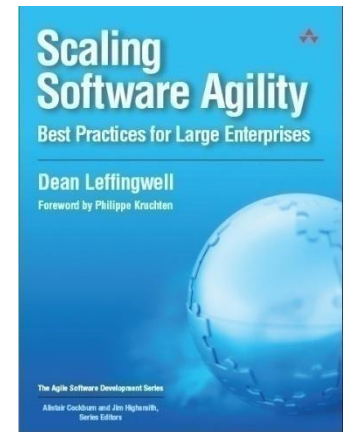


# Harnessing Innovation

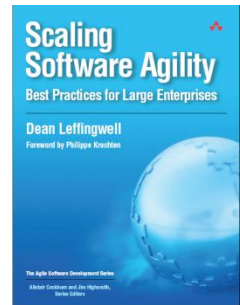
---

## Lightweight Governance Models for High Performing Agile Teams

May 22, 2008  
Dean Leffingwell



# Approaching Challenge at Scale



***“We place the highest value on actual implementation and taking action.***

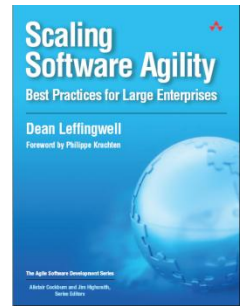
***There are many things one doesn’t understand; therefore, we ask them, **why don’t you just go ahead and take action?*****

***You realize how little you know, and you face your own failures and redo it again, and at the second trial you realize another mistake . . . **So you can redo it once again.*****

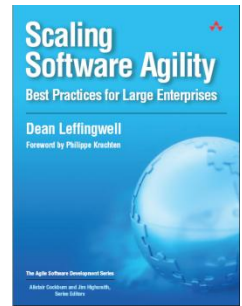
***So by constant improvement one can rise to the higher level of practice and knowledge.***

***This guidance reminds us that there is no problem too large to be solved if we are only willing to take the first step.”***

Fujiio Sho, President, Toyota



# READY, AIM AND ....

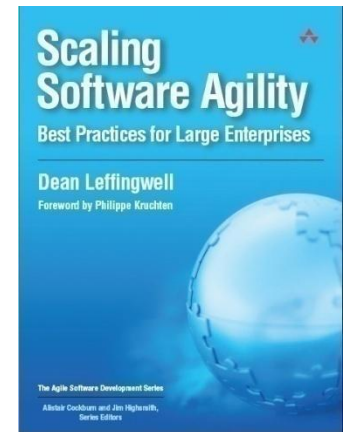


**READY**

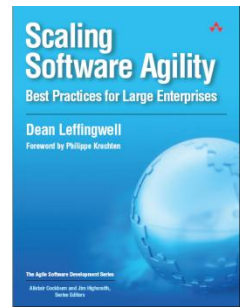
Are we getting the  
productivity, quality and  
morale that we all deserve?

# Seven Agile Team Practices That Scale

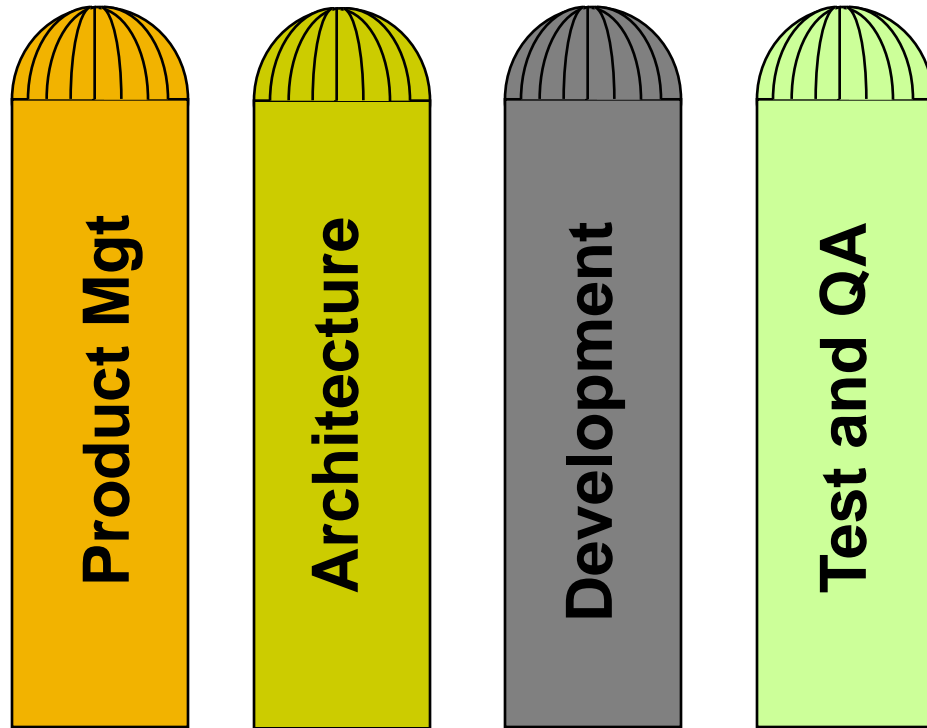
The Define/Build/Test Component Team  
Mastering the Iteration  
Two-level Planning and Tracking  
Smaller, More frequent releases  
Concurrent Testing  
Continuous Integration  
Regular Reflection and Adaptation



# 1. Define/Build/Test Component Team



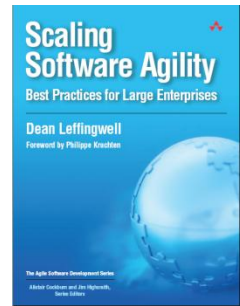
## Before Agile: Typical Functional Silos



- Optimized for vertical communication
- Friction across the silos
- Location via function
- Political boundaries between functions

**Management Challenge: Connect the Silos**

# Conway's Law

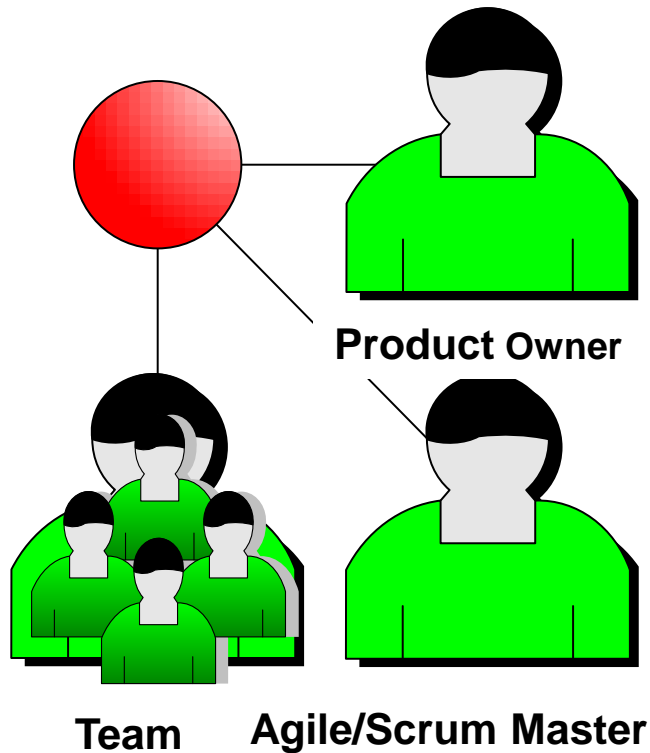
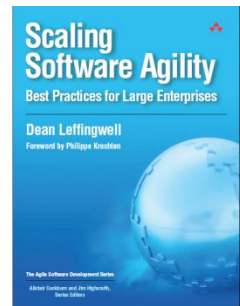


*“Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.”*

- Mel Conway (1968)

*(rigid organizations that are not willing to re-organize to generate an optimal design, can end up producing a sub-standard design that merely reflects the pre-existing organization.)*

# Define/Build/Test Team



## Product Owner

- Assure team is pursuing a common vision
- Establish priorities to track business value
- Act as 'the customer' for developer questions
- Work with product management to plan releases
- Accept user stories and iteration

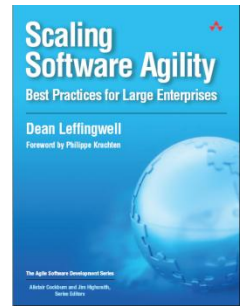
## Scrum Master

- Run team meetings, enforce scrum
- Remove impediments
- Attend integration scrum meetings
- Protect the team from outside influence

## Team

- Create user stories from product backlog
- Commit to iteration plan
- Define/Build/Test/Deliver stories (fully accepted)

