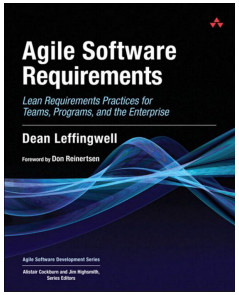
CE Department

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

The Big Picture Of Agile Requirements

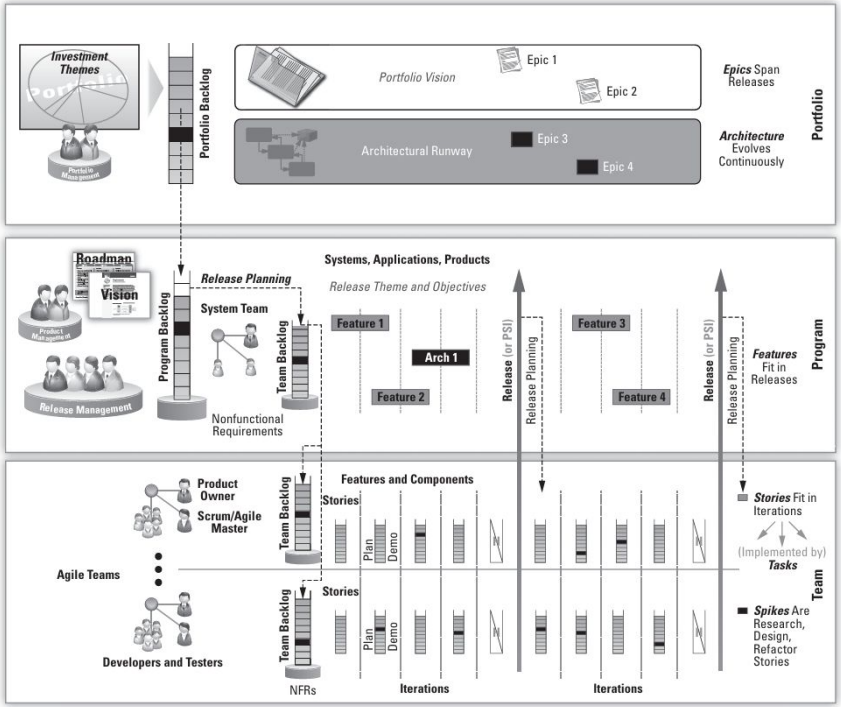
Chapter 2

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

Contents

1. **The Big Picture Explained**

The Agile Enterprise Big Picture



The requirements model in the Big Picture was developed with the help of Juha-Markus Aalto of Nokia Corporation.

Contents

2. **Big Picture: Team Level**

Team Level

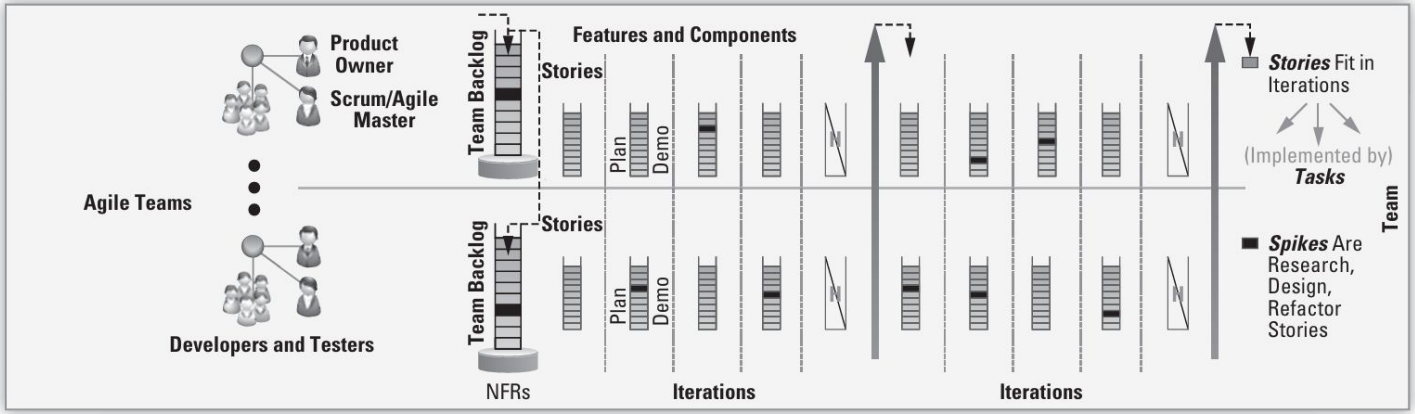
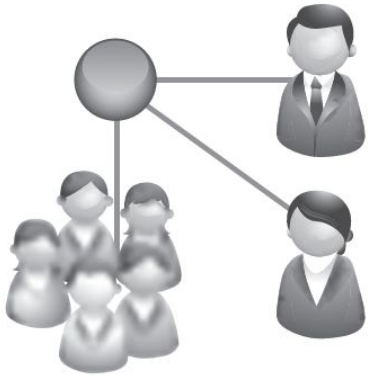


