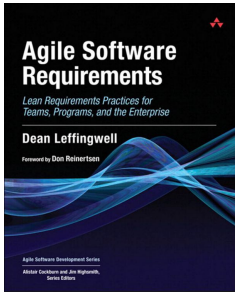
CE Department

Software Requirements Engineering

40688

These slides are designed to accompany Agile Software Requirements (2011) by Dean Leffingwell and support the university course Software Requirements Engineering, instructed by Mehran Rivadeh. Created and designed by Mahnaz Rasekhi.



Agile Software Requirements (2011)
Dean Leffingwell

Agile Requirements For The Program

Chapter 4

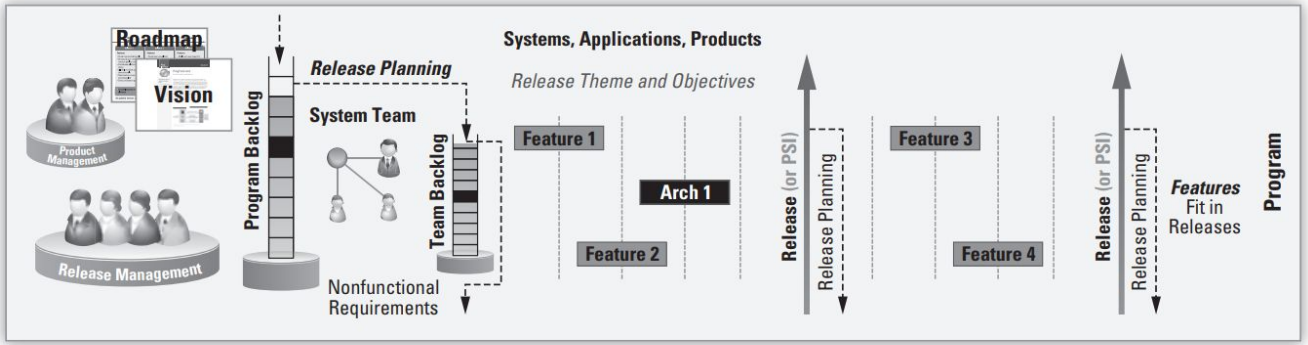
Mehran Rivadeh
mrivadeh@sharif.edu
Software Requirements Engineering
October 2025 - Fall 1404 - SUT

Contents

1. Introduction To The Program Level

The Program Level

➔ At the program level, we see an organizational, process, and requirements model that provides mechanisms to harness some number of agile teams to a larger enterprise purpose—the delivery of an entire product, system, or application suite to the customers.



The Program Level

- At the Team level:
  - **Teams** are empowered and are largely **self-organizing** and **self managing**.
  - They work from a **local backlog** that is under the purview of the team's product owners.
  - They **have control of** their **local destiny** and **can define**, **build**, and **test their feature** or **component**.
  - In accordance with the principles of the Agile Manifesto, that is the **optimum mechanism** for **incentivizing** and **motivating** a **team** to **produce the best possible results**.
- At the Program level, however, the problem changes, and the enterprise faces an additional set of challenges to successfully execute agility at this next level of scale.

The Objectives Of The Program Level

- ➔ Maintaining vision and Roadmap
- ➔ Release management
- ➔ Quality Assurance
- ➔ Deployment
- ➔ Resource management
- ➔ Eliminating impediments

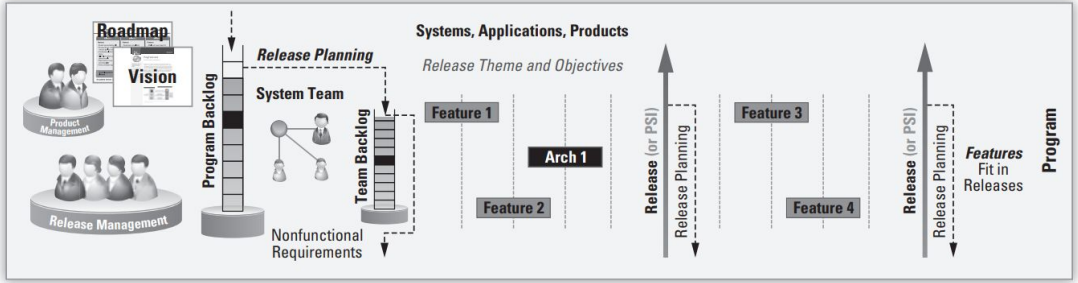


Figure 4–1 The Program level of the Big Picture

Contents

2. **Organizing Agile Teams At Scales**

Organizing Agile Teams At Scales

How to organize the agile teams in order to optimize value delivery of requirements?

➔ For the smaller enterprise, this is usually no issue at all.

- They will organize naturally around the few products or applications that reflect the mission.
- The silos that tend to separate development, product management, and test in the larger enterprise do not exist (ideally!).
- The teams are probably already co-located, rather than being distributed across disparate geographies.
- Creating an agile team in this context is mostly a matter of deciding what roles the individuals will play and rolling out some standard training.

Organizing Agile Teams At Scales

How to organize the agile teams in order to optimize value delivery of requirements?

- ➔ At scale, however, like most other things agile, the problem is different, and the challenge is to understand who works on what and where.
- ➔ Do we organize around features, components, product lines, services, or what?
- ➔ Although there is no easy answer to this question, the question must be explored because so many **agile practices** must be reflected in that decision.
  - How many backlogs there are and who manages them.
  - How the vision and features are communicated to groups of teams
  - How the teams coordinate their activities to produce a larger solution.