## 2. Mastering the Iteration

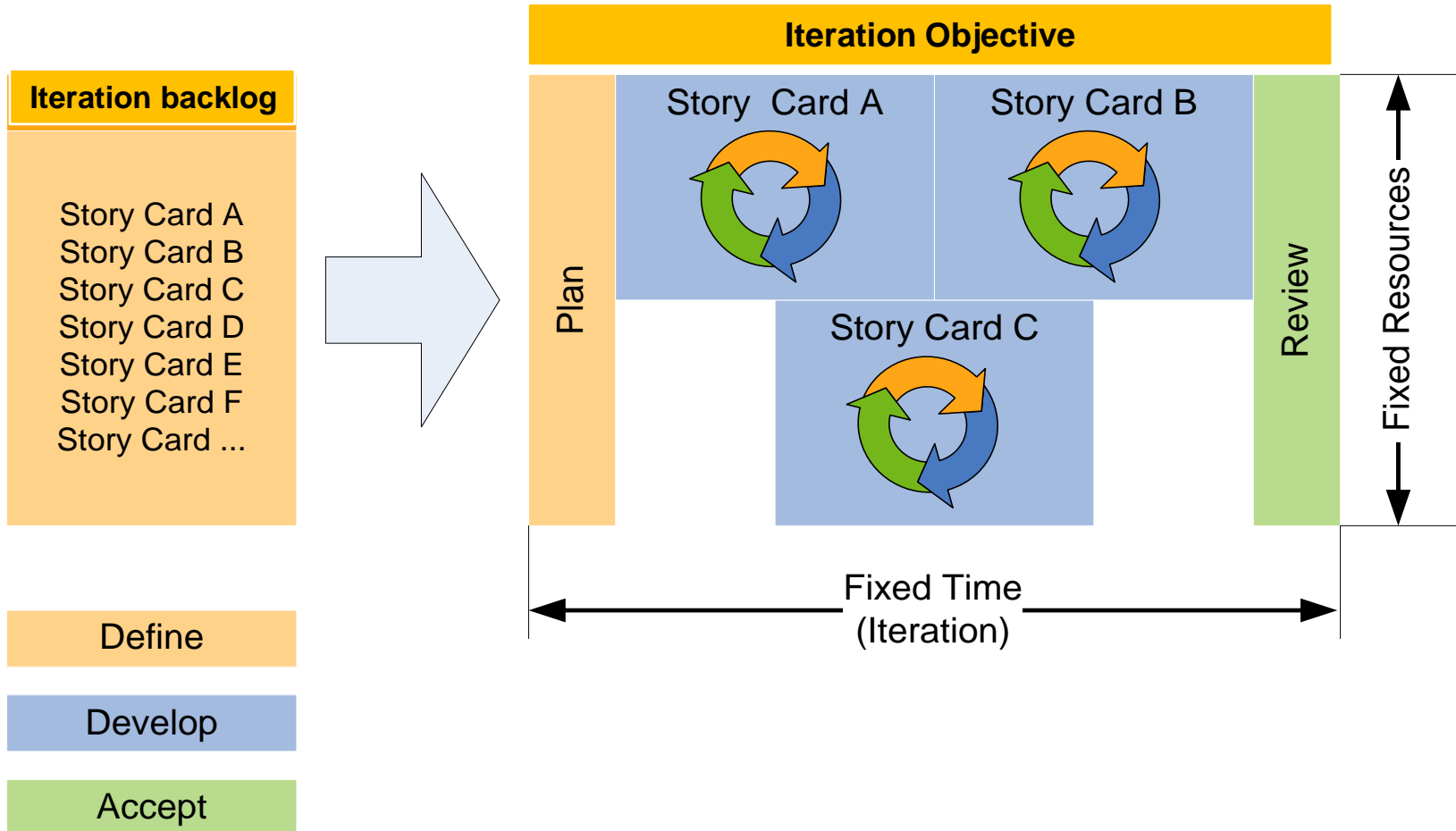
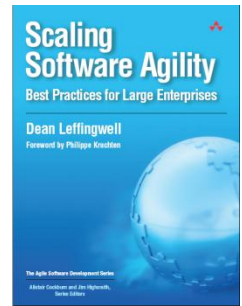


*The iteration is the heartbeat of agility.*

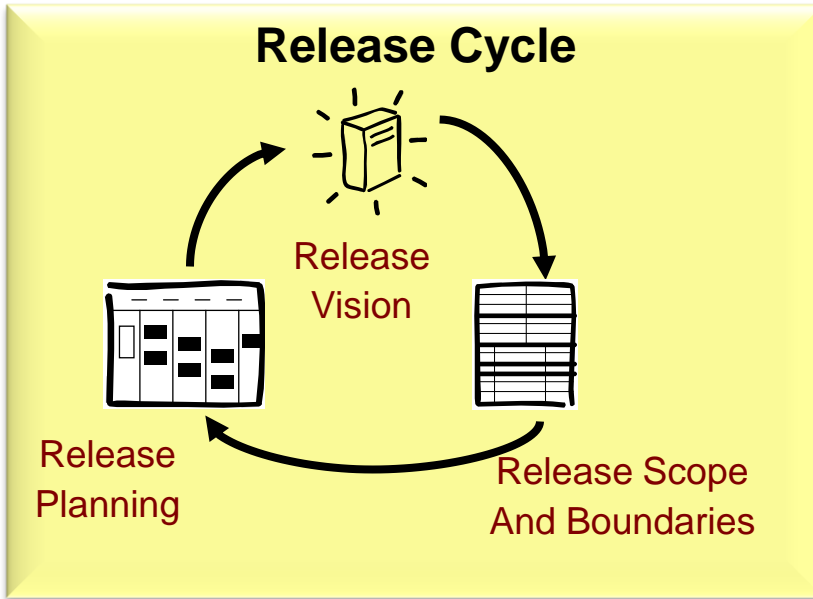
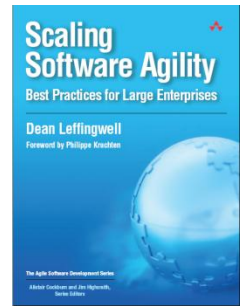
*Each iteration is a “potentially shippable increment” of software.*

*Master that, and most other things agile tend to fall naturally into place.*

# Iteration Pattern



# 3. Two-Level Planning and Tracking

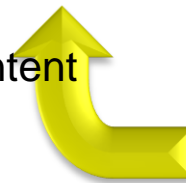


**Drives** Plan iterations at the **component** level

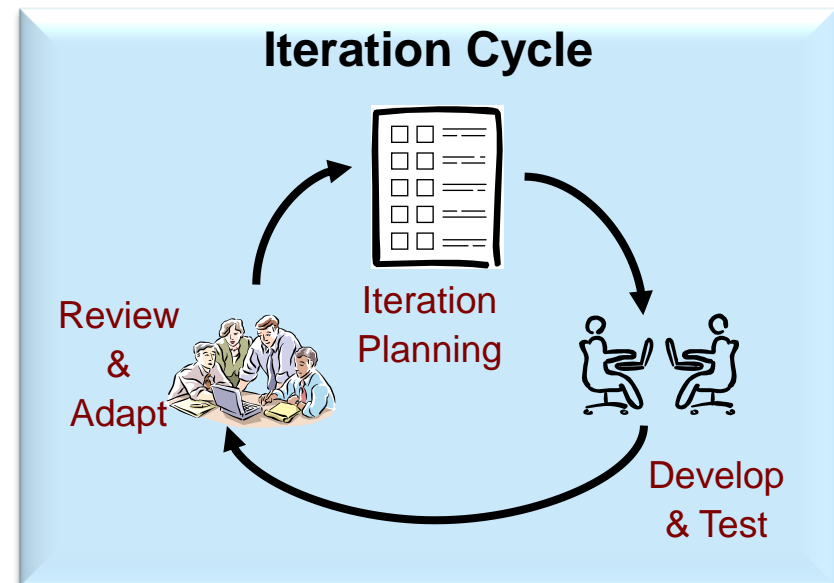
- 2-4 iteration visibility
- Currency: user stories

Plan Releases at the *System* Level

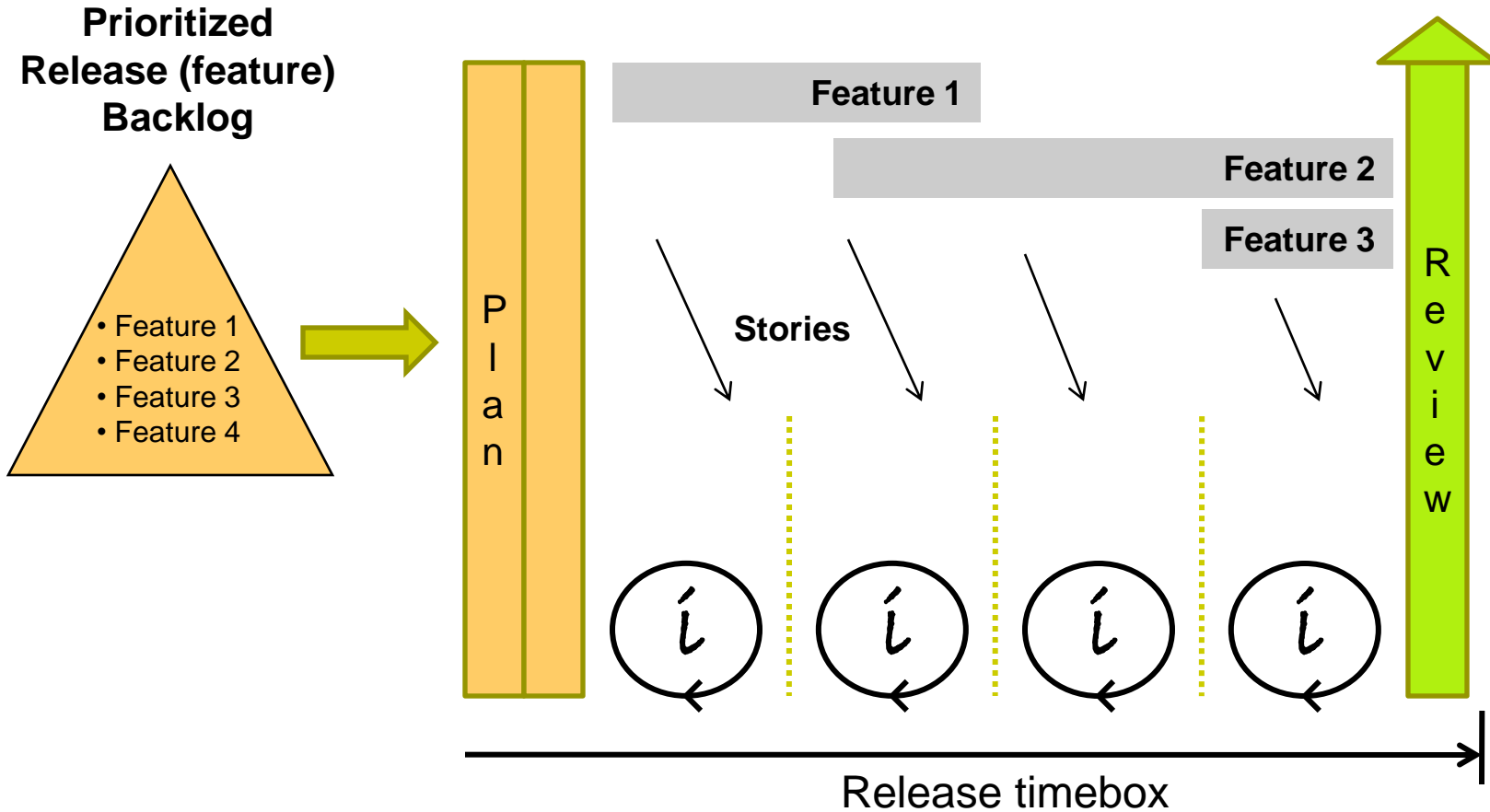
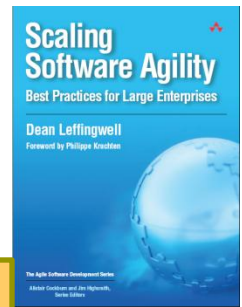
- Three to six months horizon
- Prioritized feature sets define content