Figure 2–2 Team level of the Big Picture

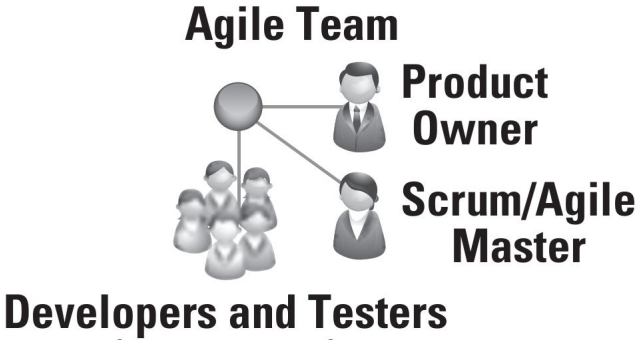
The Agile Team

- Agile teams typically consist of **7±2** members.
- They define, build, and test **user stories**.
- Work is organized into **iterations** and **releases**.
- Small enterprises may have just a few agile teams.
- Large enterprises organize teams into **pods** or groups.
- These pods collaborate to build full products and complex systems.
- **Product Owner** manages the **backlog** of User stories.



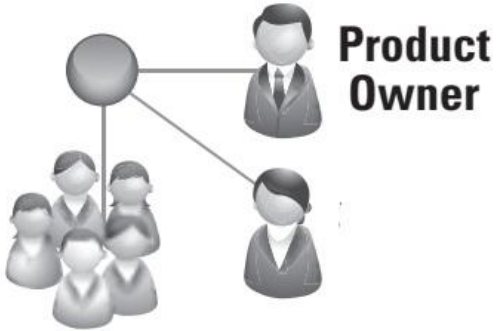
The Agile Team

- Key roles include:
  - Product Owner
  - Scrum/Agile Master
  - Developers, testers, and test automation experts
  - (Optional) Tech Lead



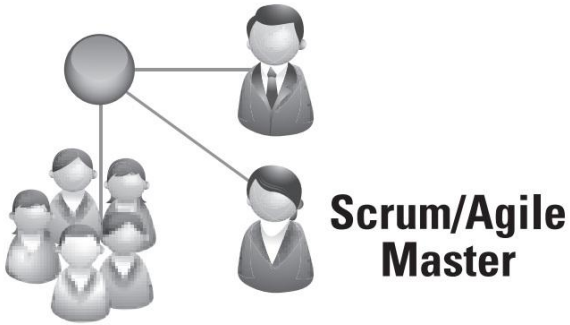
The Agile Team

- ➔ Key roles include:
  - ◆ **Product Owner**
    - Defines and prioritizes user requirements.
    - Maintains and updates the product backlog.
    - Co-located with the team and participates daily with the team and its activities.



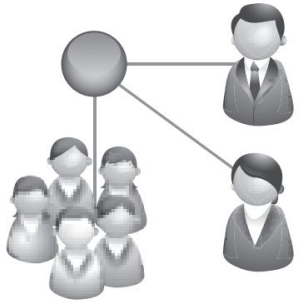
The Agile Team

- Key roles include:
  - **Scrum/Agile Master**
    - The team-based management/leadership proxy.
    - Assist the team in its transition to the new method.
    - Continuously facilitate a team dynamic intended to maximize performance of the team.



The Agile Team

- Key roles include:
  - **Developers, Testers**
    - 3–4 developers and 1–2 testers
    - Co-located and work together to define, build, test, and delive stories into the code baseline



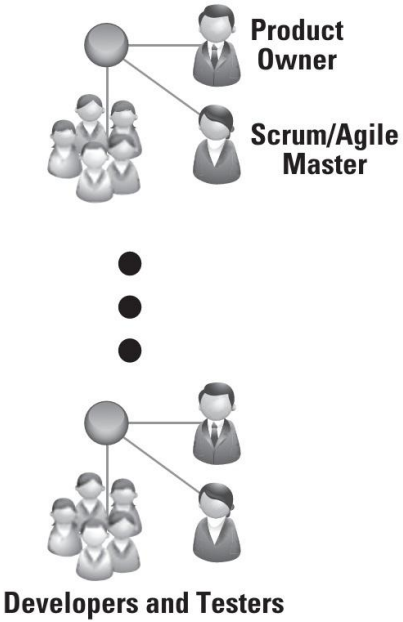
Developers and Testers
(Four to Six)

The Agile Team

- Since **testing software** is **integral** to **value delivery**, **testers** are **integral to** the **team**.
- Testers are logically **part** of the **QA** organization but are **physically assigned** and dedicated to an **agile team**.
- Their **primary allegiance** is **to** the **team**, but as members of the QA organization, they can leverage other QA teammates and managers for skills development, automation expertise, and any specialty testing capabilities that may be necessary at the system level.
- In any case, the **agile team itself** is **responsible for** the **quality** of **their work product** and that responsibility cannot be delegated to any other organization, in or out of house.