Contents

Component Teams

- ➔ It is a typical organizational model whereby **many** of the **agile teams** are **organized around the architecture of a larger-scale system**.
- ➔ They leverage their technical skills and interest and focus on building robust components, making them as **reliable** and **extensible** as possible.
- ➔ They leverage **common technologies** and usage models, and **facilitating reuse.**
- ➔ In a component-based approach, the development of a new feature is implemented by the affected component teams.
- ➔ **A new feature** requires the creation of **new backlog items** for each team that contributes to the feature.

Component Teams

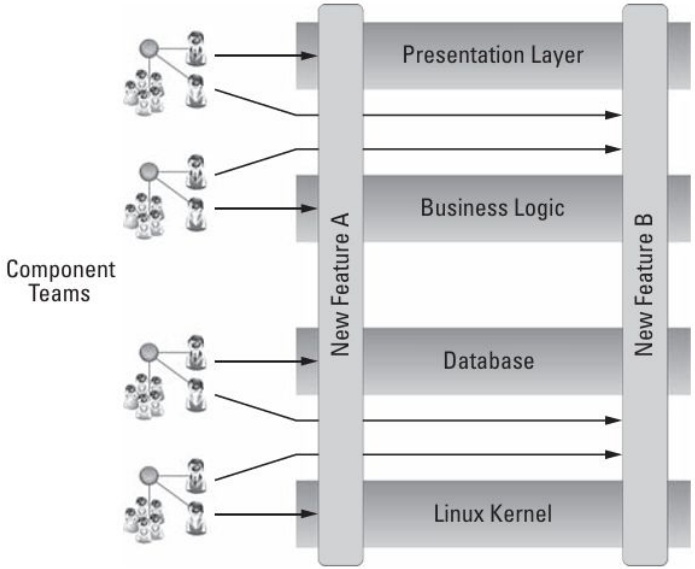


Figure 4–2 Component teams often have responsibility for a single layer in an architectural stack.

Component Teams

Advantages

- ➔ Component teams **minimize multiplexing** across features by implementing them in series, rather than parallel.
- ➔ Each team is able to aggregate the needs of multiple features into the architecture for their component and can focus on building **the best-possible, long-lived component or service for their layer**.
- ➔ Their component **does not get "sliced and diced" by each new feature**; rather, it evolves as a set of services to implement current and, ideally, future features.

Component Teams

Why component-based organizations can be effective in the agile enterprise?

- ➔ Based on its past successes, **the enterprise may already organized that way**, with specialists who know large-scale databases, web services, embedded operating systems, and the like, working together.
- ➔ **Individuals**—their skills, interests, residency, friendships, cultures, and lifestyles—**are not interchangeable**.
- ➔ These teams may already be **co-located**, simplifying communication and reducing the batch handoff of requirements, design, and test data.

Component Teams

Why component-based organizations can be effective in the agile enterprise?

- ➔ **Technologies** and **programming languages** may **differ across components**, making it difficult for feature teams to do pairing, collective ownership, continuous integration, test automation, and other factors.
- ➔ At scale, a **single user feature can** be an awfully big thing that could easily **affect hundreds** of **practitioners**.
  - ◆ For example, a feature like “share my new phone video to YouTube” could affect dozens of agile teams, so organizing by feature can be nebulous when multiple teams are required for implementation.

Feature Teams

- ➔ The almost **universally accepted approach** for organizing agile teams is to **organize around features**.
- ➔ “Feature-based delivery means that the engineering teams build features of the final product.”

Highsmith [2004]

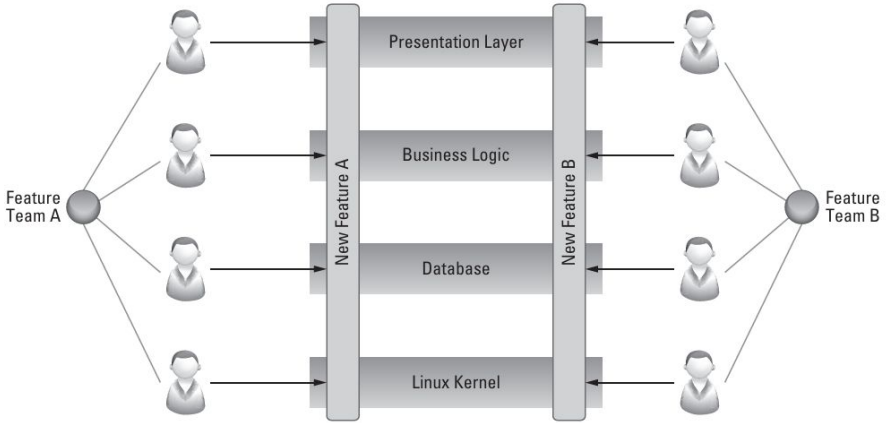


Figure 4–3 Feature team approach across an architectural stack

Feature Teams

Advantages

- **Teams build expertise** in the actual domain and usage mode of the system and can typically accelerate value delivery of any one feature.
- **There is less overhead**, because teams don't have to pass backlog items back and forth to see that a feature is implemented; there are far fewer interdependencies across teams.
- **Planning** and **execution** are **leaner**.
- **The team's core competence becomes the feature** (or set of features), as opposed to one element of the technology stack.
- **The team's backlog is simplified**, just one or two features at a time. That has to promote the fast delivery of high value-added features!

