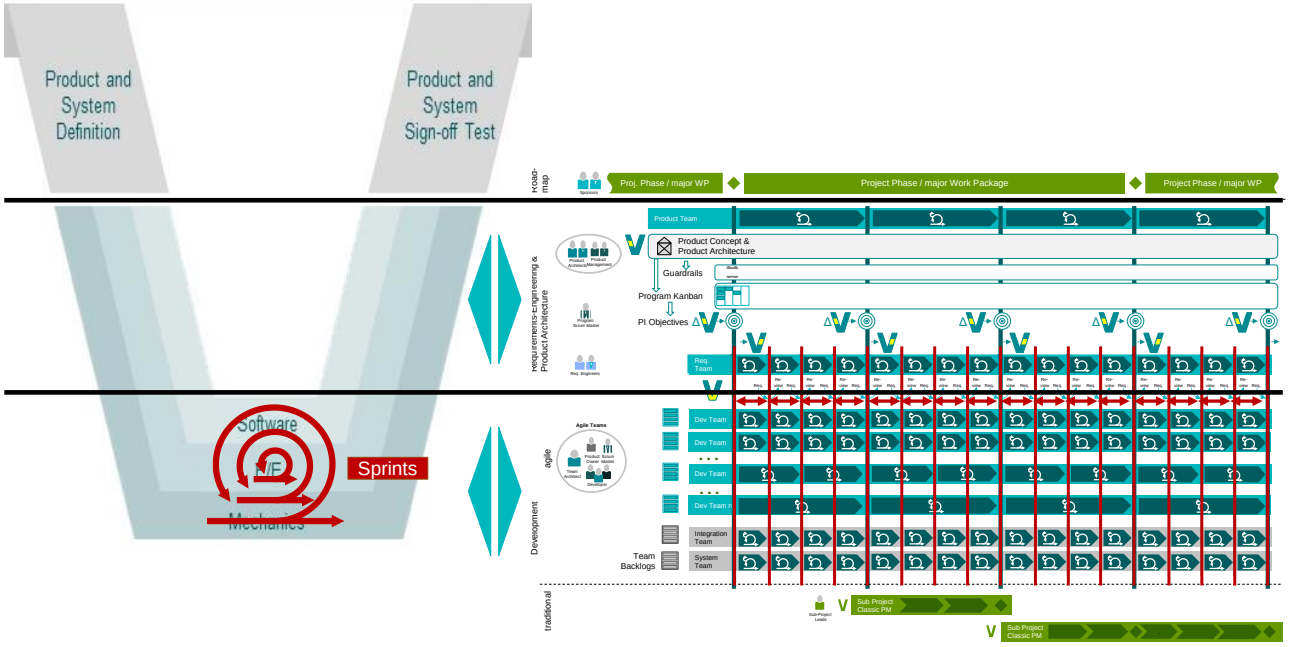


Figure 8: Sprints on Lower Levels of V-Model

At the start of each sprint the following activities should be performed:

1. **Break down** the planned features into **tasks** (or user stories), to implement them.
2. **Review** the **outcomes** of the previous sprint.
3. Maintain the **team backlog**
4. **Plan** the **next sprint.**
5. Perform a **retrospective** on team's way of working. Adjust it, where necessary.

This is standard scrum and will not be elaborated in more detail in this white paper. For more details see (Scrum.org, 2023).

A typical sprint length for software development is between two and four weeks. Chapter 3.2.4 explains, how to handle sprint length for other domains like mechanical engineering.

Figure 9 shows, how sprints are nested into program increments:

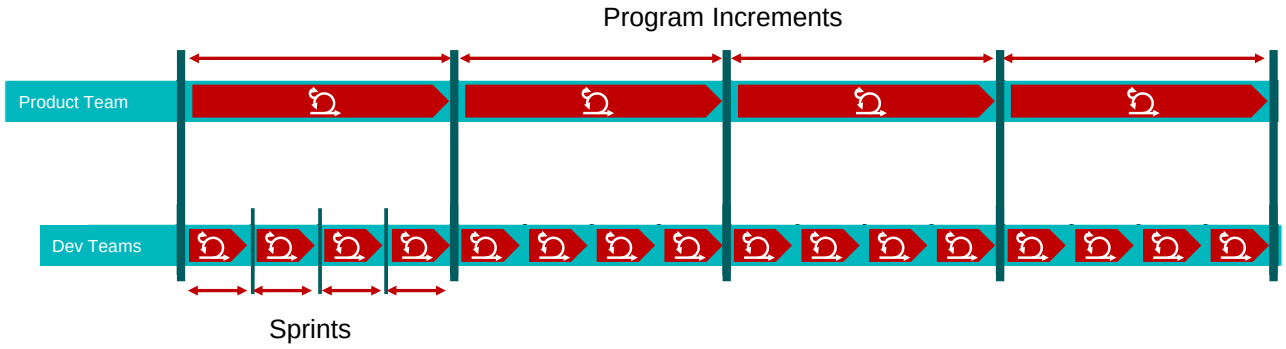


Figure 9: Nested Iterations

3.2.3 Apply Cadence

Sprints and program increments are the mayor types of iterations in the Agile SE framework. Those iterations should be synchronized across all teams. That means: sprints and program increments start and stop for all teams at the same time. This is mainly done to provide a common rhythm for frequent integration. If all teams had different sprint lengths and different starting times, delivering a fully integrated increment would become a challenge. There would be always something "in work", that prevents integration.

It is good practice, not to vary sprint length. Once a certain sprint rhythm was established, the organization should stick to that rhythm, no matter if there are public holidays or vacation times in-between. This significantly improves plannability. If a sprint has fewer working days due to public holidays, teams should consider this by planning less features for this sprint. But the sprint length should not be changed.

Ideally, the length of program increments should not vary either. However, it might be necessary to slightly vary the length of program increments for better synchronization with milestones or release dates (see also chapter 3.2.11). This should be planned carefully, and modified dates should be published well in advance. We suggest that program increment lengths should not vary by more than one sprint.

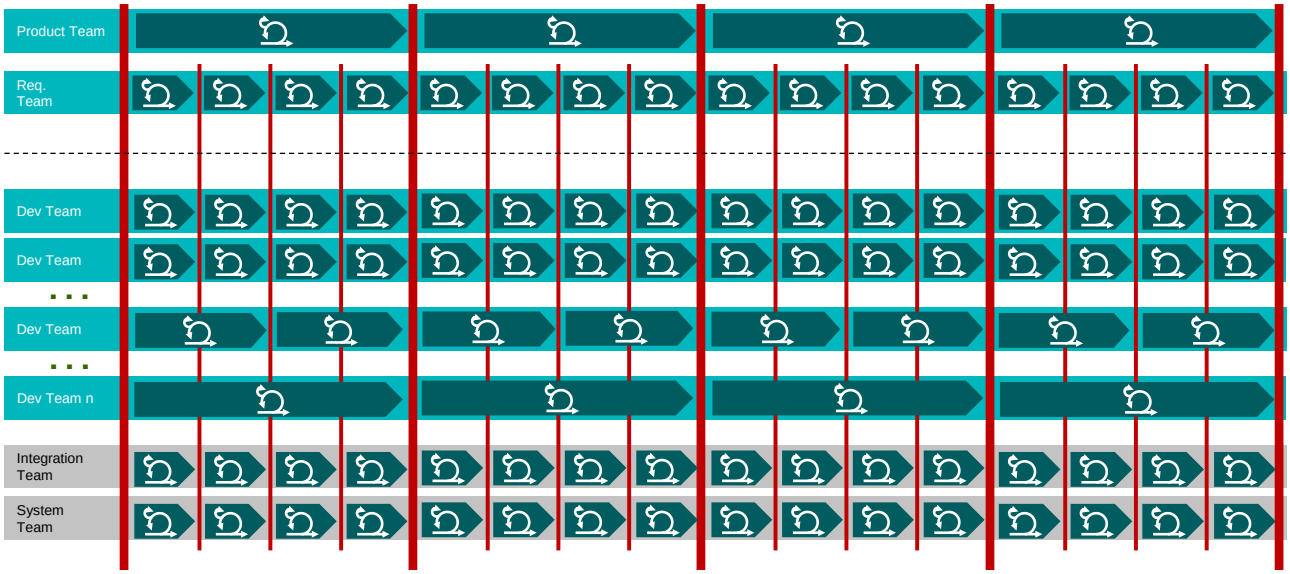


Figure 10: Apply Cadence

3.2.4 Choose Appropriate Sprint Length for Domains

In many situations it is impossible to find one sprint length that suits all technical domains. While a sprint length of two weeks is well suitable for software development, it might be too short for mechanical engineering or domains that involve physical prototying. In this case, teams working in certain domains may decide to work in longer sprints that are a multiple of the underlying sprint length.

The sprint length of the underlying sprint rhythm should be short. A two-weeks sprint cycle for the underlying sprint rhythm it is recommended.

To foster integration, cadence should be maintained, even with teams having different sprint lengths. There should be regular dates, where all sprints end at the same time, allowing to integrate and test as much as possible of the integrated product. A combination of a four-week sprint with a six-week sprint is, for example, not recommended.

Example of possible sprint cycles:

- Software development: two weeks
- Mechanical engineering 1 (mainly using virtual prototypes): two weeks
- Mechanical engineering 2 (mainly using 3D printed prototypes): four weeks
- Mechanical engineering 3 (mainly using tool-based prototypes): eight weeks

In general, sprints should be as short as possible for each domain. And efforts should be taken to shorten sprint length where possible.