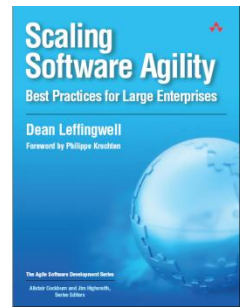
**Feedback - Adjust**



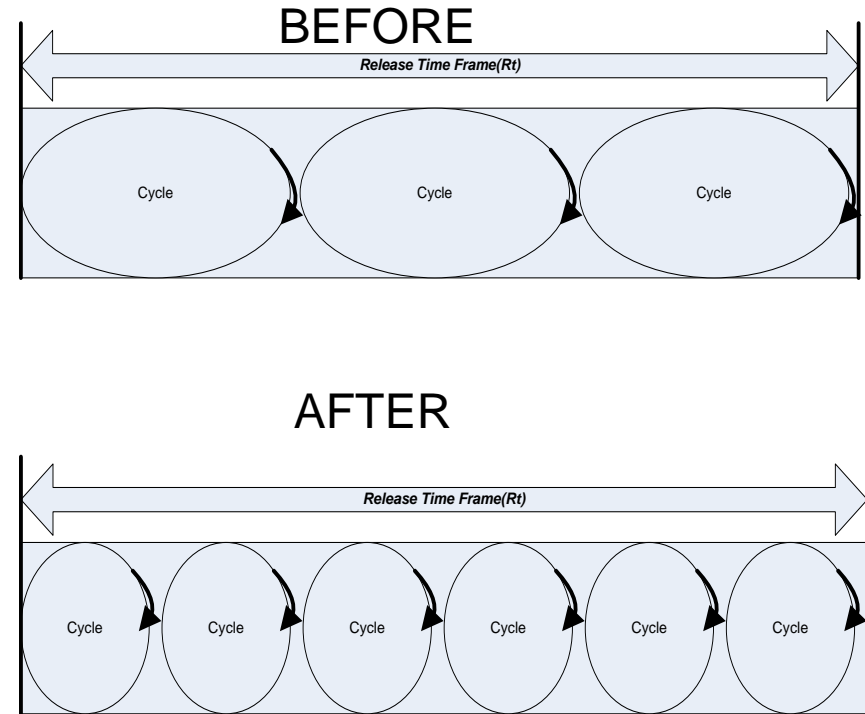
# Release Pattern



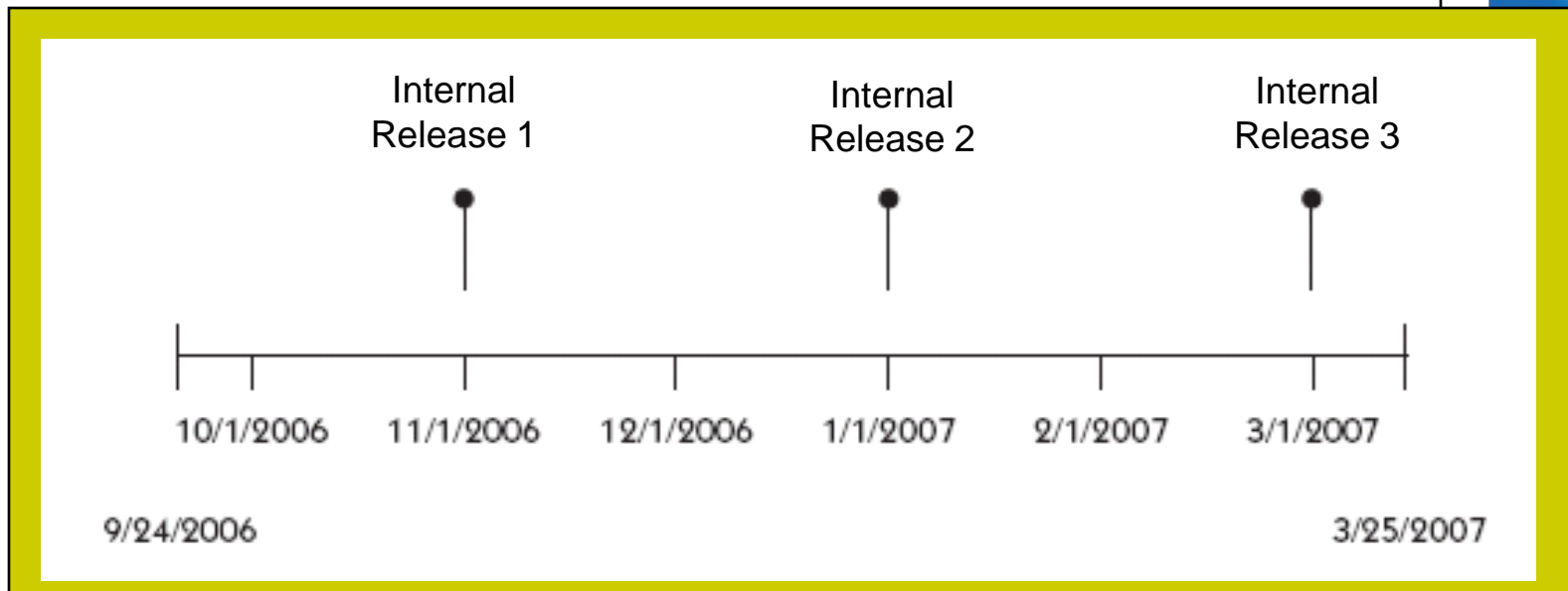
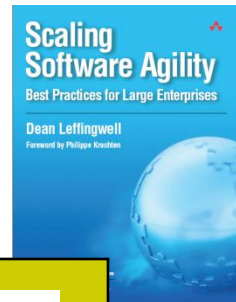
# 4. Smaller, More Frequent Releases



- Shorter release dates
  - 60-120 days
- Releases defined by
  - Date, theme, planned feature set, quality
- Scope is the variable
  - Release date and quality are fixed



# Fix the Dates - Float the Features

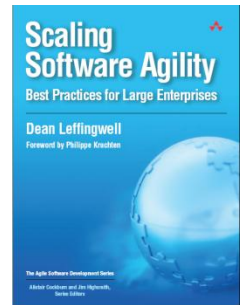


- Teams learn that dates **MATTER**
- Product owners learn that priorities **MATTER**
  - Agile teams **MEET** their commitments

# 5. Concurrent Testing

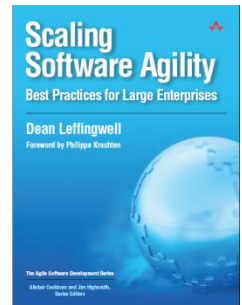
## Philosophy of Agile Testing

- All code is tested code. Teams get no credit for delivering functionality that has been coded, but not tested.
- Tests are written before, or concurrently with, the code itself.
- Testing is a team effort. Testers and developers all write tests.
- Test automation is the rule, not the exception.