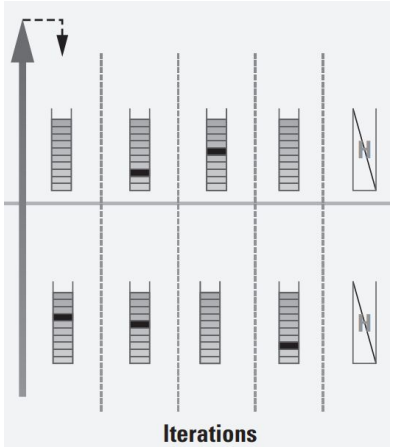
Pods of Agile Teams

- Within the larger enterprise, there are typically some number (3 to 10) or so of such teams that cooperate to build a larger feature, system, or subsystem.
- Experience has shown that even for very large systems, the logical partitions defined by system or product family architecture tend to cause “pods” of developers to be organized around the various implementation domains.
- Perhaps 50 to 100 people must intensely collaborate on building their “next bigger thing” in the hierarchy, which we’ll call a program.



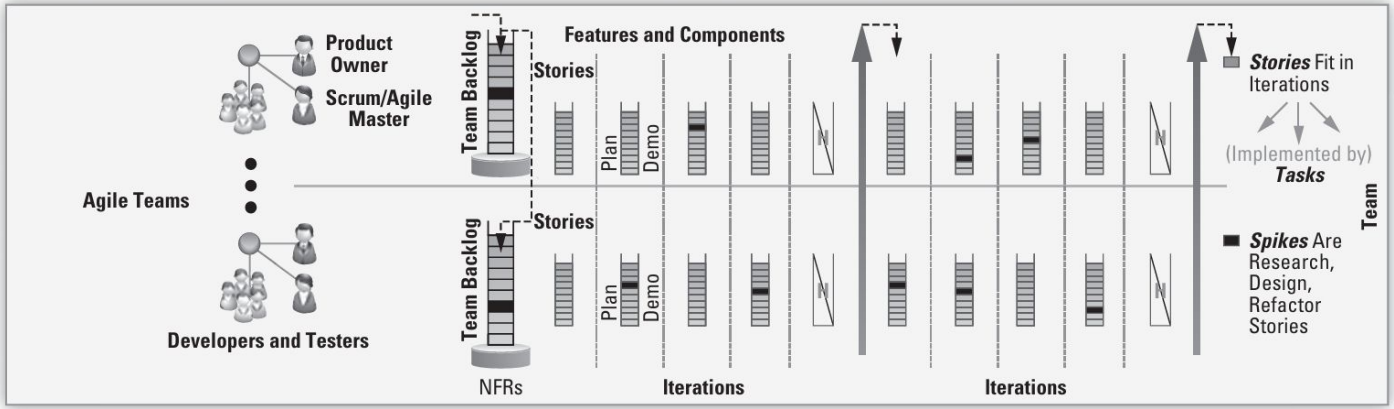
Iteration

- An iteration is a **short time boxed event** (sprint in Scrum).
- It represents a **valuable increment** of **new functionality**, accomplished via a constantly repeating **standard pattern:**
  - plan the iteration
  - build and test stories
  - demonstrate the new functionality to stakeholders
  - inspect and adapt
- Agile teams in larger enterprises typically adopt a standard iteration length.