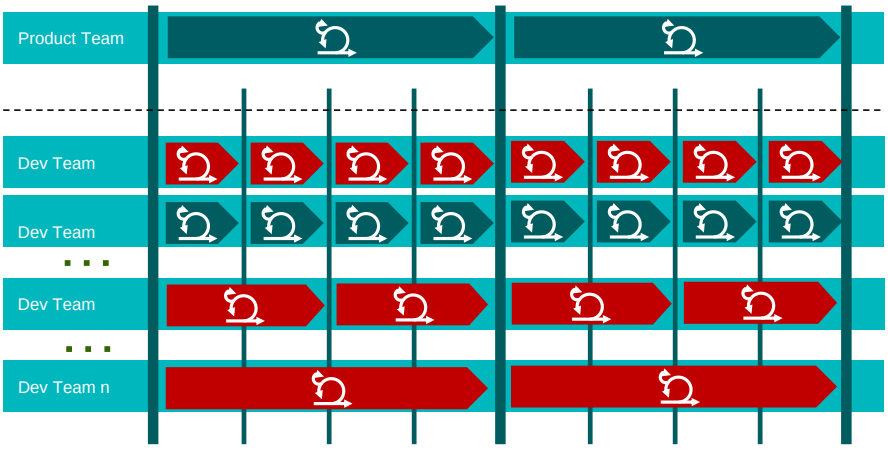


Figure 11: Choose Appropriate Sprint Length for Domains

3.2.5 Provide PI Objectives and PI Guardrails instead of Fixed Deliveries

Traditional quality gates are often defined as a set of fixed deliveries to be delivered at a certain date. A lot of effort is spent in advance, to define all requirements with a high level of detail for those deliveries. However, as practice shows, those detailed plannings frequently become obsolete due to new insights or changed requirements. Therefore, flexibility is needed wherever possible. There is no point in planning the details of a concrete solution in advance, when practice shows, that those plans will frequently change. In consequence, detailed plannings should be replaced by directional objectives. Detailed plannings should be made "just-in-time" and by the teams themselves. They should be made as late as possible to minimize waste due to unforeseen changes. They should be made timely enough to keep the backlog filled without interruptions. Teams must be trusted to find the best solutions.

So, instead of providing a detailed list of deliveries for a specific milestone, it should be dynamically defined, iteration by iteration, what is important in the next iteration to meet the high-level requirements.

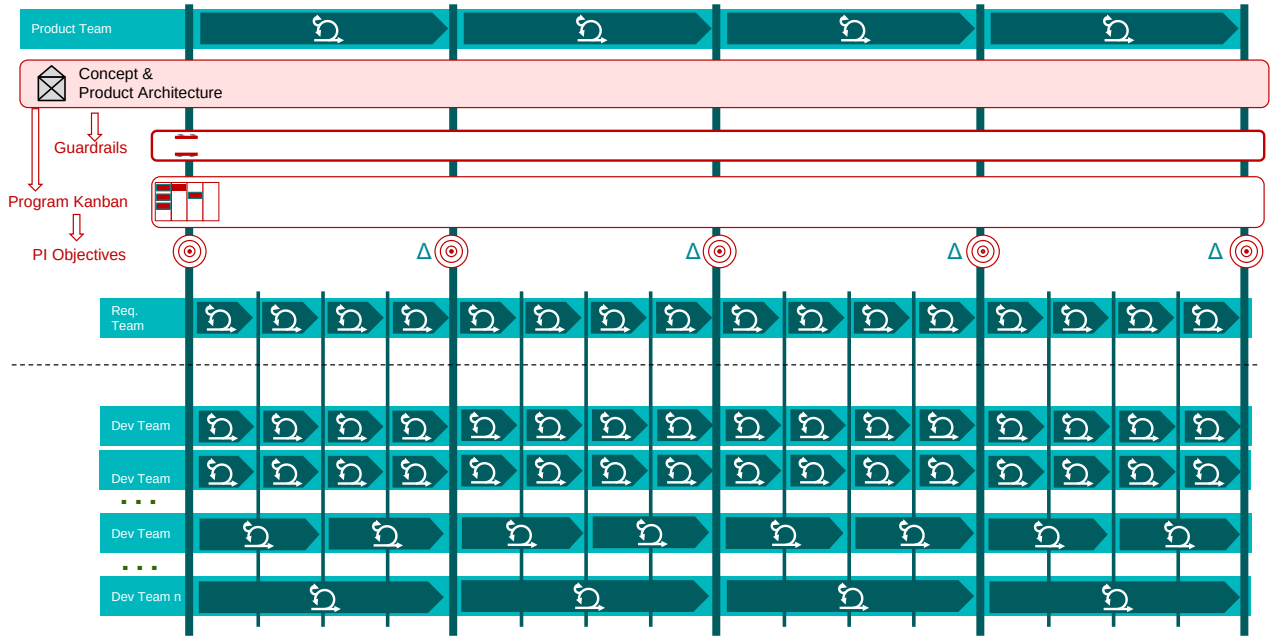


Figure 12: Guardrails and PI Objectives

The Agile SE framework introduces two elements to achieve this: PI objectives and guardrails.

Guardrails are based on the underlying concept and product architecture. They define for the teams, which decisions are fix and where flexibility is available. By defining wiggle room for decisions, they provide flexibility to the teams to find the best solution, based on the latest knowledge, to fulfill the high-level requirements. They avoid having to specify every detail in advance, but foster just-in time decisions, based on the latest, up-to-date information.

PI objectives define, what should be achieved in the next program increment. They are based on the program-backlog holding the features that product management and product architects have identified to be the most important for the next iterations. They do not define in detail how each feature should be implemented. Finding the detailed solution is task of the development teams.

3.2.6 Define PI Objectives Incrementally

Agile SE utilizes a **program backlog**, which holds so-called **program backlog items** (PBIs). Program backlog items are features and architectural enablers required to achieve the high-level requirements for the product. A **program kanban** manages the progress of those PBIs.

PI objectives are based on the PBIs in the program kanban. They define what should be achieved in the next program increment. PI objectives are not static. Instead, they define the delta that should be achieved in the next program increment, based on what has been achieved so far and adding up to the maturity of the product. They are created incrementally by product management and product architects, right before the beginning of

each program increment and considering what has been achieved so far as well as new insights and requirement changes.

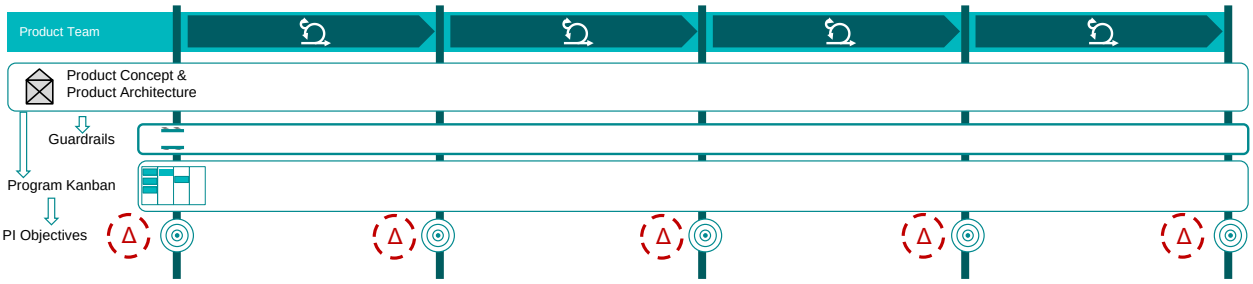


Figure 13: Define PI Objectives Incrementally

3.2.7 Align Requirements Frequently

Based on guardrails, PI objectives and program backlog and as part of the sprint-preparation, the agile development teams break down the program backlog items into low-level tasks, that are maintained in the team's backlog. Often those tasks are documented in the form of "user stories".

Defining user stories typically requires defining the low-level requirements for the features to be implemented. To achieve this, agile developers closely collaborate with the requirements engineers.

Part of this collaboration are reviews of the intermediate sprint results: are there technical issues or impediments? How could they be circumvented? The idea is to identify and discuss issues as soon as possible and react quickly and appropriately.

Requirements engineers must prioritize the tasks in the team backlog. Therefore, they must closely align with the product management. In case of conflicts, the product management has the final right to decide on priority.

The result is a continuously maintained and prioritized team backlog and quickly identified issues and impediments.

This alignment between requirements engineers and agile development teams should be done at least once per sprint, but it is good practice to aim for a higher frequency.

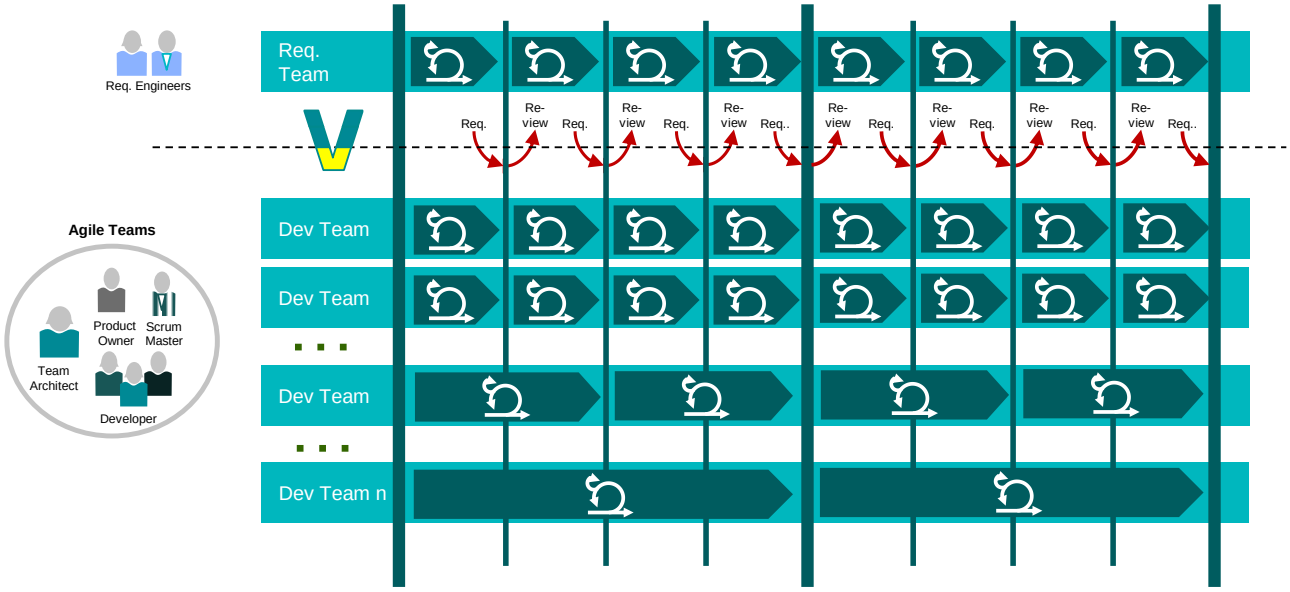


Figure 14: Align Requirements Frequently

3.2.8 Foster Communication across the Teams

Agile approaches aim to frequently deliver integrated and valuable product increments. This requires breaking up silos und collaboration across team boundaries.