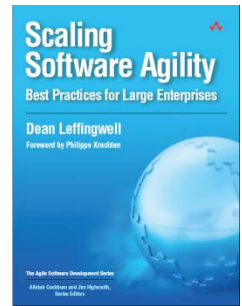


# Concurrent

- Unit Testing
  - Developer written
- Acceptance Testing
  - Customer, product owner, tester written
- Component Testing
  - Integrated BVT (build verification tests) at component/module level
- System, Performance and Reliability Testing
  - Systems tester and developer Written
  - QA Involvement



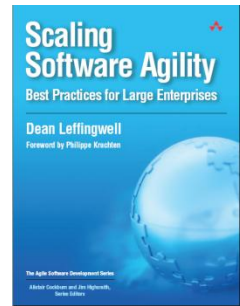
# On Test Automation



# Automate Now.

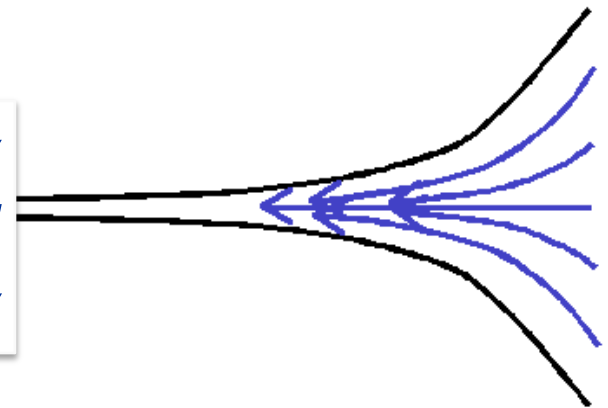
- You have no choice
- Manual tests bottleneck velocity
- You can't ship what you can't test

# 6. Continuous Integration



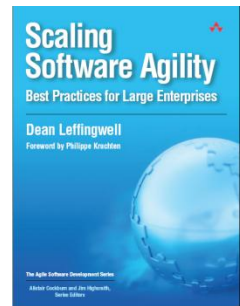
- Continuous integration is neither new nor invented by agile
- It has been applied as a best practice for at least a decade
- However, continuous integration is mandatory with agile

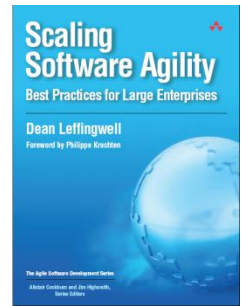
*the teams ability to build continuously is a critical bottleneck to delivered velocity*



# Continuous Integration Success

- Team can build at least once a day
  - Effort is inversely proportional to time between builds!
  - A broken build “stops” production and is addressed immediately
- Successful builds
  - Checks in all the latest source code
  - Recompile every file from scratch
  - Successfully execute all unit tests
  - Link and deploy for execution
  - Successfully execute automated Build Verification Test



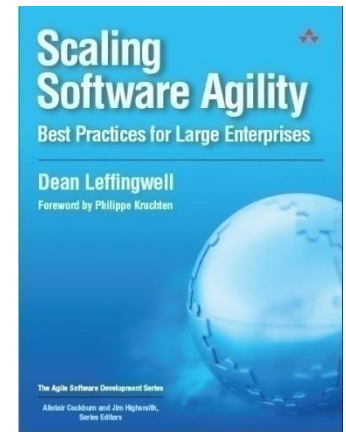


**AIM**

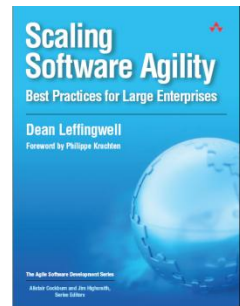
It is managements  
responsibility to steer the  
ship.

# AIM

Vision and Lean Requirements  
Intentional Architecture  
Collaborative, synchronized, multi-  
level Release Planning



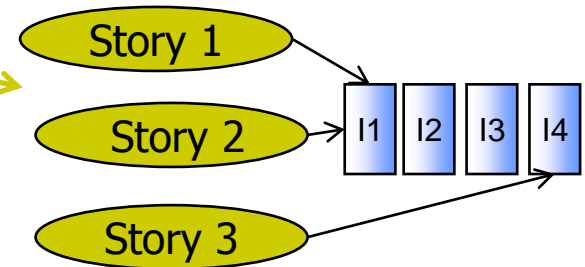
# Lean Requirements at Scale



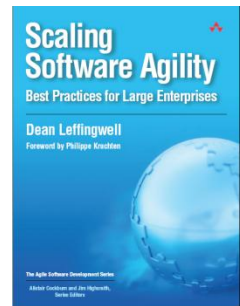
## Vision+



## Just-in-Time Elaboration



# Vision – Management 's Responsibility



- Where are we headed?
- What problem does it solve?
- What **features** and benefits does it provide?
- For whom does it provide it
- What performance does it deliver?
- What platforms, standards, applications, etc will it support?

**PingIdentity CORPORATION** FEDERATED IDENTITY. SIMPLIFIED. Dean Leffingwell Signed In | My Account - Sign Out

Home » Products & Services » PingFederate

### PingFederate

advanced federation server

Enterprises require a flexible and cost effective way to integrate, manage, and secure disparate applications and users across internal and external domains. Doing so is simplified with PingFederate's best-of-breed, light-weight approach to standards-based identity federation.

**Introducing PingFederate v2.0** [Data Sheet](#) | [Download Now](#)

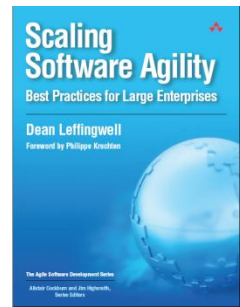
PingFederate is a best-of-breed identity federation server which implements the SAML 1.1 protocol to provide authentication, attribute and authorization portability across autonomous security domains. Use PingFederate to enable standards based single sign-on and attribute exchange across domains. PingFederate is unique in its focus on simple administration, integrated security services, flexible "pay as you succeed" pricing, and enabling innovative cross-domain provisioning and policy management capabilities. Leveraging PingFederate within your identity federation initiatives will reduce complexity, cost, and time-to-production for both you and your partners.

**How It Works**

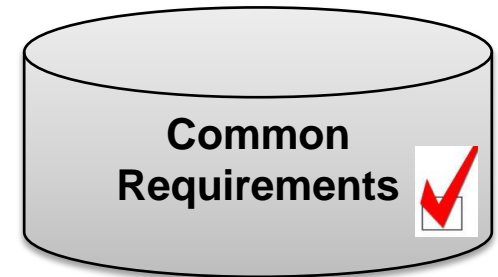
Contact Sales  
888-468-2882  
sales@pingidentity.com

Ping Newsletter  
[email address]

# Vision+ Records Common Requirements

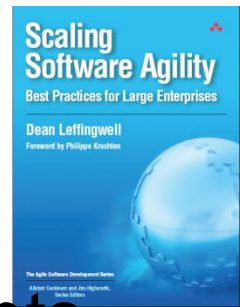


- Some requirements *must* be known by all teams
  - Performance, reliability and security requirements
  - Industry/Regulatory/Customer standards and imposed specifications
  - Internationalization, accessibility
  - Corporate standards: copyright, logo, graphics, legal

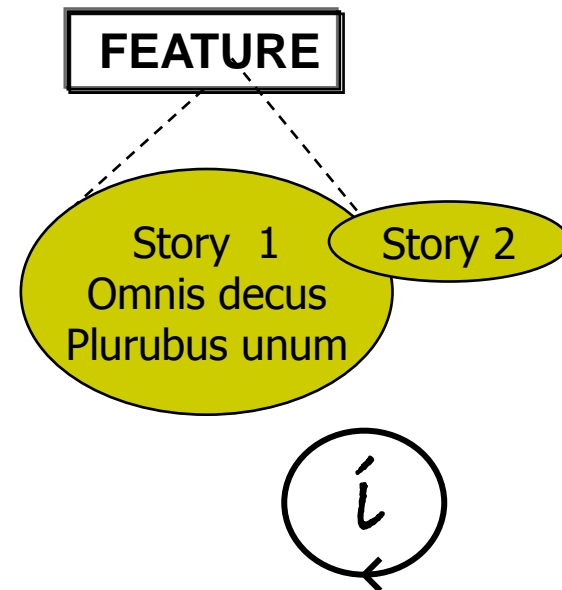


***These must ALL be documented online and be continuously available to all affected component teams.***

# Just-In-Time Elaboration – Component Team’s Responsibility



- Agile investment in documenting requirements is minimal prior to implementation
  - Features are high level, abstract
  - Communicate only concept
  - Little “work in process”
- At iteration boundaries, elaboration is required
  - Refine the team’s understanding
  - Support design, implementation and testing
  - Define acceptance criteria
- User Stories are the currency

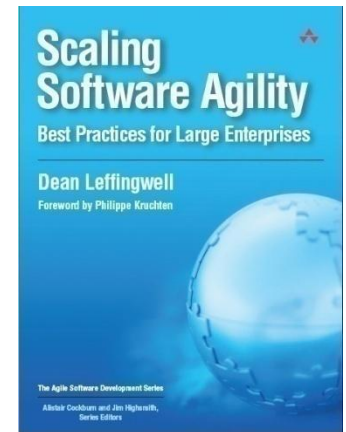


# AIM

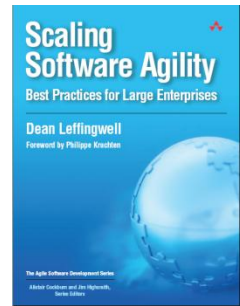
Vision and Lean Requirements

**Intentional Architecture**

Collaborative, multi-level Release  
Planning



# Intentional Architecture

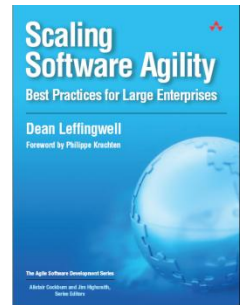


Continuous refactoring of large-scale, system-level architectures is problematic:

- Substantive rework for large numbers of teams
  - Some of whom would otherwise NOT have to refactor their component or module
- Potential Impact on deployed systems/ users
  - Best possible BVT (Build Verification Tests) are imperfect
- Common architectural constructs ease usability, extensibility, performance and maintenance

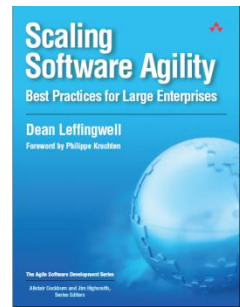
**For systems of scale, some “*intentional architecture*” is necessary**