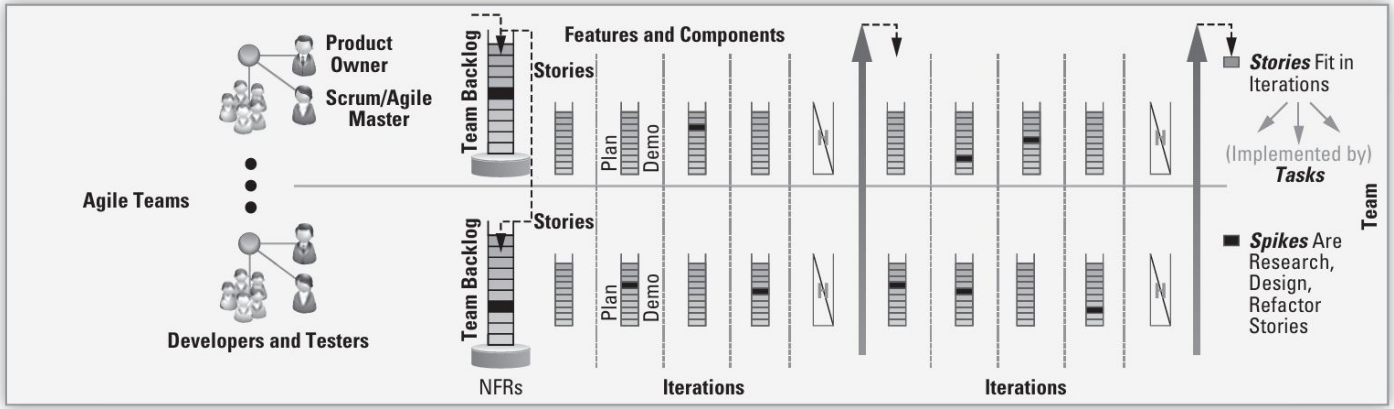
Iteration

- **All teams** follow the **same iteration length** for simplicity.
- No fixed rule, but **two-week iterations** are commonly adopted.
- A **series of iterations** is used to aggregate larger functionality for **release** to the external users.



Iteration

- **Four development iterations** followed by **one hardening iteration** prior to each release increment.
- The **length** and **number** of **iterations per release increment**, and the decision as to when to actually release an increment, are left to the **judgment of the enterprise**.



User Stories

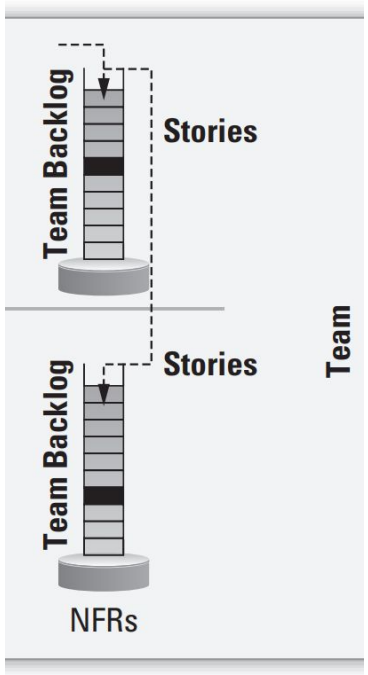
- **Requirements** represents something the system must do to fulfill a business need or contractual obligation.
- **User stories** are **brief statements** of intent that describe **something** the **system needs to do for** some **user**.

As a <user role>, I can <activity> so that <business value>.

- With this form, the team learns to focus on both the user's role and the business benefit that the new functionality provides.
- This construct is integral to agile's intense focus on value delivery.

Team Backlog

- Team backlog / project backlog / product backlog
- Each agile team maintains its own product backlog.
- Backlog includes user stories, defects, refactors, and infrastructure tasks.
- Product Owner is responsible for backlog prioritization and upkeep.
- User stories are the team's primary focus.
- Managing user stories is the core requirements process in agile.