Sometimes The Line Is Blurry

- Features and components are both abstractions, and the line is not so clear.
- One person's feature may be another's component.
- And sometimes a single feature may best be implemented as a standalone, service-oriented component.

Sometimes The Line Is Blurry

Example:

Imagine a Bakery App. The app has several features like:

- ➔ Ordering cakes
- ➔ Viewing the menu
- ➔ Tracking deliveries
- ➔ Customer reviews

Feature And Component Teams

Example (Cont.):

Now, to build the “Ordering cakes” feature, you need:

- ➔ A user interface (where customers choose cakes)
- ➔ A payment system
- ➔ A database to store orders
- ➔ A notification system to confirm the order

Each of these parts (UI, payment, database, notifications) are components.

They’re reusable building blocks that can be used in other features too (like “Tracking deliveries”).

Feature And Component Teams

Example (Cont.): So what's the confusion?

- ➔ Let's say you have a team working on the Ordering cakes feature. You might call them a feature team.
- ➔ But someone else might say, "Wait — they're just working on the payment system and notifications — aren't those components?"
- ➔ And here's the twist: sometimes a feature (like "Ordering cakes") is so big and important that it becomes its own component — like a "Cake Ordering Module" that other apps or features can use.
- ➔ In complex systems (like online trading or even our bakery app), it's not always practical to form one team with experts from every area.

Lean Toward Feature Teams

- Given the advantages and disadvantages to each approach, the answer is not always obvious.
- But with agile's focus on immediate value delivery, there is an appropriate leaning toward feature teams.
- "I tend to start with the feature team approach and only move toward components if I have to . . . but the decision is situation-specific."

Mike Cottmeyer

Lean Toward Feature Teams

To make this decision,

- ➔ You'll have to explore the diversity of your technology.
- ➔ How well your system is designed.
- ➔ What tools you have to manage your code base.
- ➔ The size and competence of your team.
- ➔ How and where your teams are distributed.
- ➔ The quality of your infrastructure automation.
- ➔ You need to take a hard look at what scale your feature teams WILL break down.
  - ◆ Is scaling to this level something we need to address now or can it wait?

The Best Answer Is Likely A Mix

- In the larger enterprise where there are many teams and many, many features, one should consider the previous factors and then select the best strategy for your specific context.
- In most cases, as you can see, the answer will likely be a mix of feature teams and component teams.
- Even in the modest-sized agile shop, a mix is likely to be appropriate.
- Given that a mix is most likely appropriate, there are two main factors that drive the mix:
  - The practical limitation of the degree of specialization required
  - The economics of potential reuse.

The Best Answer Is Likely A Mix

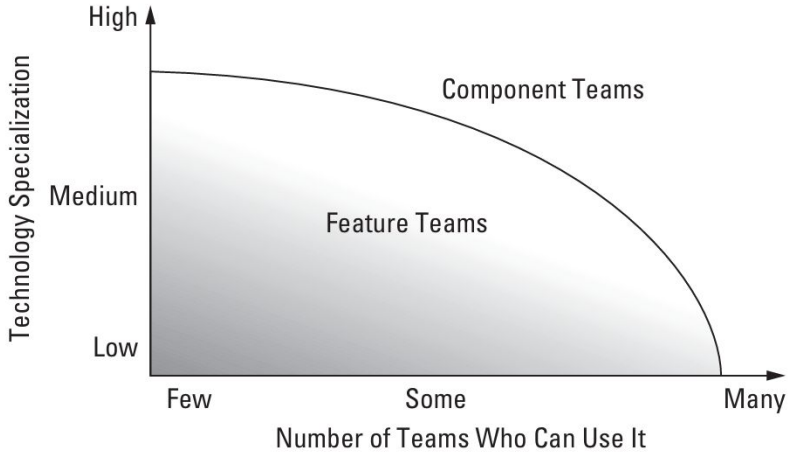


Figure 4–5 Organizing around feature and component teams

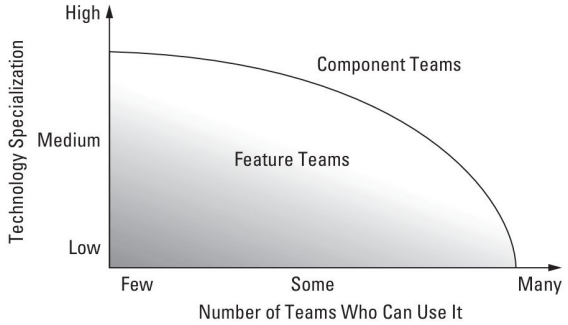
Consider Co-Location

Are you still unsure about the best way to organize?

- ➔ You may want to optimize around co-location.
- ➔ Because the communication, team dynamics, and velocity benefits of co-location may well exceed the benefits of the perfect component or feature organization.

Consider Co-Location

- First, apply feature teams if they are, or can readily become, co-located.
- If not, and you find yourselves on the outer edges of the curve in the figure, then apply component teams, especially since it is likely that the teams are already organized that way.
- As the organization evolves, you'll have lots of opportunity to inspect and adapt your model and evolve your organization in a way that makes sense to you.
- But, in the meantime, maybe nobody will have to move.