At the beginning of each program increment, during the program increment planning, teams identify collaboration needs and dependencies across teams and plan accordingly. However, this is not sufficient. Continuous cross team communication during the program increment is essential to manage dependencies and unforeseen impediments.

A good approach is to establish a so-called "**cross-team alignment meeting**". This meeting provides a platform to quickly arrange cross-team collaboration or remove impediments. It is held once per week. Each team should send at least one representative to this meeting.

Depending on the number of teams the cross-team alignment meeting can be quite a large meeting. Therefore, it must be kept short. A good approach is to conduct it like a Daily Stand-Up in Scrum: each team representative shortly tells what the team is working on, what impediments the team is fighting with and where the team needs help of others. He also listens to the reports of the others and shares the insights later with his own team. If an issue cannot be resolved within two minutes, the moderator stops the discussion and arranges a follow-up meeting with only the required participants. If a team has no impediments or dependencies, it is fine if the representative just says: "nothing to report".

Practice shows that such a meeting, even with more than 15 participants, can be finished within less than 45 minutes.

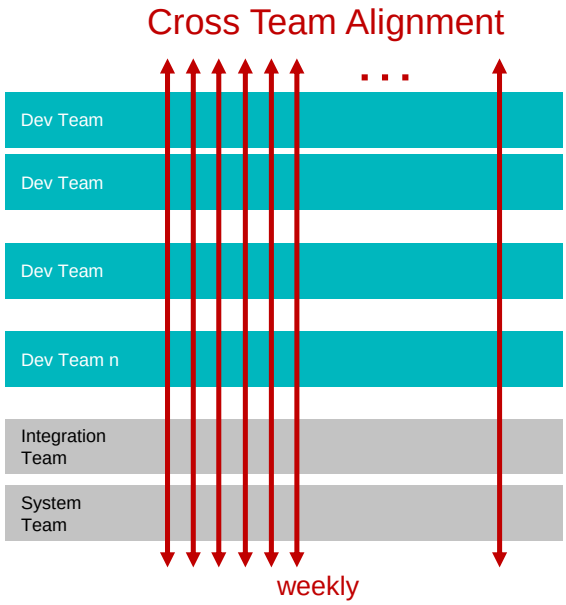


Figure 15: Cross Team Alignment

3.2.9 Integrate Frequently

One of the main purposes of agile approaches is to shorten feedback loops. This requires frequent integration of the deliveries of the different development teams. The goal is, to integrate at least once at the end of each program increment. Where possible, integration should be done even more frequently, for example after each sprint or in case of different sprint length (see chapter 3.2.3) at each synchronization point.

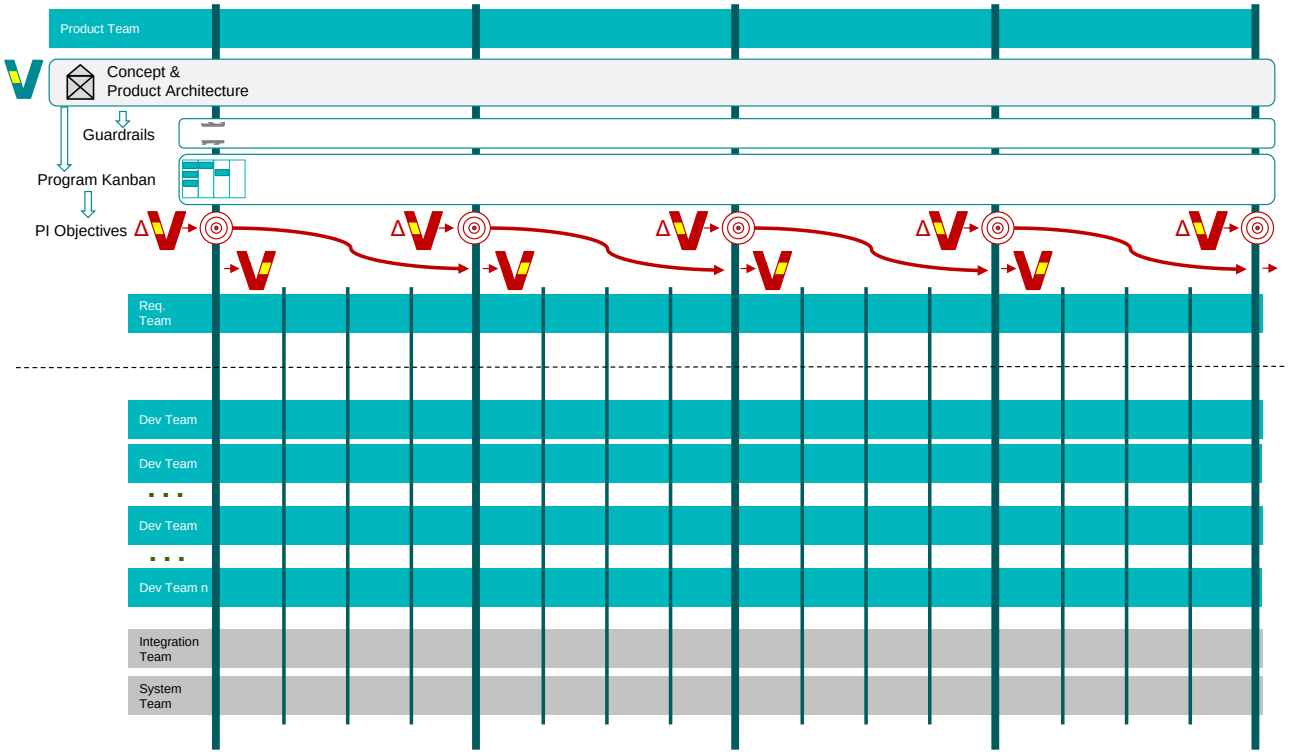


Figure 16: Frequent Integration - At least once per PI

Frequent integration can be a challenge – specifically when physical components are involved. There is no one-size-fits-all approach to this. Projects must find their own approach how frequent integration can be achieved. Potential approaches are:

- **Modeling and simulation**: simulating the interaction of mechanical, electronical and software components based on models. (Chapter 3.5.2)
- **3D printed prototypes**: quick creation of physical prototypes using 3D print. (Chapter)
- **Modular, change friendly product architecture:** designing the product in a modular way, so that interfaces between components are minimized. E.g., an ECU capable to run multiple types of software. (Chapter 3.5.4)

3.2.10 Define a High-Level Roadmap

Only few large real-world projects can be developed fully agile. Most large projects require a high-level roadmap as guidance and milestones for coordination. In Agile SE a high-level roadmap is used to coordinate the major deliveries. It is aligned to the top levels of the V-Model (Figure 17):

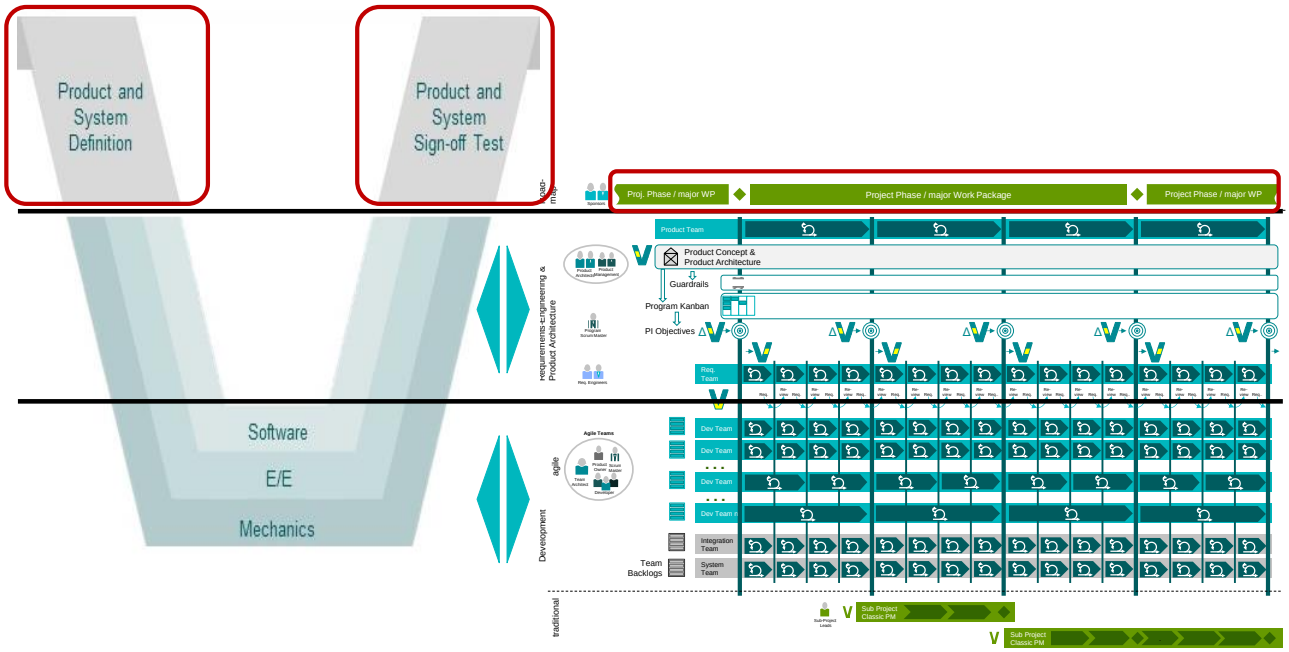


Figure 17: Top Level of V-Model Handled by a Roadmap

Typically, the phases of a roadmap span across multiple program increments:

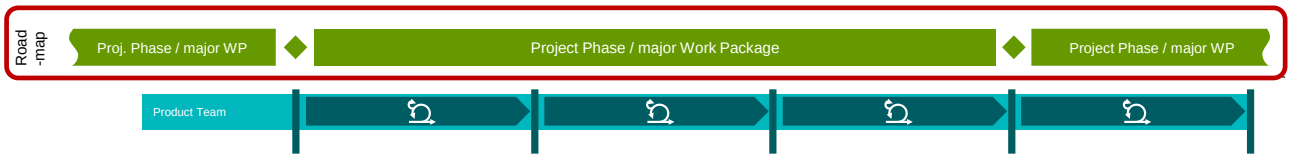


Figure 18: Phases Span across Multiple Program Increments

3.2.11 Align Major Milestones with Program Increments

Ideally, major milestones on the roadmap are aligned with program increments. When setting up a project according to the Agile SE framework and defining the overall roadmap, major milestones on the roadmap can be often aligned with program increment boundaries. Typically, this means, that a milestone is shifted a couple of weeks back or forth. A small adjustment of a PI length is also possible (see chapter 3.2.3).