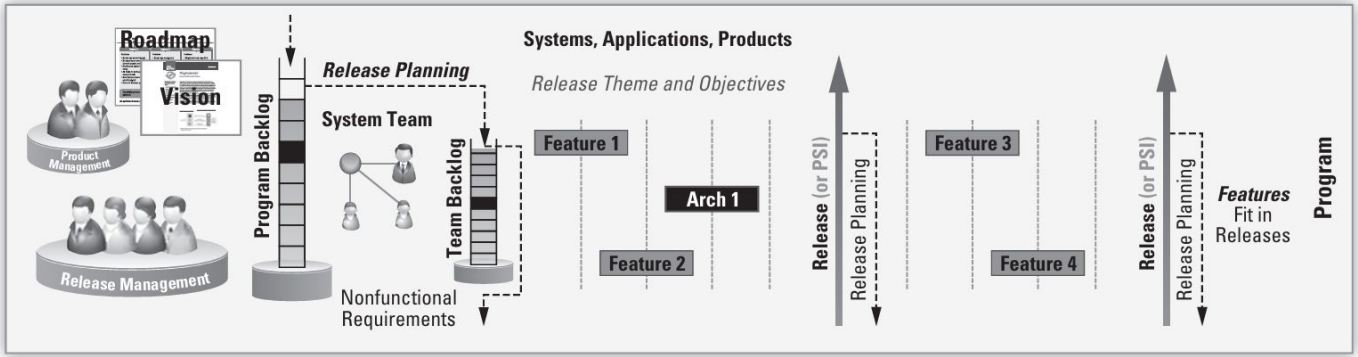


Contents

3. Big Picture: Program Level

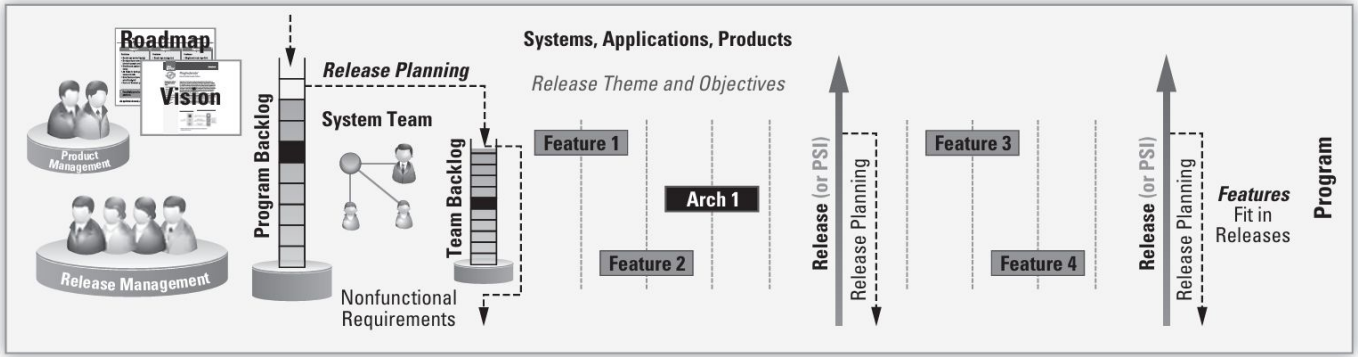
Program Level

- Program-level development involves **multiple** agile teams.
- **Teams collaborate** through a synchronized **Agile Release Train** (ART).
- ART enables delivery of **large-scale system functionality.**
- ART follows a **fixed schedule** of iterations and milestones.



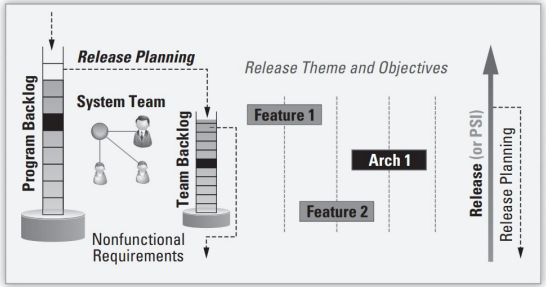
Program Level

- **Dates** and **quality** are **fixed**; **scope** remains **flexible.**
- Produces releases or PSIs every **60–120 days.**
- Release timing depends on customer readiness and external factors.
- **Product manager** is responsible for defining the **features** of the system at this level.



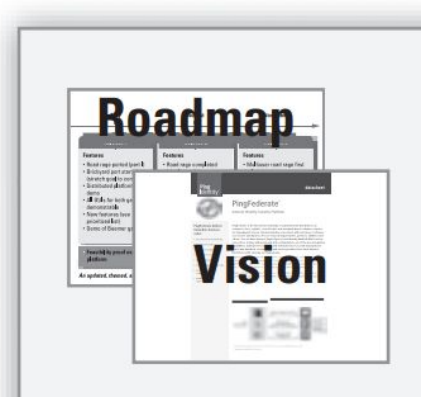
Release And PSI

- Each **iteration** aims to **produce** a **shippable software increment.**
- Shipping after every iteration may not be feasible in large enterprises.
- **Technical debt** may accumulate and must be addressed before release e.g.:
  - Defects
  - Code refactoring
  - System-wide testing
  - Finalizing documentation
- Hardening iterations (empty backlog) allow time for these activities.



Vision

- Answers the big questions for the system, application, or product, including:
  - What **problem** does this particular solution solve?
  - What **features** and **benefits** does it provide?
  - **For whom** does it provide it?
  - What **performance**, **reliability**, and so on, does it deliver?
  - What **platforms**, **standards**, **applications**, and so on, will it support?

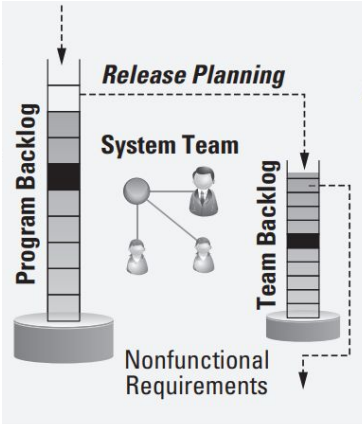


Vision

- The primary content of the vision is **a prioritized set of features** intended to deliver benefits to the users.
- Various **nonfunctional requirements** that are necessary for the system to meet its objectives:
  - Reliability
  - Accuracy
  - Performance
  - Quality
  - Compatibility standards, and so on
- A Vision may be **maintained in** a **document**, a **backlog repository**, or even in a simple briefing or **presentation form**.