# Principles of Agile Architecture

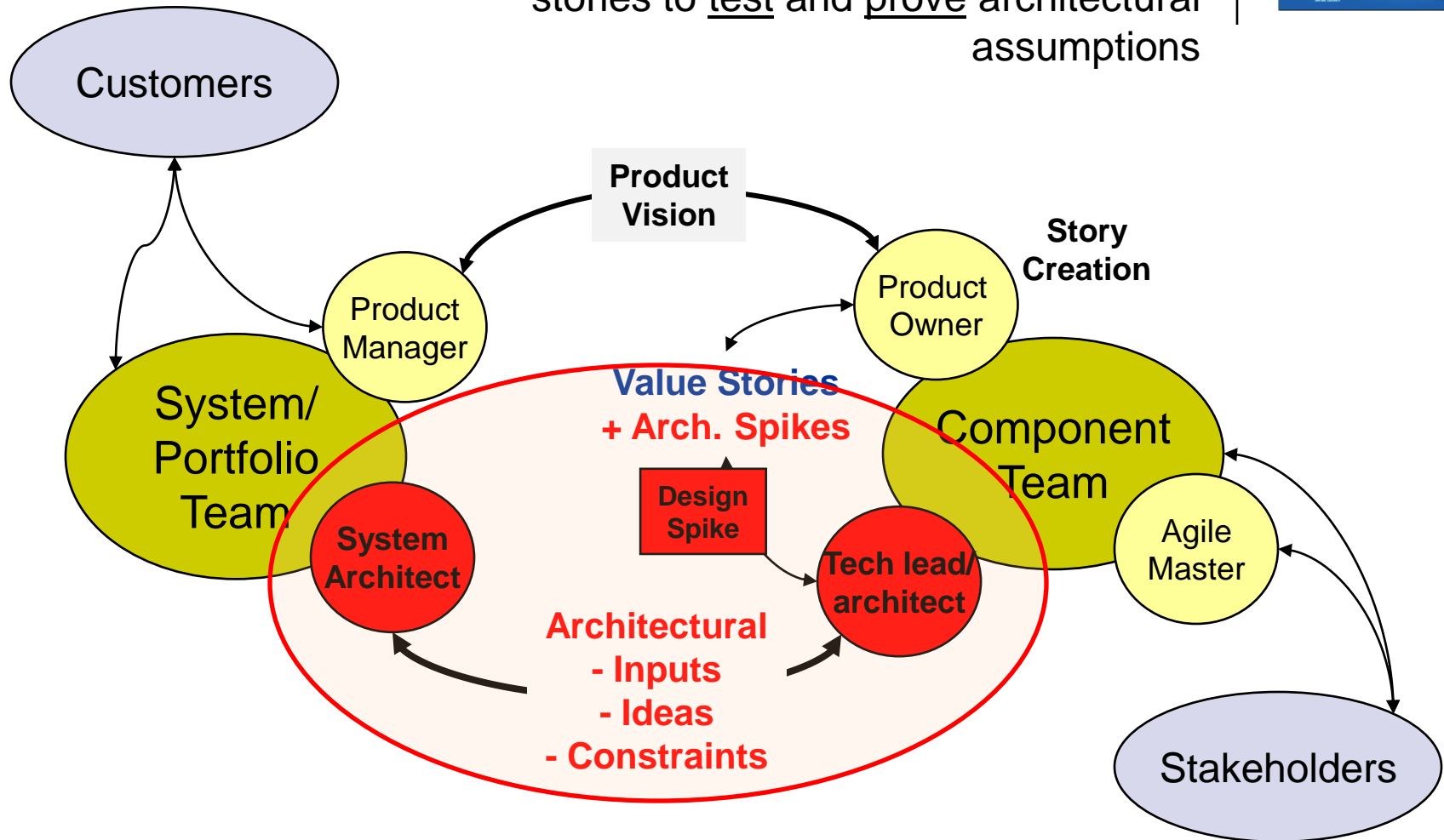


- Principle # 1 The teams that code the system design the system.
- Principle # 2 Build the simplest architecture that can possibly work.
- Principle # 3 When in doubt, code it out.
- Principle # 4 They build it, they test it.
- Principle # 5 The bigger the system, the longer the runway.
- Principle # 6 System architecture is a role collaboration.
- Principle # 7 There is no monopoly on innovation

# System Architecture is a role collaboration

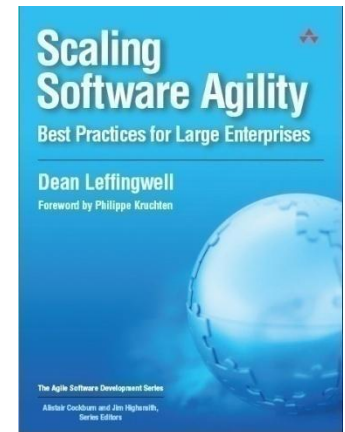


Architects and tech leads collaborate on stories to test and prove architectural assumptions



# AIM

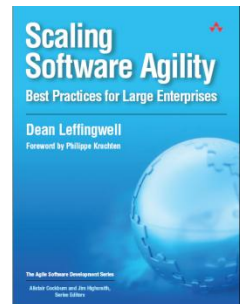
Vision and Lean Requirements  
Intentional Architecture  
**Collaborative, multi-level  
Release Planning**



## You Might Need Better Release Planning if.....

Posted on February 8, 2008 by ssaieffing | [Edit](#)

- **IF** your agile team is just getting into the flow in a nascent agile enterprise, and if your teams have been head down so long meeting near term iteration objectives that they are starting to ask about the bigger picture - **then there is something wrong with your release planning process (inadequate team vision)**
- **IF** your company organizes special projects to emphasize new initiatives and project managers spend time meeting about them - **then there is something wrong with your release planning process (new epics not factored into release commitments)**
- **IF** late discovery of interdependencies amongst teams prevent increased velocities of system- level deliveries - **then there is something wrong with your release planning process (interdependencies not considered in release commitments)**
- **IF** your team invests too little time in longer term architectural initiatives and big refactors - **then there is something wrong with your release planning process (architectural runway given insufficient consideration in release commitments)**



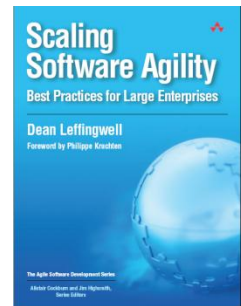
# Multi-level Release Planning

- Agile maturity requires planning cycles longer than a sprint
- Planning requires managing complex interdependencies amongst teams

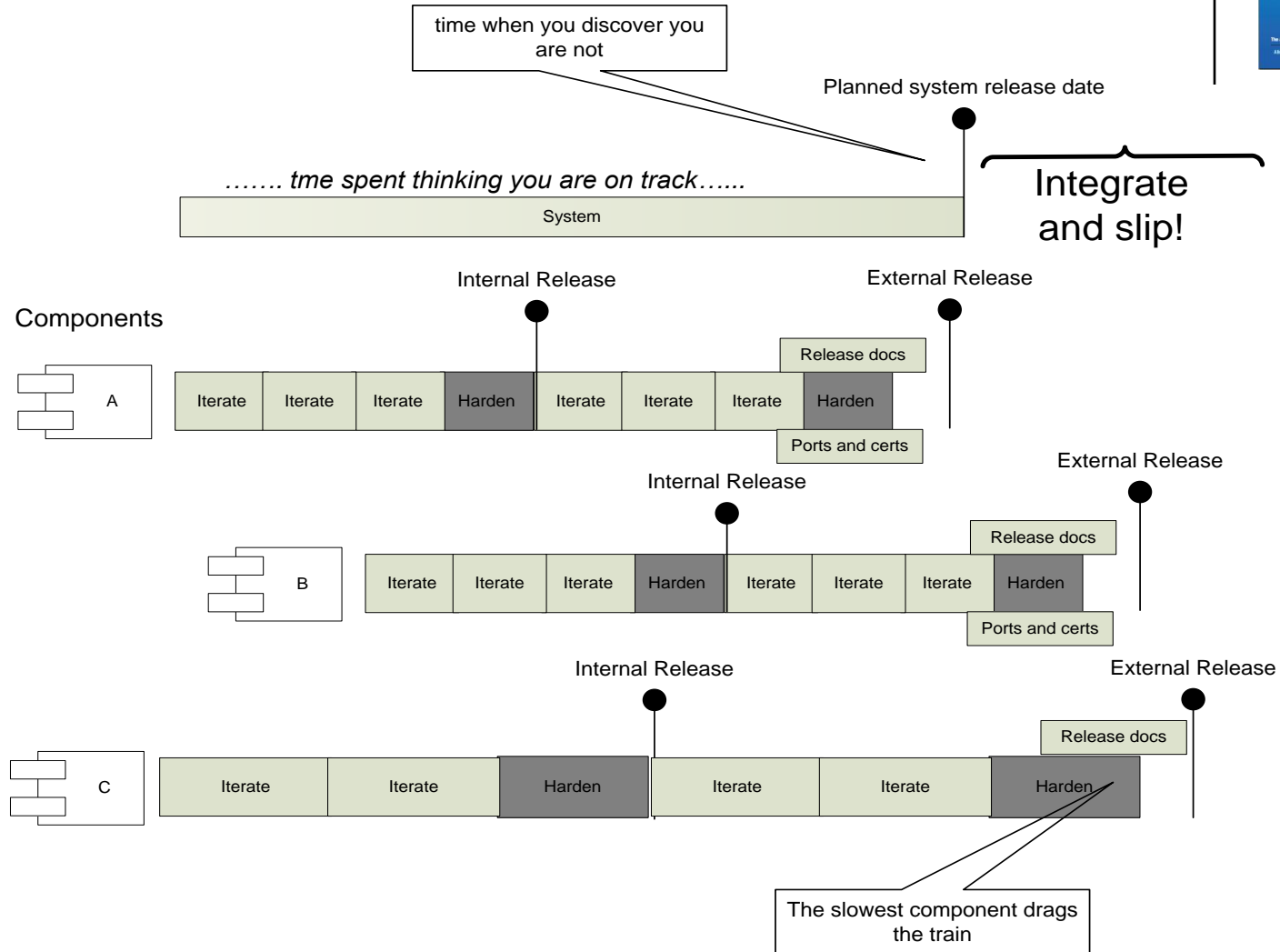
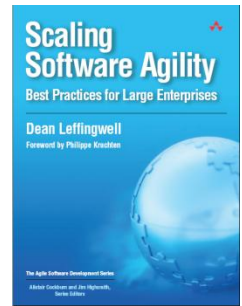
Only the teams themselves can plan and manage this complexity