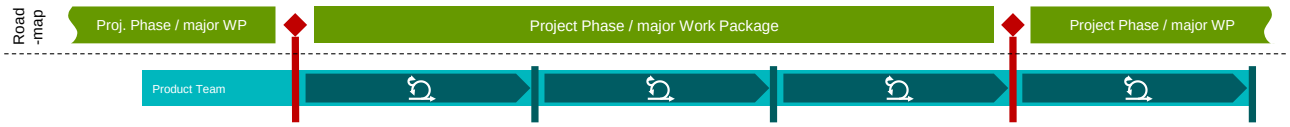


Figure 19: Alignment of Major Milestones with Program Increments

There will be cases where a synchronization between program increments and major milestones is not possible. In this case, non-aligned major milestones must be considered during PI planning. However, this can seriously impact the benefits of an agile approach. In general, it is better to have program increments with unequal length than having non-aligned major milestones.

3.2.12 Handle Roadmap and Concept with Appropriate Flexibility

Handling the roadmap with an appropriate level of flexibility is a way to balance between traditional project management and agility.

Traditional organizations often define a roadmap and a concept and stick to it, only adjusting it, if it cannot be avoided. They consider the roadmap a high-level project plan, that must be followed.

More agile organizations will review the roadmap and the concept after each program increment, evaluate achievements and changes to the project context and adjust the roadmap accordingly. The roadmap provides guidance and vision but stays dynamic. In this case the roadmap becomes a result of the sequence of PI plannings.

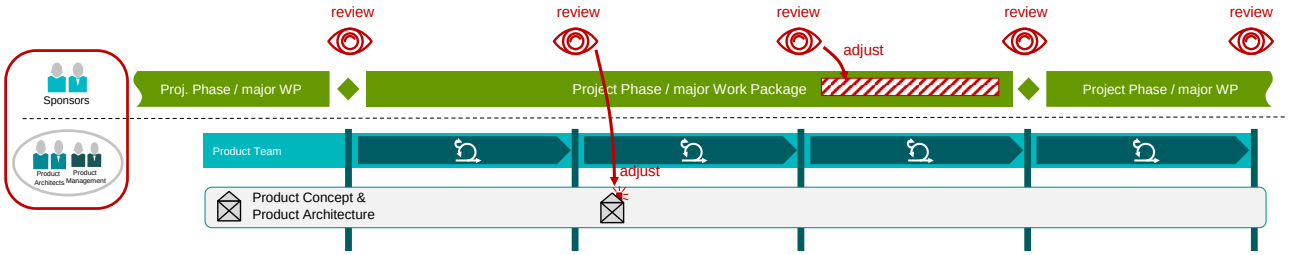


Figure 20: Flexibility in Roadmap and Concept

3.2.13 Allow Coexistence of Traditional and Agile Teams

In some cases, certain deliveries must be managed by traditional project management. For example, due to contract requirements with external partners or constraints in large organizations. While Agile SE does not recommend this, Agile SE acknowledges that there are circumstances that cannot be avoided. Therefore, Agile SE supports the coexistence of agile teams and teams, managed by traditional project management.

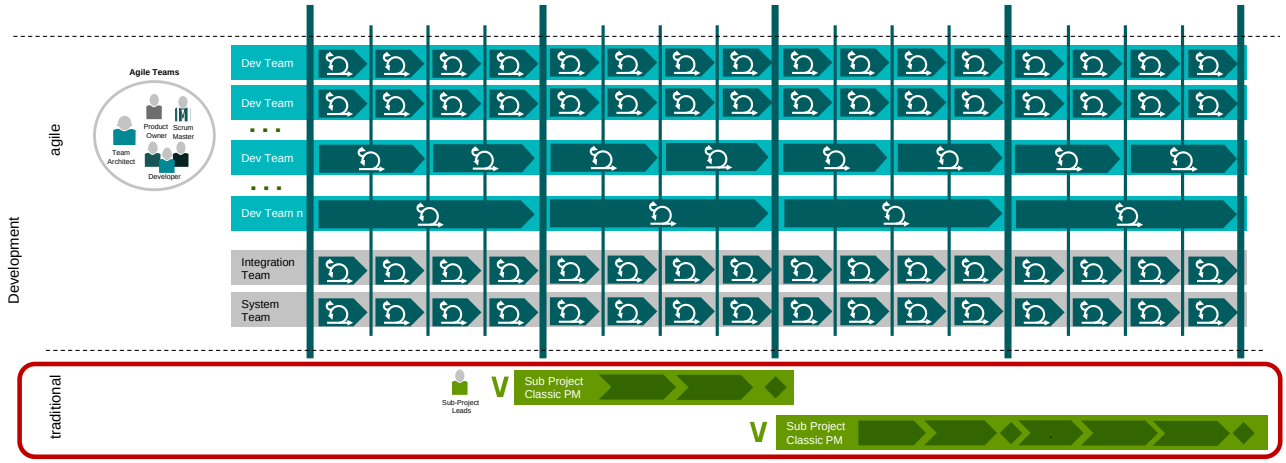


Figure 21: Traditional and Agile Teams

As far as possible, traditional teams should be involved in agile program level events like PI planning, PI review or program-level retrospective. Specifically, the cross-team alignment event (see chapter 3.2.8) is very valuable to keep the agile and traditional teams connected.

3.2.14 Synchronize after each Program Increment

When traditional and agile teams coexist, milestones of traditional teams should be aligned with program increments of agile teams, to simplify integration of agile and traditional deliveries.

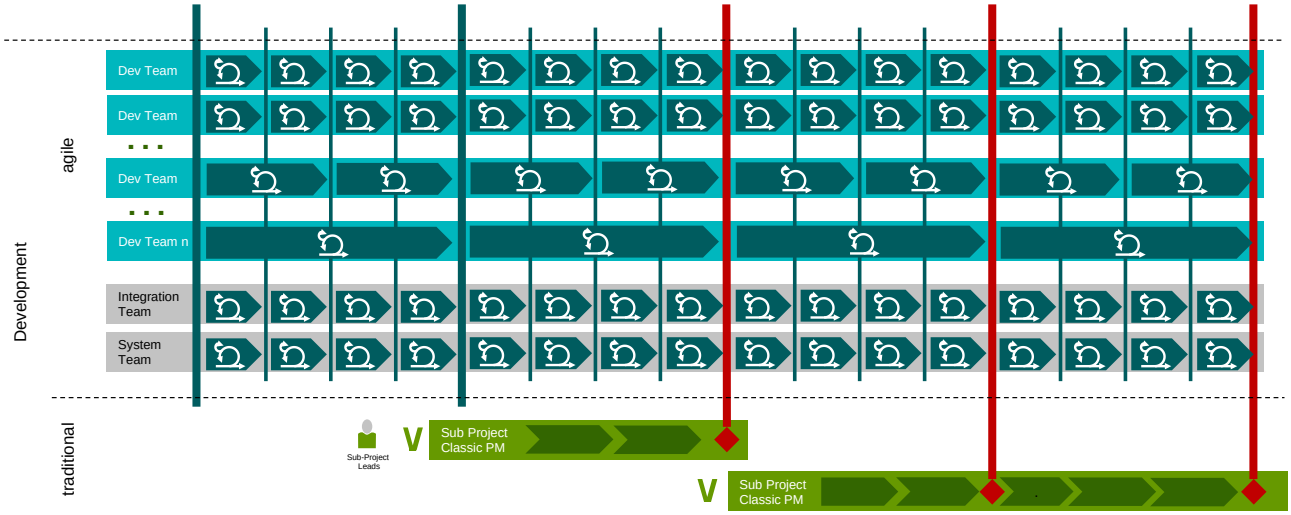


Figure 22: Synchronize after each Program Increment

There will be cases where a synchronization between program increments and traditional milestones is not possible. In this case non-aligned milestones can be accepted and must be considered during PI planning as intermediate PI objectives. However, too many non-aligned milestones can seriously impact the benefits of an agile approach. Agile SE therefore recommends avoiding non-aligned milestones as far as possible.

3.2.15 Balance between Agile and Traditional Teams

Each organization and each project are different. There is no one-size-fits-all approach when it comes to finding the right balance between traditional and agile approaches. Therefore, Agile SE provides flexibility to adapt to a variety of different contexts found in different organizations and products.

One option to achieve flexibility, is to vary the number of agile and traditional teams. More agile organizations will introduce more agile teams and only few (or none) traditional teams. More traditional organizations will have more traditional teams than agile teams. Note, that there should be at least one agile team, otherwise, Agile SE becomes a pure traditional organization that should be better managed traditionally.