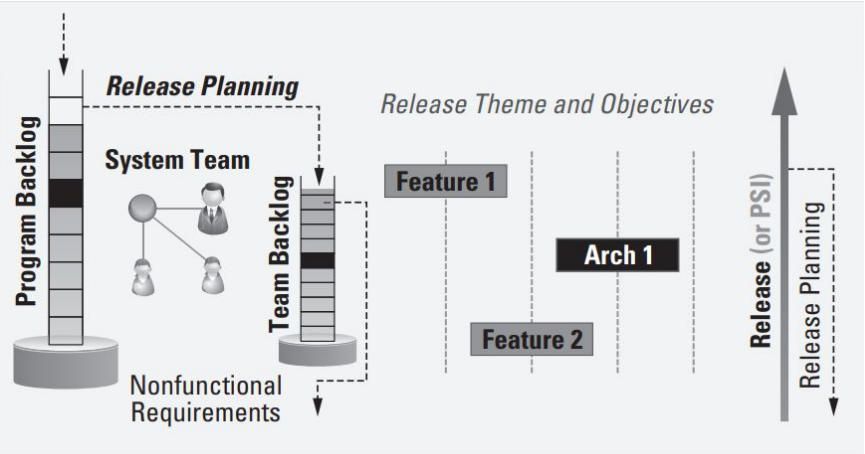
Program Backlog

- The program / release backlog contains the **set of desired** and **prioritized features** that have **not** yet been **implemented**.
- The program backlog **may or may not** also **contain estimates for** the **features**.



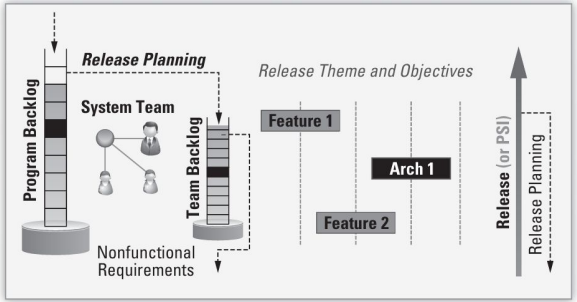
Release Planning

- Each release increment begins with a kickoff **planning session**.
- **Aligns teams** to company context and business objectives.
- Inputs include:
  - Current Vision
  - Objectives
  - Prioritized feature set



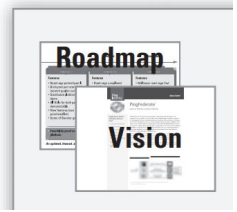
Release Planning

- **Teams** break **features** into **stories** and **plan** using **iteration** cadence and **velocity**.
- **Interdependencies** are **resolved** and **stories mapped** to **PSI timebox.**
- **Scope trade-offs** are **negotiated** with **product management.**
- Outcome: **commitment** to **release objectives** and **prioritized features.**
- Teams strive to meet objectives, even if some features slip past the deadline.