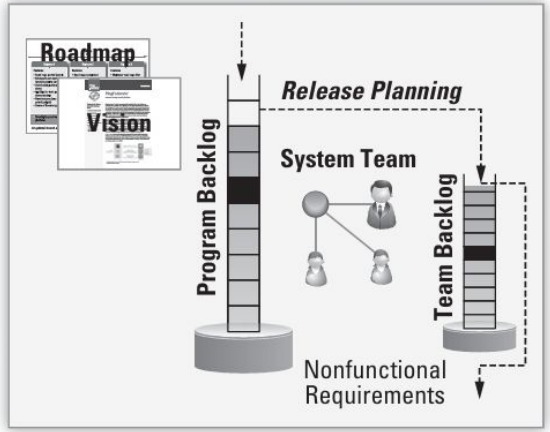
Contents

The System Team

- ➔ Agile teams are the software production engines that create and test the code.
- ➔ Each team should have the requisite skills and assets necessary to specify, design, code, and test the component or feature of their domain.
- ➔ At the Program level, however, these individual teams may not have all the capabilities to integrate, test, and deploy a full solution.
- ➔ Therefore, we often observe an additional team that complements the feature/component teams.

The System Team

- The name of the team varies; it could be called
  - System Integration
  - QA and Deployment
  - Release Team
  - System Team
- This team shares the same mission.
- They work on the same release train cadence.
- It typically has a set of specific, system-level set of responsibilities.



The System Team

System-Level Testing

- ➔ Ideally, each team would have the ability to test all the features at the system level.
- ➔ Many feature teams do have such capabilities, and that's one of the reasons why feature teams work so well.
- ➔ However, the fact is that it is often not practical for an individual feature or component team to be able to test a feature in its full system context.
- ➔ Many teams may not have the local resources (test bed, hardware configuration items, other applications, access to third-party data feeds, production simulation environment) necessary to test a full system.

The System Team

System-Level Testing

- Moreover, at scale, many teams are developing interfaces, infrastructure components, drivers, and the like, and they may not even have an understanding of the full scope of the system feature that drove their new functionality to exist.
- In this case, the system team builds the skills and capabilities to perform the more extensive end-to-end behavior of the larger features and use cases that deliver the ultimate value.

The System Team

System Quality Assurance

- ➔ Many teams do not have the specialty skills and resources necessary to test some of the nonfunctional and other quality requirements for the system.
- ➔ This may include load and performance testing, reliability testing, conformance to industry compliance standards, and so on.
- ➔ Simply running a full validation suite on a large-scale system may even require a small but dedicated team who constantly updates the full system verification and test platforms and runs the validation.
- ➔ The system team may also be the only practical means to test against the “matrix of death”—the umpteen odd variants that occur in the customer’s various, supported platform and application environments.

The System Team

System-Level Continuous Integration

- ➔ As systems grow larger, teams and their existing build and configuration management environments struggle more.
- ➔ It's less likely they can independently integrate all aspects of the solution.
- ➔ This makes daily full system builds harder to achieve.

The System Team

Building Development Infrastructure

- ➔ The transition to agile methods typically involves a significant investment in the environment to support configuration management, automated builds and deployment, and automated build verification tests (faster feedback).
- ➔ This may involve analysis, procurement of tools and systems, deployment, scripting, ongoing maintenance, and so on.

The System Team

Building Development Infrastructure

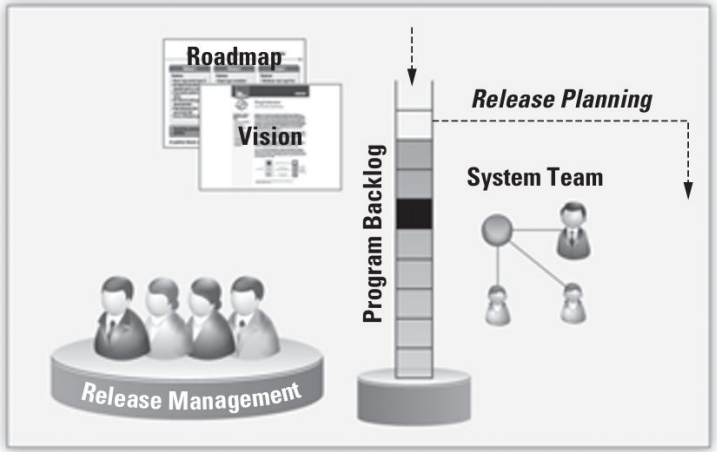
- ➔ This is a complicated and technical set of tasks that takes time and dedicated software development–capable resources.
- ➔ Building an initial infrastructure (system) team that is integral to the system release train process is one way to assure the commitment, visibility, and accountability of those resources.
- ➔ More importantly, it helps assure that the job will actually get done, because the program depends upon its success.

Contents

2. **Organizing Agile Teams At Scales**

The Release Management Team

- In addition to the agile teams and the system team, there is typically another significant organizational unit.
- There is no standard convention for its name, but it takes on **a release management team** or **steering committee function** [Leffingwell 2007].
- This team exists because, even though empowered, the agile teams do not necessarily have the requisite visibility, quality assurance, or release governance authority to decide **when** and **how the solution should be delivered to the end users**.



The Release Management Team

Members of this team may include key stakeholders of the Program level of the enterprise:

- **Line-of-business owners** and **product managers**

  Who focus on the content and market impact of the release.