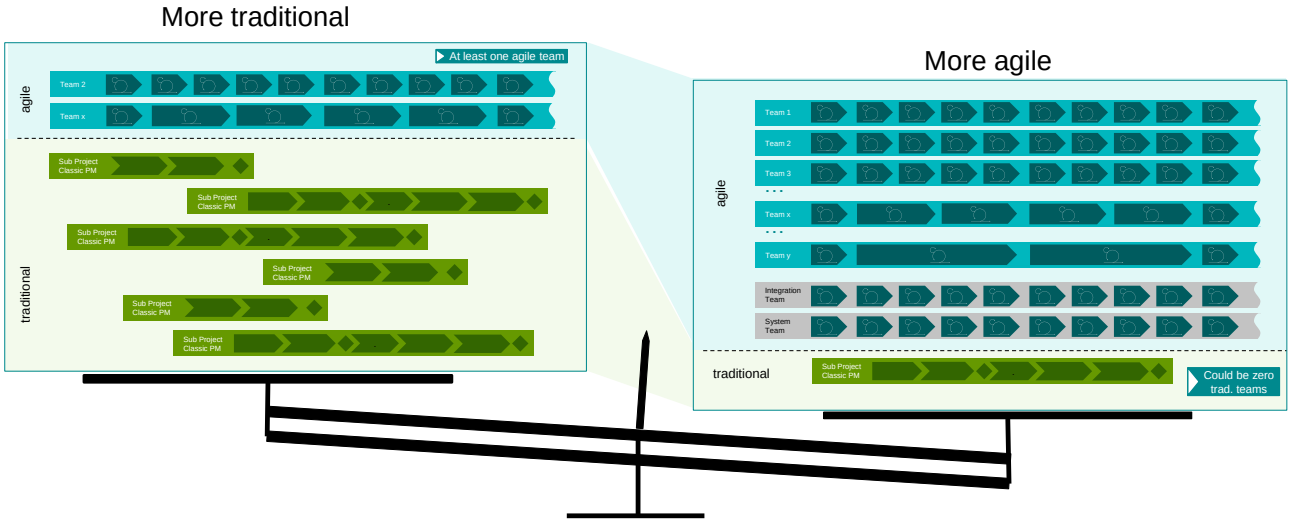


Figure 23: Balance between Agile and Traditional Teams

3.3 Organization

The Agile SE defines the following types of teams:

- Product team
- Requirement team
- Development team
- Integration team
- System team

It also utilizes the following roles:

- Sponsors
- Product architects
- Product management
- Program scrum master
- Requirement engineers
- Product owner
- Scrum master
- Developer
- Team architect

Those teams and roles are described in more detail in the subsequent chapters.

3.3.1 Product Team

The product team consists of product architects and product managers. Its main purpose is:

- Create a concept for the product.
- Define a high- and medium level product architecture.
- Define guardrails for the development teams based on that.
- Identify and prioritize features to be implemented.
- Fill the program backlog and maintain the program kanban
- Define and align PI objectives for each program increment

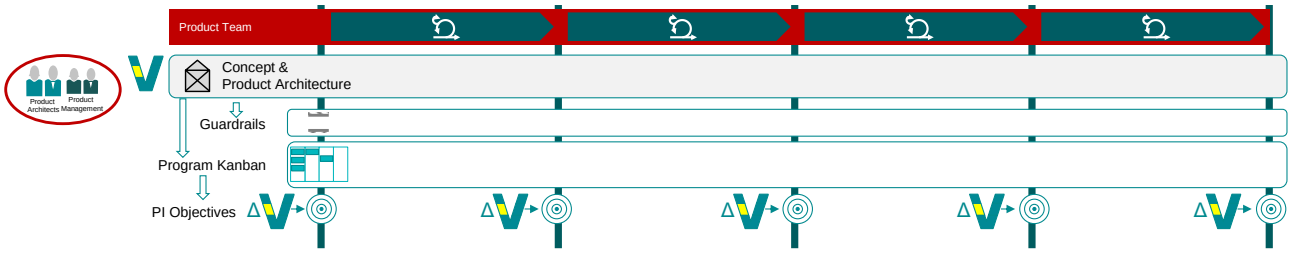


Figure 24: Product Team

3.3.2 Requirement Team

The main task of the requirement team is to break down the medium level requirements, defined in the program Kanban, into low-level requirements, often in form of user stories. It therefore closely collaborates with the agile development teams.

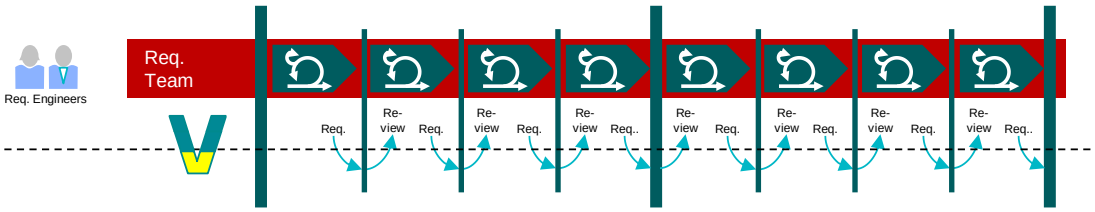


Figure 25: Requirements Team

3.3.3 Development Team

Large projects typically consist of multiple development teams. Those development teams are responsible for implementing features and components of the product. This includes defining the low-level architecture and requirements, needed to fulfill the medium level requirements. To do this, the development teams closely collaborate with the requirements team.

Additional to the traditional Scrum roles of **product manager**, **Scrum master** and **developer** (Scrum.org, 2023), Agile SE suggests adding a **team architect** to each team.

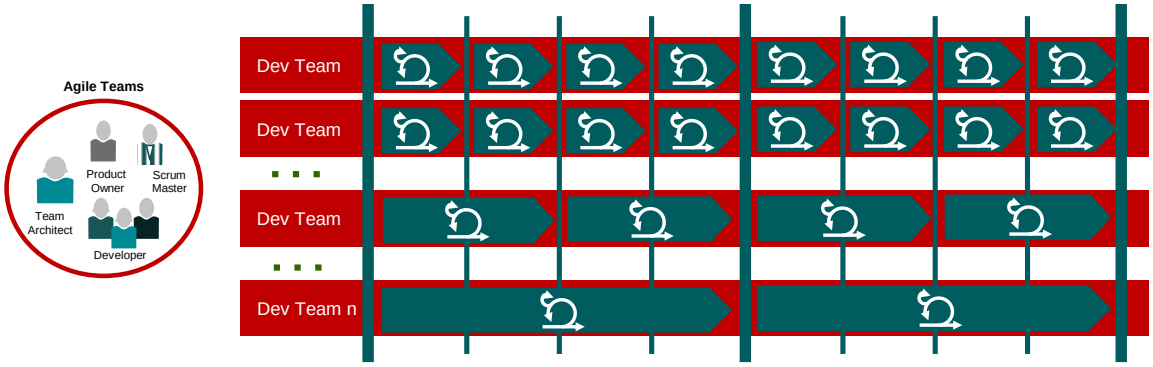


Figure 26: Agile Development Teams

Ideally, development teams are organized cross-functional. That means, each team has all the capabilities (mechanics, software, …) to deliver a certain set of features completely on their own. However, in many cases fully cross-functional teams are not possible. Teams must collaborate to deliver a certain feature due to dependencies on deliveries of other teams. Therefore, frequent cross-team alignment is required. The Agile SE framework defines two types of cross-team alignment:

- The product increment planning event (chapter 3.4.1) and
- The cross-team alignment meeting (chapter 3.2.8).

3.3.4 Integration Team

One of the key principles of Agile SE is frequent integration (chapter 0). In many scaled agile frameworks (like SAFe), integration is task of the agile development teams. However, integration can be a challenge with complex products. This specifically is true when complex simulations (chapter 3.5.2) or sophisticated technologies (rapid prototyping, chapter 3.5.3) are involved in integrations.

Therefore, Agile SE introduces an integration team, which supports the agile development teams in integrating their deliveries into the overall product. Tasks of the integration team are:

- Supporting the teams integrating their deliveries into the digital twin.
- Maintenance of the simulation infrastructure needed for integration tests.
- Executing integration tests

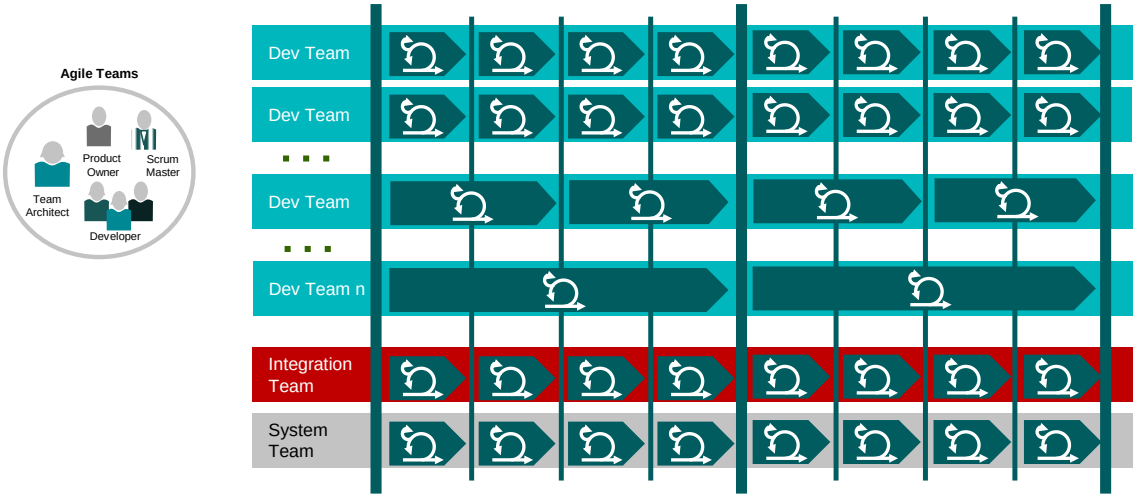


Figure 27: Integration Team

The integration team is optional. If omitted, integration is responsibility of the development teams.

3.3.5 System Team

The system team provides supporting services to the project. Its core task is to maintain and operate the IT infrastructure needed by the project, like Jira, Confluence, automatic build systems, automatic testing infrastructure and such.