Only the teams can commit to the schedule

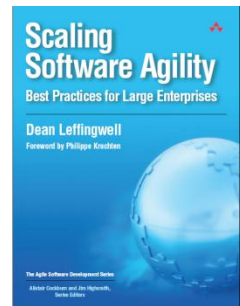
- Collaborative, multi-level Release Planning is the seminal event
- Requires some rules, some practice and an “agile release train” delivery model



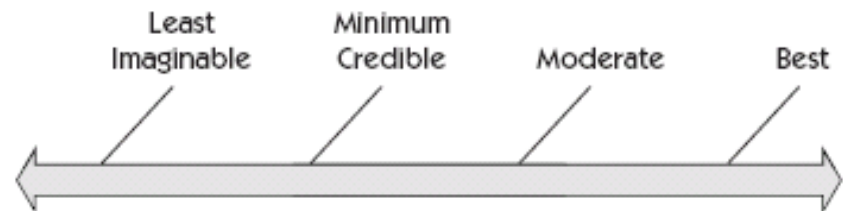
# Component Agile is not System Agile



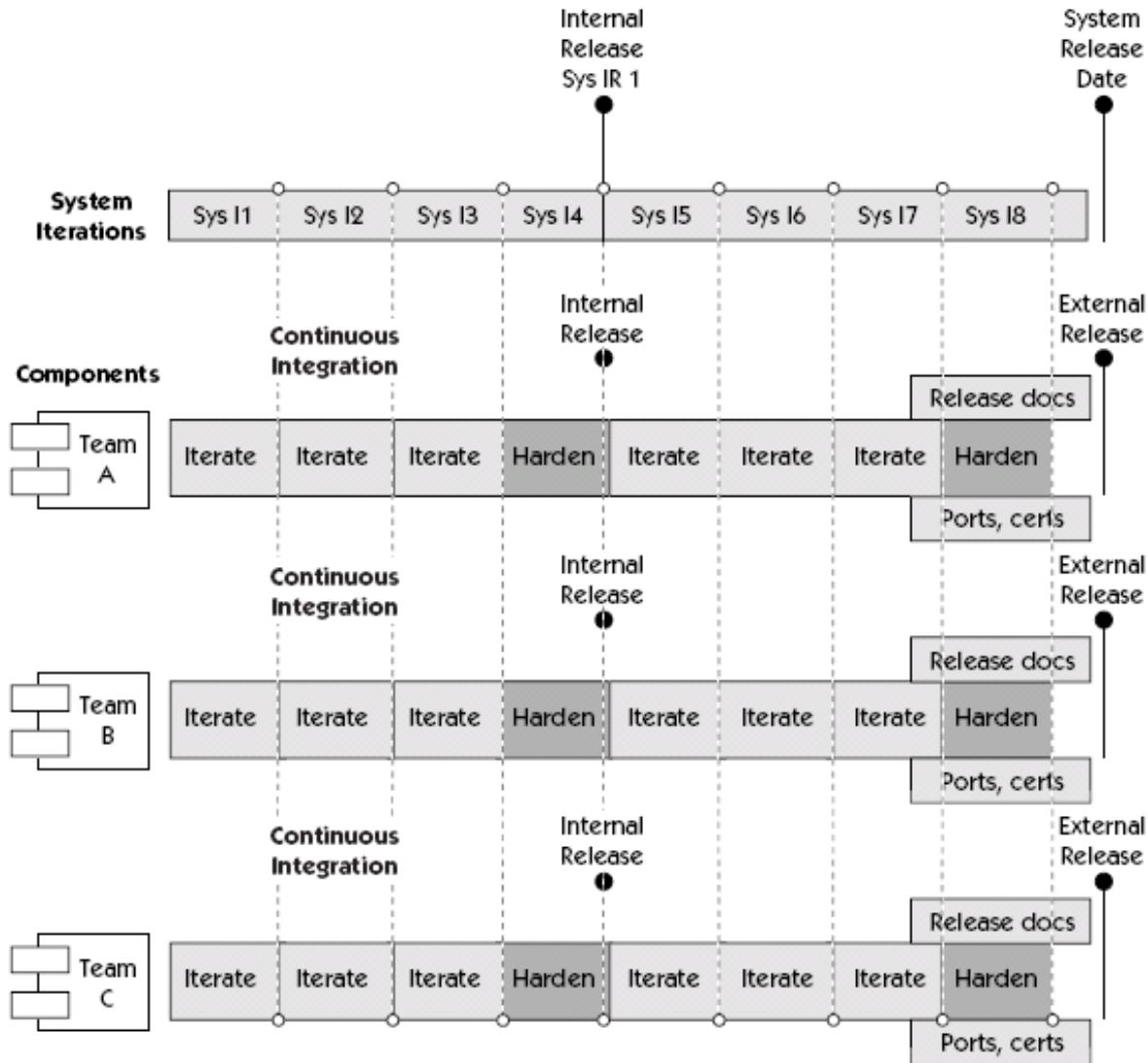
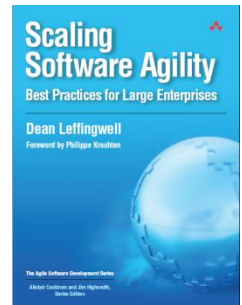
# Rules of the Agile Release Train



- Iteration lengths and release dates are fixed
- Intermediate system integration milestones are established
- Constraining these means that component functionality must flex
- Shared infrastructure components must track ahead
- Component providers evolve to a flexible model:
  - Design spectrum for new functionality
  - Backup plan to ship the old version if necessary

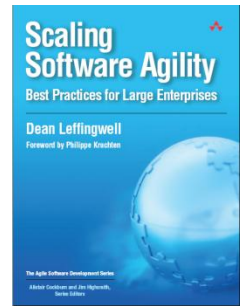


# Synchronized Agile Release Train

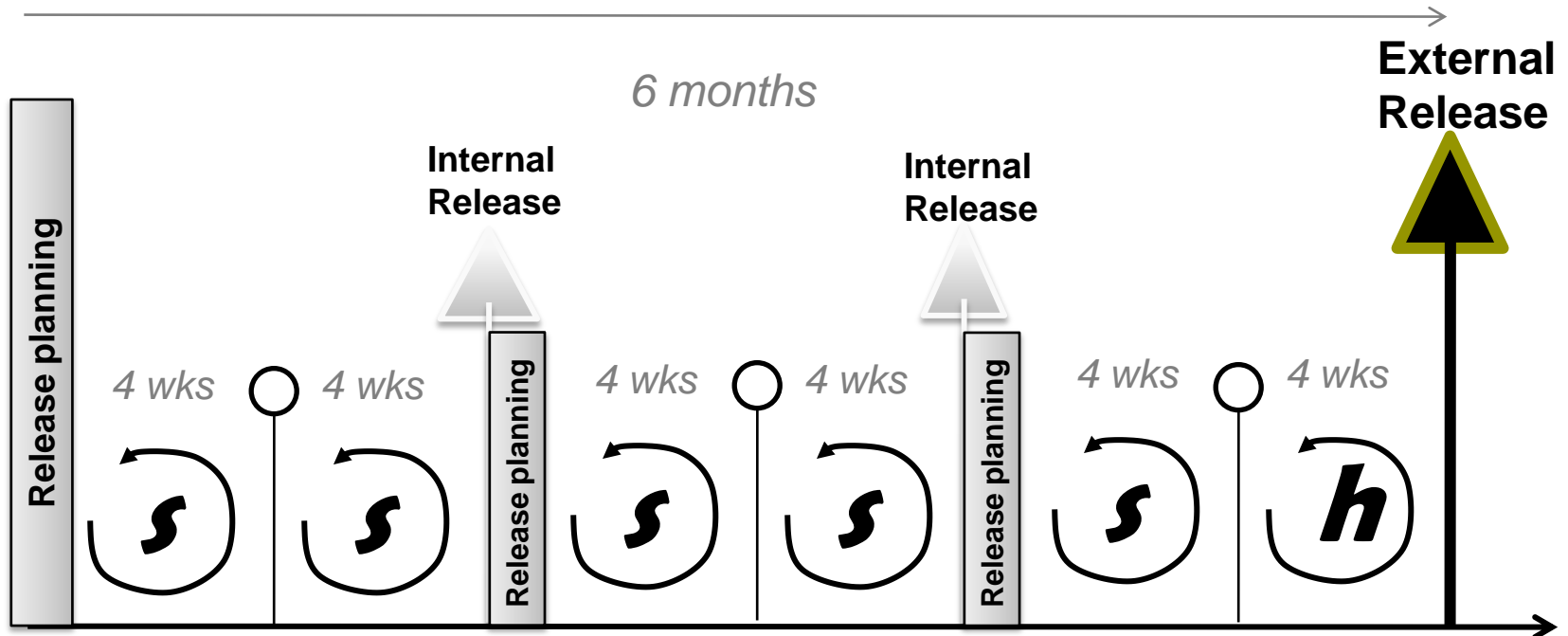


S  
H  
I  
P  
!