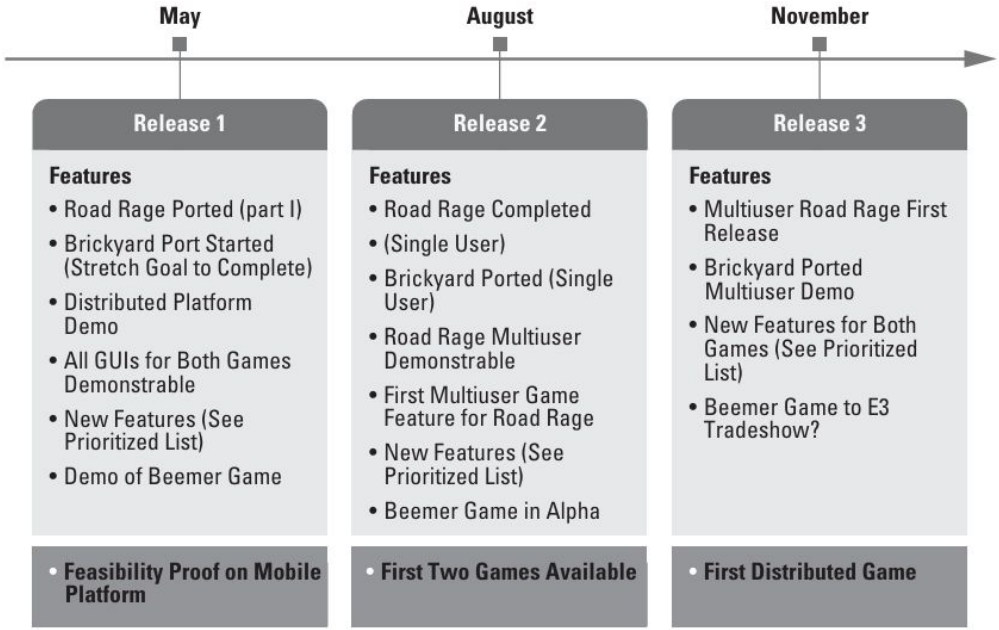


Roadmap

- The **results** of **release planning** are used to update the **product Roadmap**.
- Reflects the enterprise's intent to **deliver value over time**.
- Each release has a **theme**, **objectives**, and **prioritized features**.
- The "next" release on the Roadmap is committed to the enterprise, based on the work done in the most recent release planning session.
- Releases beyond the next one are not committed, and their scope is fuzzy at best.
- It represents the enterprise's **current** "**plan of intent**" for the **next** and **future releases**.
- **Plans** may **change** based on development, business, or customer needs.



Roadmap


May
August
November
Release 1
Features
• Road Rage Ported (part I)
• Brickyard Port Started (Stretch Goal to Complete)
• Distributed Platform Demo
• All GUIs for Both Games Demonstrable
• New Features (See Prioritized List)
• Demo of Beemer Game
• Feasibility Proof on Mobile Platform
Release 2
Features
• Road Rage Completed
• (Single User)
• Brickyard Ported (Single User)
• Road Rage Multiuser Demonstrable
• First Multiuser Game Feature for Road Rage
• New Features (See Prioritized List)
• Beemer Game in Alpha
• First Two Games Available
Release 3
Features
• Multiuser Road Rage First Release
• Brickyard Ported Multiuser Demo
• New Features for Both Games (See Prioritized List)
• Beemer Game to E3 Tradeshow?
• First Distributed Game

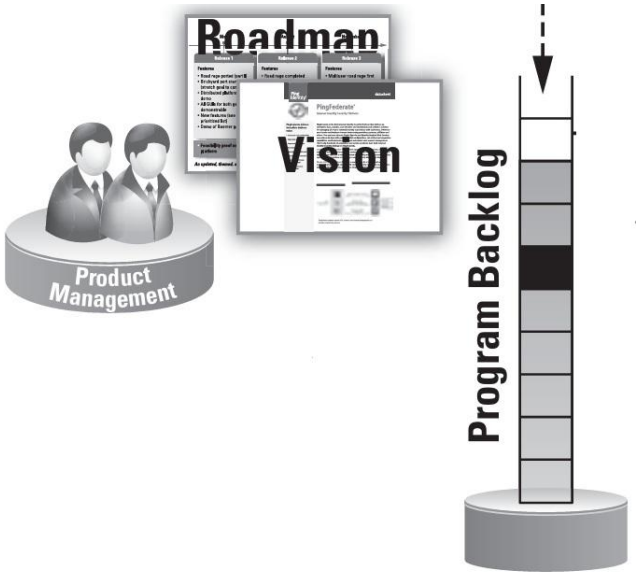
Product Management

- In the **smaller** software enterprise, **one or two product owners** are all that are needed to **define** and **prioritize software requirements.**
- In the **larger** software enterprise, **the set of responsibilities** imbued in the product owner is more typically a **much broader** set of responsibilities shared between team and technology-based product owners and market or program-based product managers, who carry out their traditional responsibilities of both defining the product and presenting the solution to the marketplace.

Product Management

Responsibilities Of The Agile Product Manager In The Enterprise:

- ➔ Own the Vision and program (release) backlog
- ➔ Manage release content
- ➔ Maintain the product Roadmap
- ➔ Build an effective product manager/product owner team