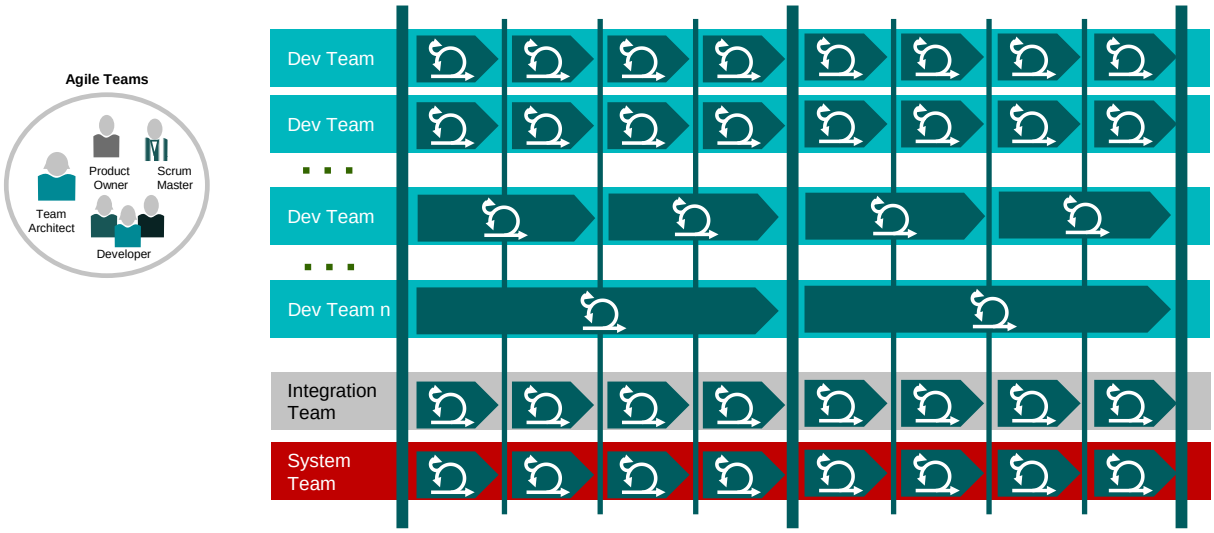


Figure 28: System Team

Some projects may decide to combine the integration- and system teams into one single team.

3.3.6 Sponsors

Sponsors are the initiators and funders of the project. They are responsible for the high-level objectives of the project, the high-level product requirements, and the high-level roadmap. They assign value to the program backlog items which allows product management to prioritize them properly. Sponsors have the final decision when it comes to prioritization of features or changes to the high-level objectives of a project.

Figure 29: Sponsors

In real-world organizations more roles are typically involved in those tasks, like marketing and strategy. These roles are not elaborated here any further, as they are highly specific to each organization and have only little relevance for agility.

3.3.7 Product Architects

Product architects are responsible for the high and medium level architecture of the product. One of their core tasks is to define a modular, change friendly architecture, which supports an agile development approach as described in Agile SE (chapter 3.5.4). They closely collaborate with the product management, but also collaborate with the sponsors, the requirements engineers, and the development teams.

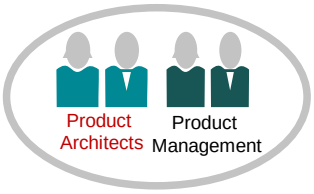


Figure 30: Product architects

3.3.8 Product Management

Product management is responsible for maximizing the value achieved by the project. They define and prioritize the features to be implemented. They are responsible for:

- Creating an overall concept for the product.
- Identifying and prioritizing features to be implemented.
- Filling the program backlog and maintaining the program kanban
- Defining PI objectives for each program increment and aligning them with the development teams

Product management closely collaborates with the product architects, but also collaborates with the sponsors, the requirements engineers, and the development teams.

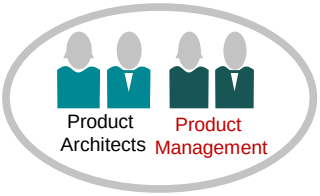


Figure 31: Product Management

3.3.9 Program Scrum Master

The program Scrum master is responsible for keeping the Agile SE development process running. He acts as a servant leader and coach and closely collaborates with the Scrum masters of the agile development team.

Figure 32: Program Scrum Master

The program Scrum master tasks are:

- Continuously improve the Agile SE development process
- Facilitate program-level events like
  - Program increment planning
  - Cross-team alignment meeting
  - Program level retrospectives
- Coaching

This role is very similar to the role “release train engineer (RTE)” as defined in SAFe (Scaled Agile, 2023).

3.3.10 Requirement Engineers

Requirement engineers are members of the requirements team and are working on the tasks of the requirements team. See chapter 3.3.2 for more details.

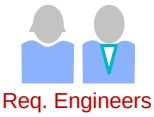


Figure 33: Requirements Engineers

3.3.11 Product Owner

Product owner is a classical role of Scrum. A product owner is responsible for maximizing the value delivered by the team with respect to the high-level project goals. As the team’s main contact person for product management and sponsors he connects the development team with business.

Together with the team and the requirement engineers he breaks down the features defined in the PI objectives and the program kanban into manageable tasks (user-stories) for the team.

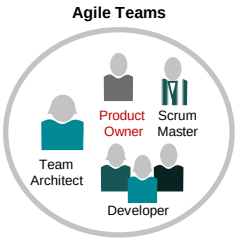


Figure 34: Product Owner

3.3.12 Scrum Master

Scrum master is another classical role of Scrum. A Scrum master acts as a servant leader to support the team in performing and continuously improving its development tasks. His tasks are:

- Facilitating team events like dailies, sprint planning or retrospectives
- Supporting sprint execution
- Supporting PI execution and PI planning
- Fostering continuous improvement of the development process of his team.

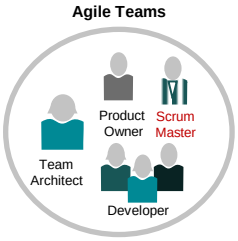


Figure 35: Scrum Master

3.3.13 Developer

Developer is a classical role of Scrum. Developers are responsible for implementing features and components of the product. This includes defining the low-level architecture and requirements, needed to fulfill the medium level requirements together with the rest of the team.

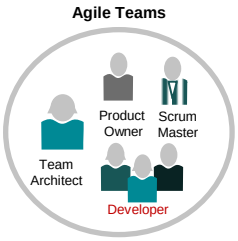


Figure 36: Developer

3.3.14 Team Architect

The role of team architect is specific to Agile SE. A team architect is responsible for breaking down the medium-level product architecture, provided by the product team, into a detailed, low-level architecture required by the agile development teams to develop a concrete solution.

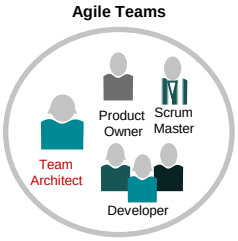


Figure 37: Team Architect

Depending on the teams' focus and the level of architecture provided by the product architect, a team architect is not always required. Therefore, the role of team architect is optional. If omitted, developers take over responsibility for breaking down the medium-level product architecture into a concrete solution.

3.4 Events

3.4.1 Program Level Events

The main program level events are:

Event	Participants	Frequency	Topics
Architecture Sync	• Product architect • Team architects	weekly	• Product architecture
Product Management Sync	• Product management • Product owners	weekly	• Product roadmap • Features • Expected business value • Prioritization
Scrum of Scrums	• Program Scrum master • Scrum masters	weekly	• Impediments • Improvements to the Agile SE process
Program Increment Planning	• All development teams • Product management • Product architect • Sponsors • Facilitator: program Scrum master	before the start of each PI	• Product vision • Architectural vision • Features planned for the next PI • PI objectives • Dependencies between the teams
Cross Team Alignment	• At least one representative of each team	weekly	• Current tasks of the team • Impediments • Dependencies, where help is needed
PI Review	• Product owner • Product management	at the end of each PI	• Achievements of the last PI
Program-Level Retrospective	• Representatives of each team • Product management • Product architect • Scrum master • Facilitator: program Scrum master	after each PI	• Review of the Agile SE process (what went well, what didn't go well?) • Improvements to the Agile SE process

Table 1: Program Level Events

3.4.2 Team Level Events

Most team level events for the agile development teams are standard Scrum events. More details can be found at Scrum.org (Scrum.org, 2023). They are:

Event	Participants	Frequency	Topics
Requirements Alignment-Meeting	• Requirements engineers • Product owner • Development team	Weekly	Breaking down the medium level requirements, of the program kanban, into low-level requirements (often in form of user stories). (In Scrum this meeting is often called "backlog refinement")
Sprint Planning	• Developers • Product owner • Scrum master	Before each sprint	Deciding on the backlog items to be implemented in the next sprint

Daily Scrum	• Developers • Product owner • Scrum master	Every day	Tasks in progress, impediments
Sprint Review	• Developers • Scrum master • Product owner	At the end of each sprint	Review the deliveries of each sprint
Retrospective	• Developers • Scrum master • Product owner	After each sprint	• Review of the team's way of working • Improvements to the team's way of working

Table 2: Team level events

3.5 Enablers

3.5.1 Agile Culture

Establishing an agile culture is essential for Agil SE. Agility needs agile leaders – people who understand the value of agile approaches and who are willing to foster and coach the organization in adopting an agile mindset. The agile manifesto states: “Responding to change over following a plan” (Beck, et al., 2001), and this is a key value for Agile SE.

Teams should work on the overall goal, not on individual goals. They should focus on value for the final product, not on local optimizations. An open cross-team communication must be established. A good communication culture, where “individuals and interactions are more important than processes and tools” (Beck, et al., 2001) is essential for Agile SE. Transparency and openness is key to improve the whole system. Silos must be avoided.

3.5.2 Modeling and Simulation

Agile SE postulates frequent integration in order to shorten feedback loops (chapter 0). However, frequent integration can be a challenge – specifically when physical components are involved.

Working with a virtual product (“Digital Twin”) where possible can be an option. Integration is done virtually instead of physically, allowing much shorter feedback cycles. This requires high quality and connected models of the product and a solid simulation framework to do virtual integration tests. It is task of the integration team (chapter 3.3.4) to support this.

3.5.3 Rapid Prototyping

When modeling and simulation are not suitable, rapid prototyping (e.g., 3D printing) is another option. In many cases, prototypes created by rapid prototyping technologies can be integrated into the overall product for testing purposes.

3.5.4 Modular, Change Friendly Product Architecture

High complexity can inhibit refactoring and prevent iterative learning and quick responses to change. A modular, change friendly product architecture can reduce complexity. It is the responsibility of the product architects to create a flexible, modular product architecture, that supports refactoring and change.

3.5.5 Test Automation

Frequent integration means, frequent and repeated testing. Since tests can be quite expensive, repeated tests should be automated as far as possible.

For software highly automated build-, test- and deployment-chains are state-of-the-art. They should be established wherever possible. They can provide quick feedback, ideally directly after each commit.

When modeling and simulation are used for integration tests and other tests, those tests can be often highly automated. Automation servers (like Jenkins) can be utilized to frequently and automatically configure, execute, and evaluate simulations of the checked-in models.