# Example Release Planning Cadence



*Six Sprints in a six month time box*



**Legend:**

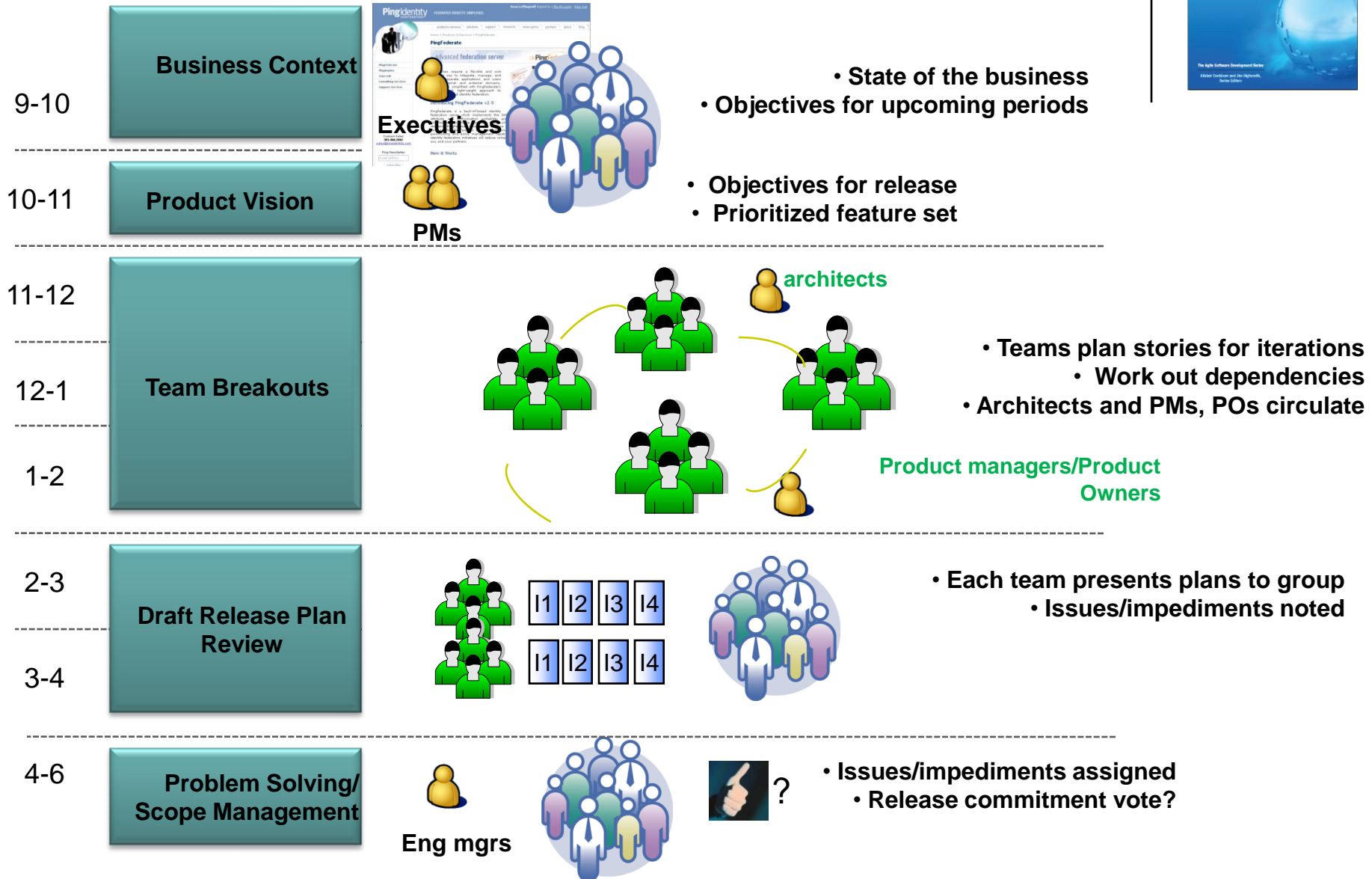
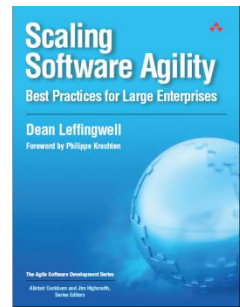
**s** – development sprint

**h** – hardening sprint

**Internal release** – two-sprint, potentially shippable increment

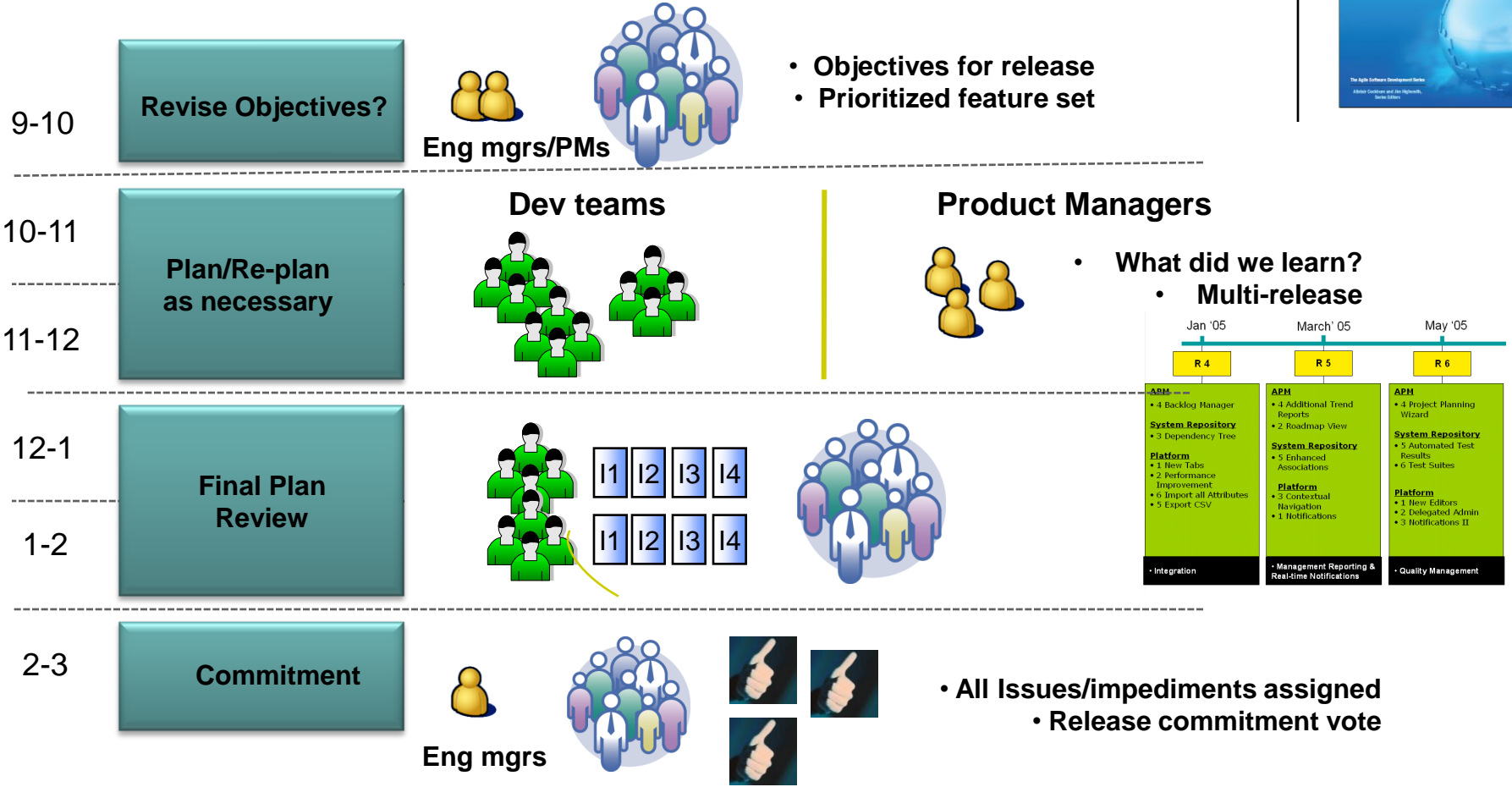
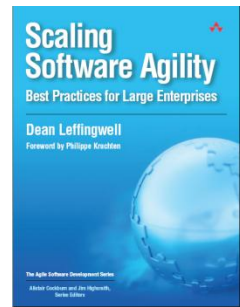
# The Seminal Event