- **Senior representatives**

  Who are from sales and marketing.

- **Senior line managers**

  Who are responsible for the teams.

  Are accountable for developing the solution for the marketplace.

The Release Management Team

Members of this team may include key stakeholders of the Program level of the enterprise:

- **Internal IT** and production deployment resources
- **Senior** and **system-level QA personnel**

  who are responsible for the final assessment of the solution's system-level quality, performance, and suitability for use

- **System architects**, **CTOs**, and others who oversee architectural integrity

The Release Management Team

In many agile enterprises, **this team meets weekly** to address the following questions.

- ➔ Do the **teams** still clearly **understand their mission**?
- ➔ Do we understand **what they are building**?
- ➔ What is the **status of the current release**?
- ➔ What **impediments** must we address to facilitate progress?
- ➔ Are we likely to meet the **release schedule**?
- ➔ If not, **how** do we adjust **the scope to assure** that we can **meet the release dates**?

The Release Management Team

- This forum provides weekly senior management visibility into the release status.
- This team also has the requisite authority to make any scope, timing, or resource adjustments necessary to help assure the release.
- In this manner, the release management team represents the final authority on all release governance issues and is an integral part of the agile enterprise.

Contents

Product Management

At The Team Level

➔ **The product owner** is responsible for defining what stories the team implements, and the order in which they are implemented, in order to deliver end-user value.

At The Program Level

➔ There are **another set of stakeholders** who have the same responsibility, but for the solution as a whole.

Product Management

- These stakeholders may have different titles, such as
  - Product Manager
  - Program Manager
  - Solution Manager
  - Business analyst
  - Area or Line Product Owner

  or whatever.

Product Management

The responsibility is clear:

- ➔ They are ultimately responsible for **the end to-end solution**.
- ➔ This includes not only the content of the release but the additional requirements for the “whole-product surrounds”.
- ➔ such as distribution, documentation, support, messaging, release governance, and so on.

Contents

3. Vision

Vision

The vision addresses the larger questions:

- ➔ What is **the strategic intent** of this program?
- ➔ What **problem** will the application, product, or system solve?
- ➔ What **features** and **benefits** will it provide?
- ➔ **For whom** does it work?
- ➔ What **performance**, **reliability**, and so on, will it deliver?
- ➔ What **platforms**, **standards**, **applications**, and so on, will it support?

Vision

Agile teams take a variety of approaches to communicating the Vision:

- ➔ Vision document
- ➔ Draft press release
- ➔ Preliminary data sheet
- ➔ Backlog and Vision briefing

Contents

4. Features

Features

- The primary content of a Vision is a set of features.
- They describe **new things the system will do for its users** and the **benefits the user will deliver**.
- In describing the features of a product or system, we take **a more abstract and higher level view of the system of interest**.
- “Features are services provided by the system that fulfill stakeholder needs.” Leffingwell [2003]

Features

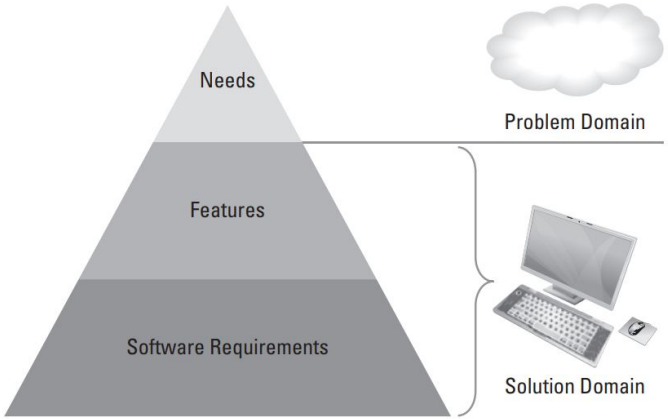
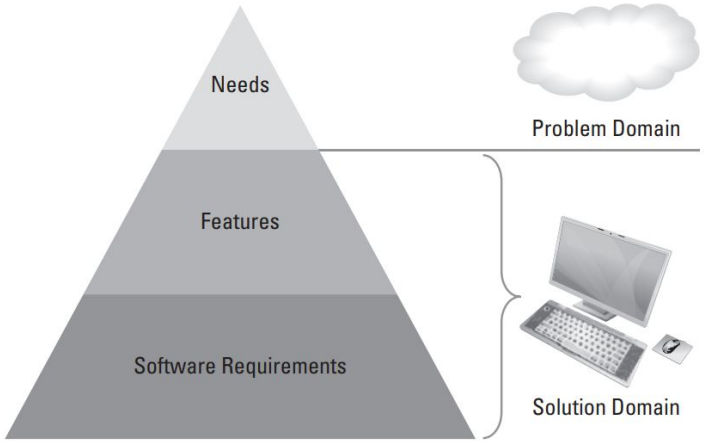


Figure 4–6 Traditional requirements pyramid

- **Problem domain:** understanding the needs of the users and stakeholders in the target market
  - What's wrong or missing in the world.
- **Solution domain:** specific requirements intended to address the user needs
  - What you're building to fix it.