4 Agile SE along the PDP

The Agile SE framework described so far assumes a steady state development process and project organization. However, large projects often follow a standardized product development process (PDP) with distinctive phases, each having different development focuses.

A typical product development process looks like this:

Figure 38: Example of a Product Development Process

Agile SE does not stay constant across those phases of the PDP. It must adapt to the different phases. Figure 39 shows a big-picture, how Agile SE can be applied to a typical PDP:

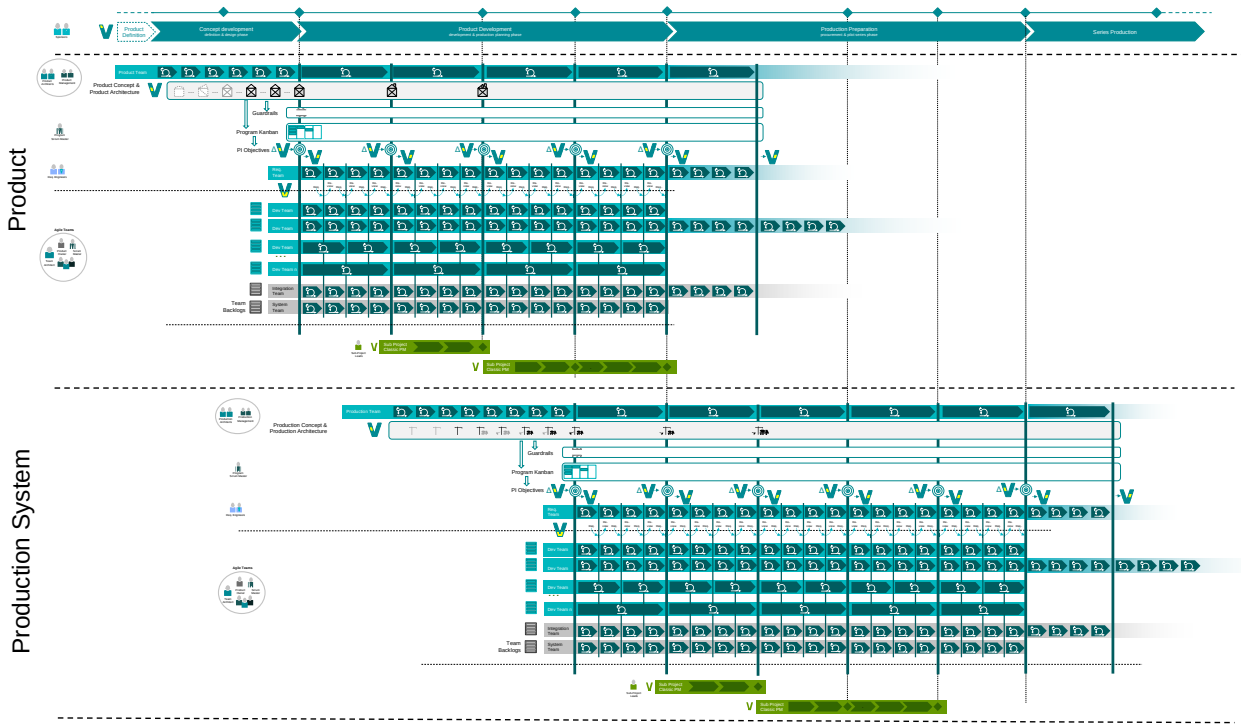


Figure 39: Agile SE Along the PDP

This big-picture is described in more detail in the subsequent chapters.

4.1 Key Principles

4.1.1 Set-Up Independent Agile SE Organizations

Development of the **product** and development of **production** are typically too different to be combined. They also typically occur at different times along the PDP. So, it makes sense to set-up independent Agile SE organizations for development of the product and for development of the production.

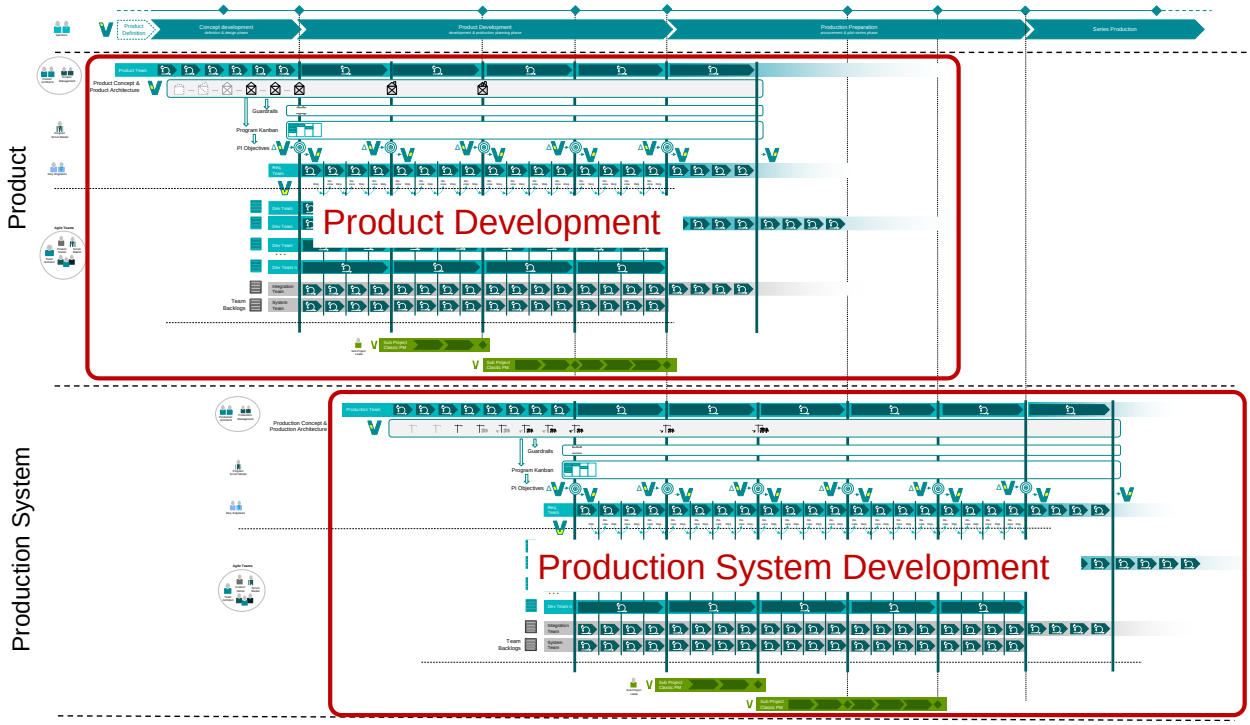


Figure 40: Independent Agile SE Organizations

In certain cases, even more Agile SE organizations can be feasible. If, for example, significant development work is needed to prepare after-sales or support, it might be reasonable to setup another, independent Agile SE organization for that.

Setting up independent Agile SE organizations is closely related to the concept of “Agile Release Trains (ARTs)” in SAFe. (Scaled Agile, 2023)

4.1.2 Ramp-Up Agile SE Organizations

Agile SE organizations cannot start from zero to hundreds. They require a ramp-up phase which typically aligns with a phase or a milestone on the roadmap. During ramp-up, product concepts and product architecture are iteratively evolved by the product team. To speed up this process, the product team works in short sprints. Once the Agile SE organization is set-up and the agile development teams start their development work, the product team changes to program increments. Optionally it could also keep its pace.

Figure 41 shows the ramp-up phase of an Agile SE organization for the example “product development”.

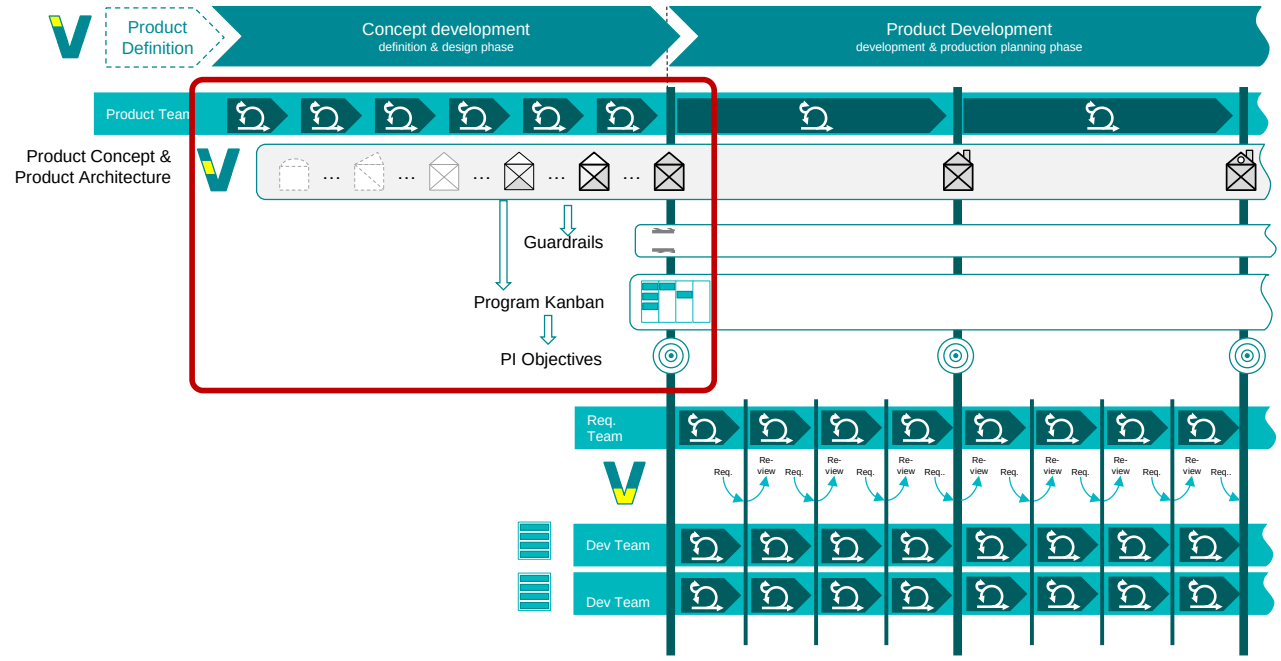


Figure 41: Ramp-Up Phase of an Agile SE Organization

Ramp-up for **production** development is analogous. Instead of a product team a dedicated **production team** starts evolving the **production concept** and the **production architecture** iteratively in sprints.