## Release Planning – Day 1

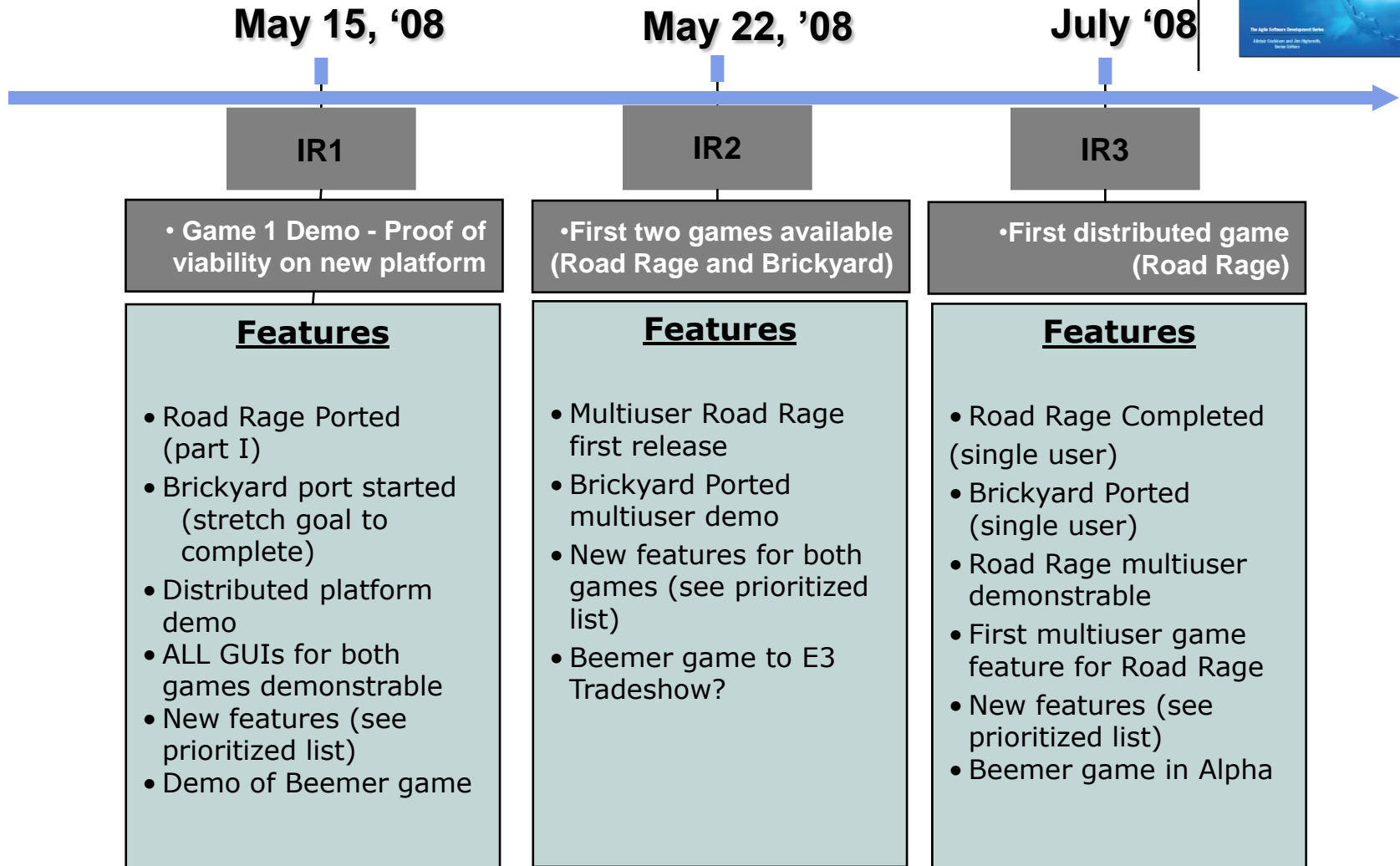
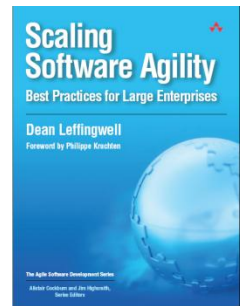


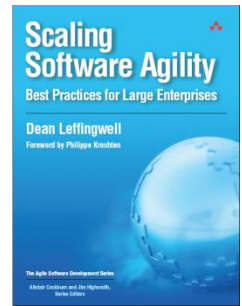
# The Seminal Event

## Release Planning Day 2



# Roadmap Output : System Team's Responsibility





**AND**



**Wait, don't fire!**

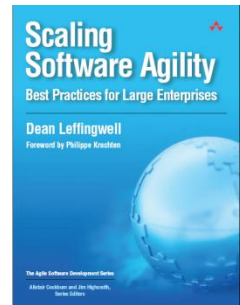
*In the agile enterprise, managements need for results must be greater than the need to control*

# Wait, Don't Fire!

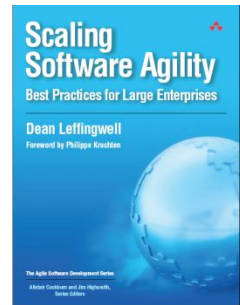
*“Although project teams are largely on their own, they are not uncontrolled. Management establishes enough checkpoints to prevent instability, ambiguity, and tension from turning into chaos.*

*At the same time, management avoids the type of rigid control that impairs creativity and spontaneity. Instead, the emphasis is on “self-control”, “control through peer pressure” and “control by love”.*

–Toyota’s Takeuchi and Nonaka  
The New New Product development Game  
Harvard Business Review, 1986  
(excerpts from the roots of Scrum)

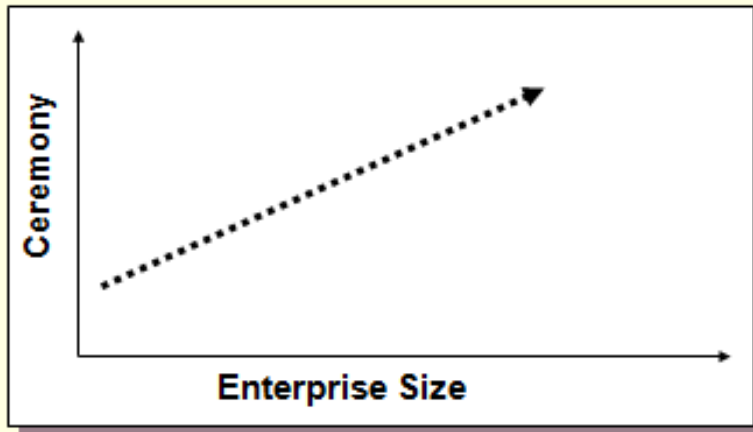
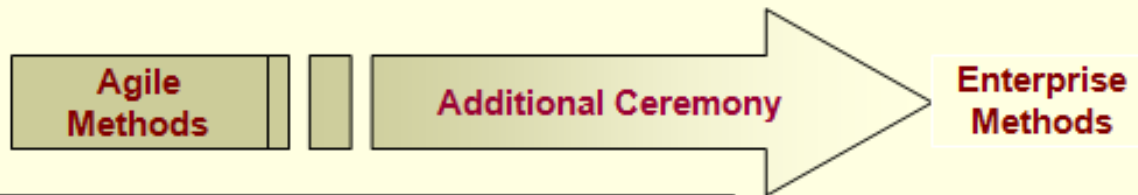


# What your teams may be seeing

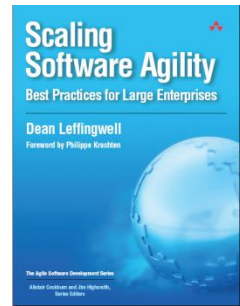


## “Enterprising” Agile Methods?

Too often agile methods are “enterprise” enabled by adding additional prescription and ceremony – thus losing their agility

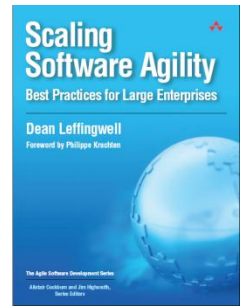


# Agile Guidelines



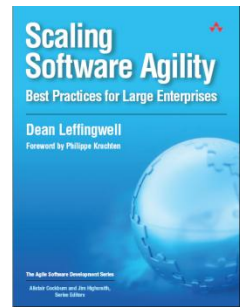
- But it *is* appropriate to create agile guidelines as governance documents
  - What agile means in this company
  - Our expectations for agile behavior
  - Define unambiguously agile mandates
    - Examples: unit testing, retrospectives, daily standup

# But



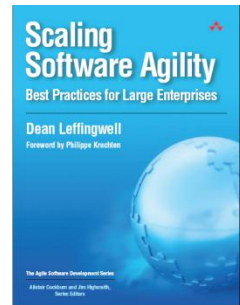
- As lightweight as possible
  - 3-5 pages
  - Serve as templates for additional site-based or project specific guidelines
    - put in place by the local teams themselves
- Recommend, but don't over-prescribe
  - Especially around controversial practices
    - Pair programming, TDD, tooling, requirements management

# Have patience: and watch for these anti-patterns...



- Company likes the potential benefits of agile, but applies the same controls, interrupts and fixed schedule commitments as before
- Insufficient refactoring of testing organizations and inadequate test automation
- Lack of team proficiency in agile technical practices
  - iterations and sprints treated as demo milestones, rather than shippable increments
- Insufficient depth/competency in the critical product owner role
- Inadequate coordination of vision and delivery strategies
  - due to lack of coordinated, multi-level release planning

# More from Dean Leffingwell



- *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley 2007
- Blog and Resources
  - [www.scalingsoftwareagility.wordpress.com](http://www.scalingsoftwareagility.wordpress.com)
- Website
  - [www.leffingwell.org](http://www.leffingwell.org)
- Reach me at [deanleffingwell@gmail.com](mailto:deanleffingwell@gmail.com)