Features



Example: Online Food Delivery App

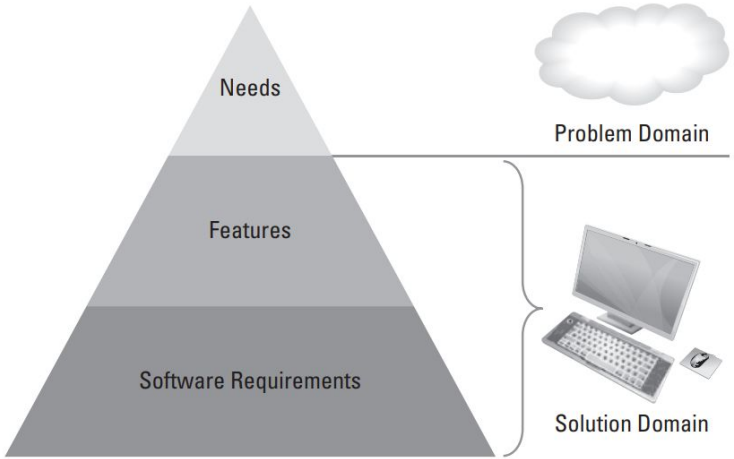
- ➔ **Problem Domain**
  - Understanding the user's needs, pain points, and goals:
  - "People want to order food quickly and reliably from their phones."
- ➔ **Solution Domain**
  - Designing features and systems to solve those problems:
  - "Build a mobile app with GPS tracking, payment integration, and restaurant menus."

Features

Example: Online Food Delivery App

Problem Domain	Feature	Solution Domain
Users want to know when their food will arrive	Real-time delivery tracking	Use GPS and mapping APIs to show driver location
User hates entering their address every time	Saved delivery addresses	Implementing user profile with address storage
Users want to reorder their favorite meals easily	Reorder button	Store order history and enable quick repeat orders
Users want to pay securely	Multiple payment options	Integrate with Stripe, paypal, and local gateway

Features



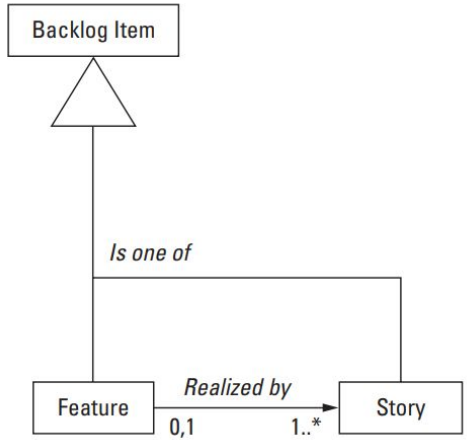
- **Problem domain** is about empathy and understanding.
- **Features translate** that empathy into actionable ideas.
- **Solution domain** turns those ideas into working code and systems.

Features

- ➔ A system of arbitrary complexity can be described with **a list of 25 to 50 features** (just like a program backlog).
- ➔ This simple rule of thumb allows us to **keep** our **high-level descriptions** exactly that—high level—and **simplifies our attempts to describe complex systems** in a short form while still communicating the full scope and intent of the proposed solution.

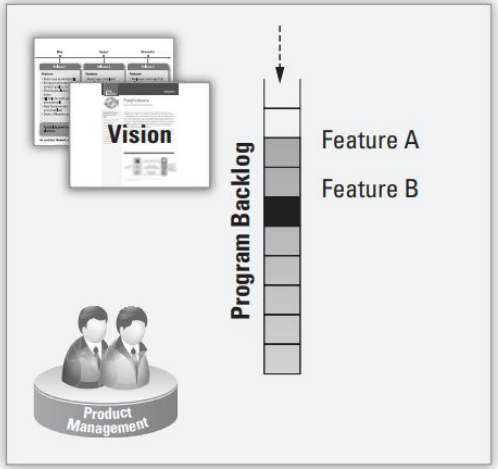
New Features Build The Program Backlog

- ➔ Features are realized by stories.
- ➔ At release planning time, features are decomposed into stories, which the teams use to implement the functionality of the feature.



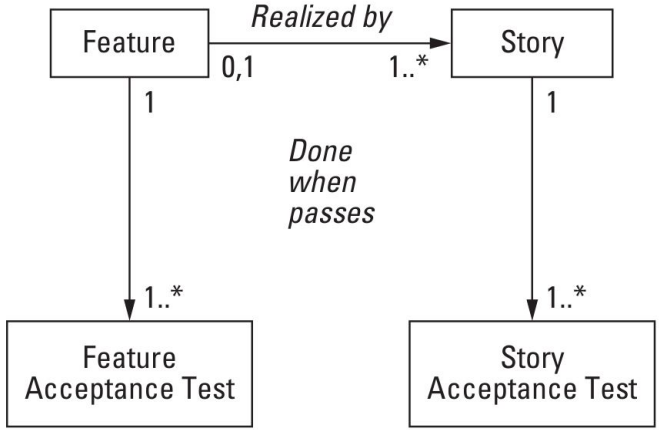
New Features Build The Program Backlog

- Features are typically expressed in bullet form or, at most, in a sentence or two.
- For example, you might describe a few features of an online email service something like this:
  - Provide “Stars” for special conversations or messages, as a visual reminder that you need to follow up on a message or conversation later.
  - Introduce “Labels” as a “folder-like” conversation-organizing metaphor.



Testing Features

- ➔ At the Program level, the question arises as to whether features also deserve (or require) acceptance tests.
- ➔ The answer is, most typically, yes.
- ➔ Although story level testing should assure that the methods and classes are reliable (unit testing) and the stories suit their intended purpose (functional testing), the feature may span multiple teams and tens to hundreds of stories. As such, it is as important to test feature functionality as it is to test story implementation.