4.1.3 Full Working Mode of Agile SE Organizations

The full working mode of an Agile SE organization is described in detail in chapter 4.1.3. During this phase the product concept and product architecture evolving much slower than during the ramp-up phase.

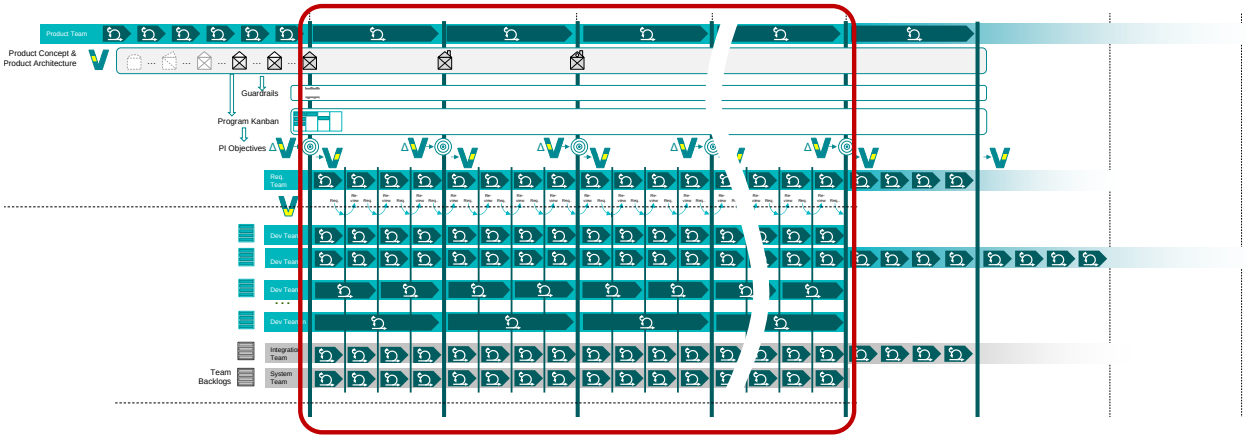


Figure 42: Full Working Mode of an Agile SE Organization

4.1.4 Ramp-Down of Agile SE Organizations

At some point in time, development of the product of an Agile SE organization is mainly done. Other organizations take over. Typically, there is still some work left, like defect-fixing or late change requests, but the amount of work continuously decreases.

During this phase the Agile SE organization phases out: teams are reduced, suspended, or reorganized into fewer teams. At the end of a ramp-down only a few small teams remain, who are quite often mainly involved in support.

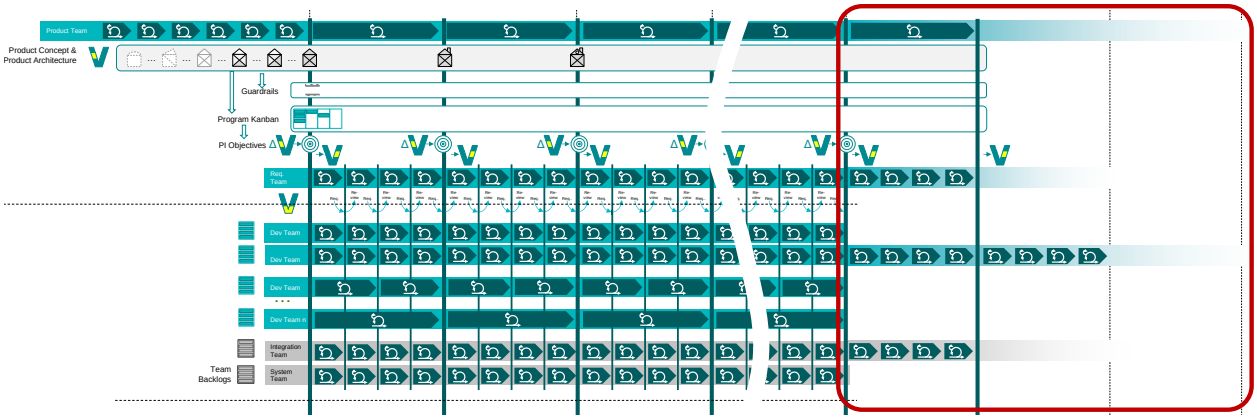


Figure 43: Ramp Down of an Agile SE Organization

4.2 Agile SE Along the Phases

Figure 44: Product Development Process

During the p**roduct definition** phase marketing, sponsors, strategists, and others decide if a new product development should be launched. In this phase no Agile SE organization exists.

During the **concept development** phase, the Agile SE organization for **product development** is being ramped-up. The Agile SE organization for production system development typically does not exist during this phase.

During the **product development** phase, the Agile SE organization for **product development** is in full operation mode as described in chapter 4.1.3. At some point of time during this phase, typically at a milestone like “Design Freeze”, the Agile SE organization for **production system development** starts being ramped up.

During the **production preparation** phase, the Agile SE organization for **production system development** is in full operation mode. The Agile SE organization for **product development** still exists to support defect-fixing or late product changes, but the amount of work reduces over time. It is ramped down.

During **series** p**roduction** standard production organizations like factory sites take over. The Agile SE organization for **production system development** still exists to hand over production and provide support, but the amount of work reduces over time. It is ramped down.

5 Comparison with SAFe

Agile SE is inspired by SAFe (Scaled Agile, 2023), but it is not SAFe. There are many similarities with SAFe, but also some distinctions.

Table 3 shows the similarities between Agile SE and SAFe 6. Table 4 shows the differences.

Agile SE	SAFe
Roadmap	Roadmap
Program Increment (PI)	Planning Interval (PI)
Program Increment Planning (PI)	PI Planning (PI)
PI Review	Inspect & Adapt (I&A)
Sprint	Iteration
Agile SE Organization	Agile Release Trains (ART)
Product Team	Value Stream Management
Requirements-Engineering & Product Architecture	Agile Product Delivery
Agile Development	Team and Technical Agility
Product Management	Product Management
Product Architects	System Architects
Program Scrum Master	Release Train Engineer (RTE)
Product Owner	Product Owner
Scrum Master	Scrum Master
Developer	Agile Team Member
System Team	System Team
Program Kanban	ART Backlog
Team Backlog	Team Backlog
Guardrails	Guardrails
Product Concept	Vision
Agile Culture	Lean Agile Mindset
Roadmap	Roadmap

Table 3: Similarities between Agile SE and SAFe 6

Aspect	Agile SE	SAFe
Portfolio Management	not covered	covered
Enterprise Solution Delivery	not covered	covered
Alignment with V-Model	in strong focus	not covered
Value Management, Business Value	not covered	in strong focus
Requirements Team	dedicated team	covered by product management and product owners

Integration Team	dedicated team	covered by development teams or system team
Team Kanban	not covered	optional
PI Objectives	defined by product team as input for PI planning (goal)	defined by development teams as outcome of PI planning (commitment)
DevOps	not covered	covered

Table 4: Differences between Agile SE and SAFe 6

6 Summary

Agile SE is a framework to combine agility and systems engineering.

Agile SE utilizes nested iterations: the higher levels of the V-Model are managed by a high-level roadmap, the medium levels are managed by long “program increments” and the low levels are managed by short “sprints”. Different teams work on the different levels. Agile development teams closely collaborate with product- and requirement teams to bridge between the lower and medium levels of the “V”. To adapt to constraints of specific technical domains, Agile SE allows sprint lengths of a multitude of a basic sprint length. In any case, teams and iterations should stay in cadence.

PI objectives and guardrails provide flexibility to the teams to find the detailed solution on their own, while still providing enough control to ensure, that the final product stays on track. This unleashes the creativity of the developers and typically leads to better results.

Integration is done frequently to shorten feedback loops. This increases agility and allows quick responses on the unavoidable unforeseen. Modeling and simulation, 3D printing, and test automation are enablers for short feedback loops. Artificial intelligence can further support this.

A modular, change friendly product architecture is required to reduce complexity and foster change.

A high-level roadmap provides guidance and a vision to keep the product development on track. Mature agile organizations will adjust the roadmap based on the outcomes of the program increments. Milestones should ideally be aligned with program increment boundaries, but Agile SE provides options to handle cases, where this is not feasible.

Agile SE supports real-world situations, by explaining how teams, managed by traditional project management, can be integrated. It is flexible enough to adapt to different contexts in different projects or organizations. Agile SE can be adjusted from a full agile organization to a quite traditional organization managed by traditional project management.

Agile SE changes over time during a typical product development process (PDP). In many projects multiple Agile SE organizations are required. Typically, one for product development and one for production development. Each Agile SE organization has a ramp-up phase, a full working mode phase and a ramp-down phase. Those phases can be aligned to the milestones of a PDP.

Agile SE is closely related to SAFe but is not SAFe.

7 Outlook

This whitepaper is partly based on insights gained from interviews held with agile and systems engineering experts from the industrial sector (see chapter 2). Due to the small number of interview partners, those interviews were not representative. Further studies should conduct additional interviews to gain broader feedback.

The Agile SE approach described in this white paper is mainly focused on the industrial sector. While this doesn't mean, that the described Agile SE approach is unsuitable for other sectors, further studies should investigate which adaptions are needed for other sectors, such as health care, finance or construction.

8 References

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, W., . . . Thomas, D. (2001). *Agile Manifesto.* Retrieved from https://agilemanifesto.org/

Scaled Agile. (2023, Aug 2). *Agile Release Train.* Retrieved from SAFe 6.0: https://scaledagileframework.com/agile-release-train

Scaled Agile. (2023, Aug. 01). *Release Train Engineer.* Retrieved from SAFe 6.0: https://scaledagileframework.com/release-train-engineer/

Scaled Agile. (2023, July 28). *Scaled Agile Framework.* Retrieved from SAFe 6.0: https://scaledagileframework.com/

Scrum.org. (2023, 07 31). *What is Scrum.* Retrieved from Scrum.org: https://www.scrum.org/learning-series/what-is-scrum

prostep IVIP

prostep ivip association

Dolivostraße 11
64293 Darmstadt
Germany

Phone +49-6151-9287336
Fax +49-6151-9287326

psev@prostep.com
www.prostep.org

ISBN 978-3-948988-35-7
Version 1.0, 2024-3