Contents

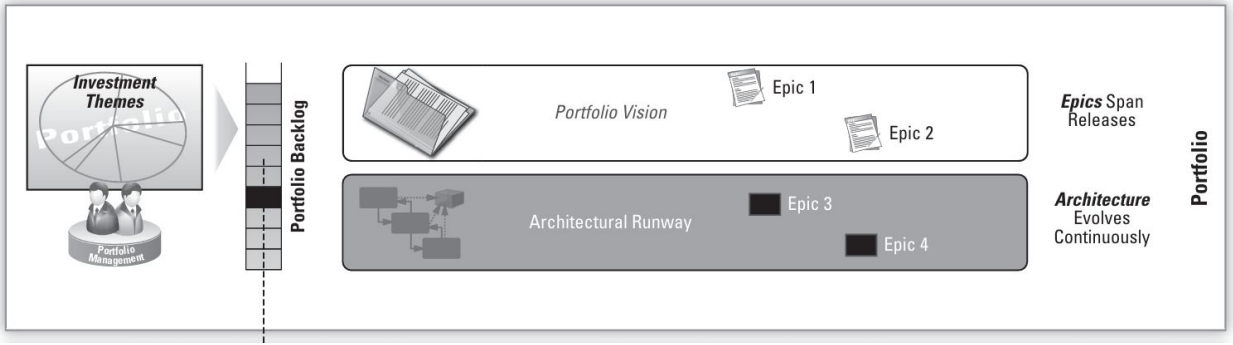
4. Big Picture: Portfolio Level

a. Investment Theme

b. Epics

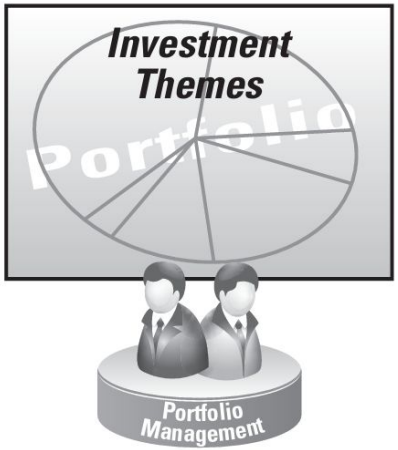
Portfolio Level

- The portfolio management manage the **investments** of the **enterprise** in accordance with the enterprise business strategy.
- New artifacts:
  - Investment Theme
  - Epics → Portfolio Vision



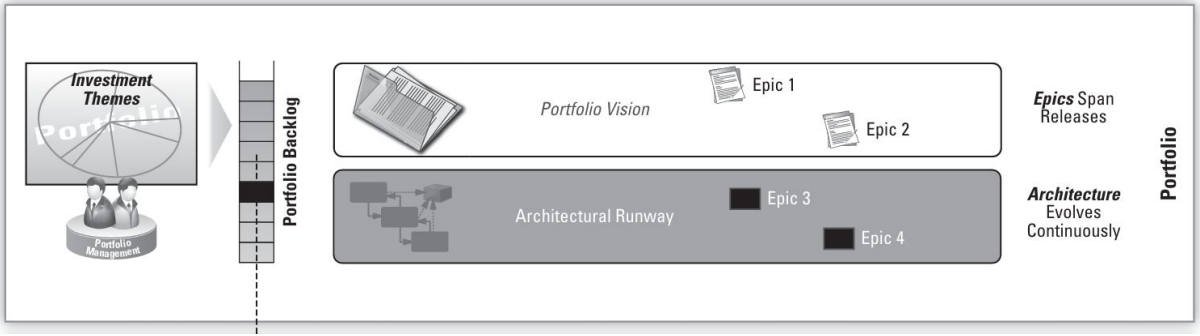
Investment Theme

- The **relative investment objectives** for the enterprise or business unit.
- Drive the **vision for all programs**, and new epics.
- Portfolio managers are responsible for these decisions:
  - line-of-business owners, product councils
- The result of the decision process is **a set of themes**
  - key product value propositions that provide marketplace differentiation and competitive advantage
- Themes have a much longer life span than epics, and **a set of themes** may be **largely unchanged** for **up to a year or more.**



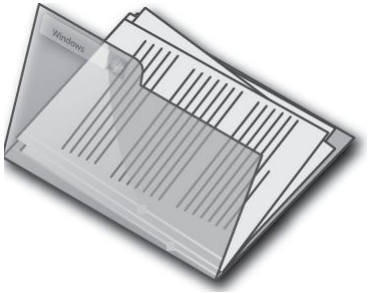
Epics

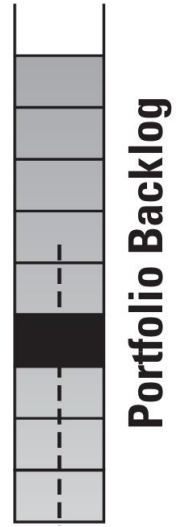
- Represent **the highest-level expression of a customer need.**
- Development initiatives that are intended to deliver the value of an investment theme.
- Identified, prioritized, estimated, and maintained in **the portfolio backlog**.
- Prior to release planning:
  - Epics → specific features → more detailed stories for implementation



Epics

- An Epic need only be described in detail sufficient to initiate a further discussion about what types of features the epic implies.
  - In bullet form
  - In uservoice story form
  - As a sentence or two
  - In video
  - In a prototype
  - In any form of expression





End of Chapter 2

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.