Contents

5. Nonfunctional Requirements (NFRs)

a. Testing Nonfunctional Requirements

NonFunctional Requirements (NFRs)

- From a requirements perspective so far, we've used the feature and user story forms of expression to describe the functional requirements of the system—those system behaviors whereby some combination of inputs produces a meaningful output (result) for the user. However, we have yet to describe how to capture and express the nonfunctional requirements (NFRs) for the system.

NonFunctional Requirements (NFRs)

- ➔ Traditionally → The System qualities
- ➔ They are critical elements of system behaviour.
  - ◆ If a system isn't
    - Reliable → crashes
    - Marketable → failure to meet some imposed regulatory standard
    - Scalable → doesn't support the number of users required

It will fail just as badly as if we forgot some critical functional requirement

NonFunctional Requirements (NFRs)

➔ They **are placed in the program backlog**, but they tend to behave a little differently.

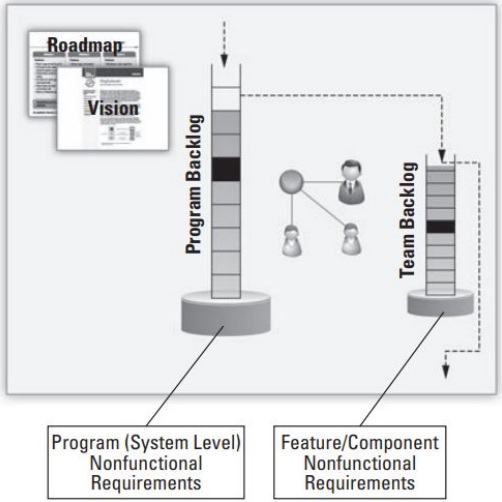


Figure 4–10 Nonfunctional requirements at the system and feature/component levels

NonFunctional Requirements (NFRs)

- Some backlog items may be constrained by nonfunctional requirements some are not.
- Some nonfunctional requirements may apply to no backlog items, meaning that they stand independently and apply to the system as a whole.

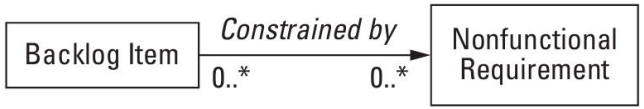
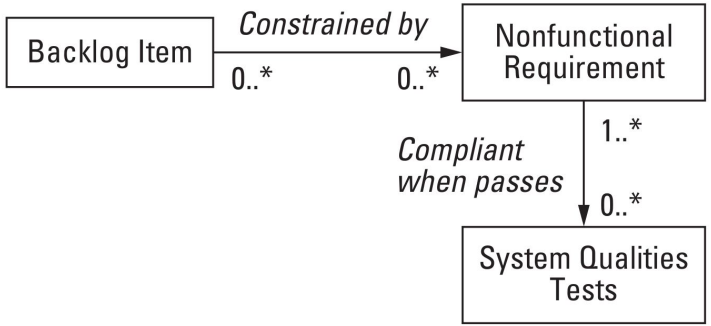


Figure 4–9 Relationship between backlog items and nonfunctional requirements

Testing NonFunctional Requirements

- Most nonfunctional (0 ...*) requirements require one or more tests.
- Rather than calling these tests another type of acceptance tests and further overloading that term, we've called them **system qualities tests**.
- This term indicates that **these tests must be run at periodic intervals** to **validate** that **the system** still **exhibits the qualities expressed by the nonfunctional requirements**.



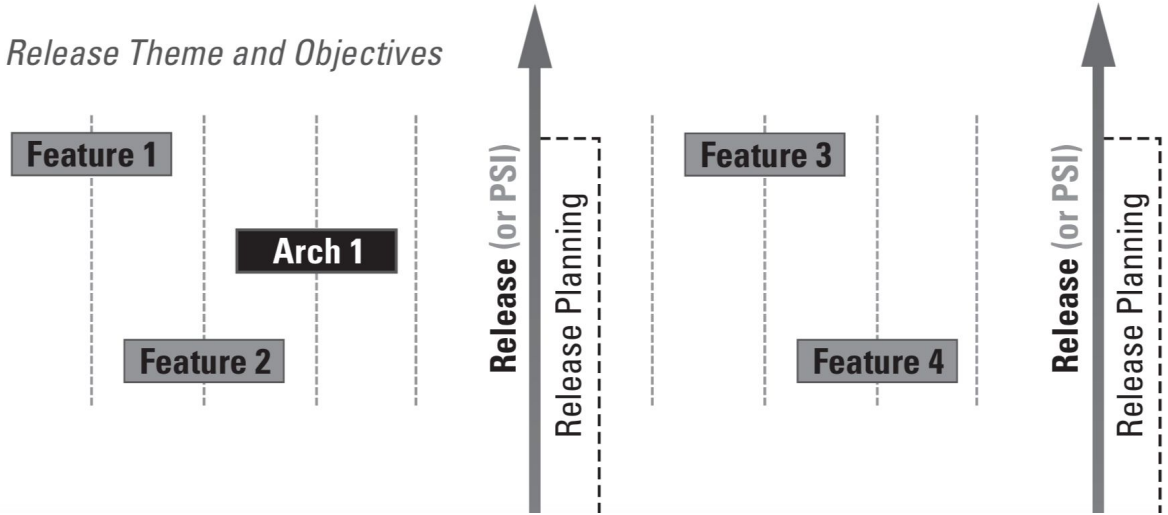
Contents

6. The Agile Release Train

The Agile Release Train

- The development of system functionality is accomplished via multiple teams in a synchronized Agile Release Train (ART).
- ART is a standard cadence of timeboxed iterations and milestones that are **date- and quality-fixed but scope variable.**
- The ART produces releases or Potentially Shippable Increments (PSIs) at frequent, typically fixed, **60- to 120-day time** boundaries.

The Agile Release Train



Release And PSIs

- The PSI is to the enterprise what iterations are to the team.
- It is a basic iterative and incremental cadence and delivery mechanism for the program.
- For many programs, release increments can be released to the customers at this chosen cadence.
- For other programs, the milestone represents achievement of a valuable and evaluable system-level increment that can then be delivered to the customer, or not, based on the business context.

Release Planning

- Release planning is **the periodic program activity** that **aligns the teams to a common mission**.
- During release planning, teams **translate the Vision into the features and stories** they will need to accomplish the objectives.

Release Planning

- ➔ It has cost and overhead!
- ➔ **BUT,** Agile principles, plus the need to assure that the teams are on a common mission, drive enterprises to engage in periodic, face-to-face release planning events.

Agile Principles

The most efficient form of communication is face-to-face.

The best requirements, architecture, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective and then tunes and adjusts its behavior accordingly.

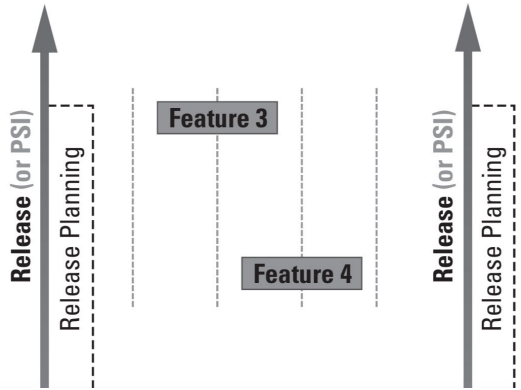
Release Planning

➔ These events gather the stakeholders to address the following objectives.

- Build and share a common Vision.
- Communicate market expectations, features, and relative priorities for the next release.
- Plan and commit to the content of the next release.
- Adjust resources to match current program priorities.
- Evolve the product Roadmap.
- Reflect and apply lessons learned from prior releases.

Release Planning

- The frequency of the event depends upon the company's required responsiveness to market conditions and the iteration and release cadence it has adopted.
- In most enterprises, it occurs **every 60 to 120 days** with **a 90-day cadence being typical.**



Contents

7. **Roadmap**

Roadmap

Vision Is Time-independent.

- ➔ The Vision describes the objectives of the product or system without any binding to time.
- ➔ This is appropriate when the objective is to communicate the overall concept of **"what this thing is we are about to build."**
- ➔ Overloading that discussion of timelines, the "when," will likely derail the discussion of the "what."

Roadmap

So, Is Time Not Necessary?

- ➔ To set priorities and plan for implementation, we need a perspective that includes time.
- ➔ This is the purpose of the Roadmap.
- ➔ The Roadmap is not a particularly complicated thing, nor is the mechanical maintenance of it difficult.

Roadmap

➔ The Roadmap consists of a series of planned release dates, each of which has a theme and a prioritized feature set. (Chapter 16)

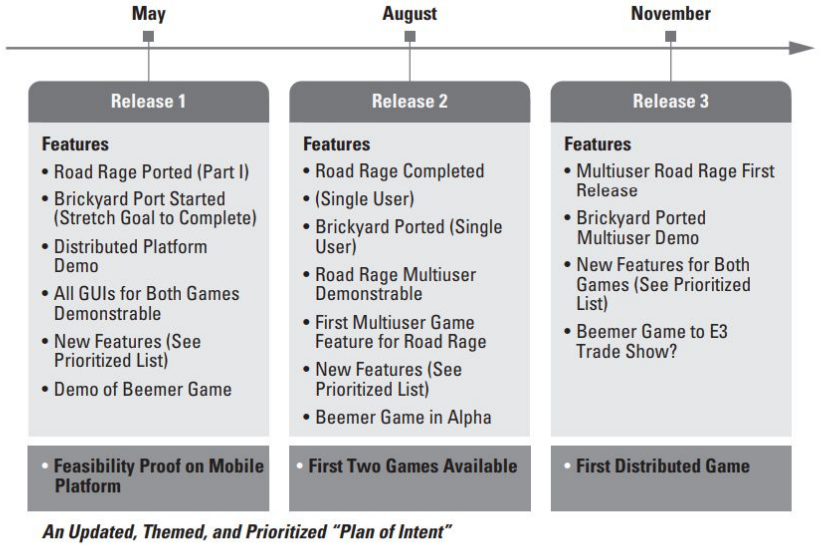


Figure 4–12 Example product roadmap for a hypothetical gaming company

Contributions

- ➔ Author of Reference Book: **Dean Leffingwell**
- ➔ Course Instructor: **Mehran Rivadeh**
- ➔ Slide Creator: **Mahnaz Rasekhi**
  - ◆ These slides are primarily based on Agile Software Requirements by Dean Leffingwell, with occasional adaptations to enhance clarity and engagement.