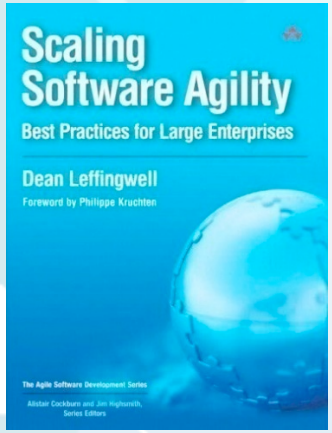
Scaling Software Agility: Best Practices for Large Enterprises

Dean Leffingwell

Agile 2009

Chicago, IL

August 26, 2009



About Dean Leffingwell

Author

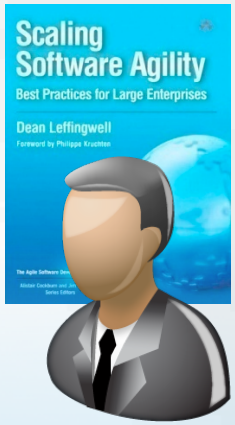
Coach

Agile Enterprise Coach
Nokia S60, Symantec, Safeco Insurance, Symbian Software….

Executive Mentor
BMC Agile Transformation

Cofounder/Advisor
Ping Identity, Roving Planet, Rally Software

Executive

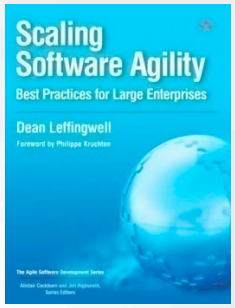


Founder and CEO
ProQuo, Inc., Internet identity

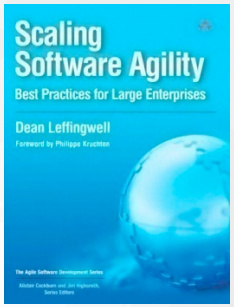
Senior VP
Rational Software
Responsible for Rational Unified Process (RUP) & Support of UML

Founder/CEO
Requisite, Inc.
Makers of RequisitePro

More from Dean Leffingwell



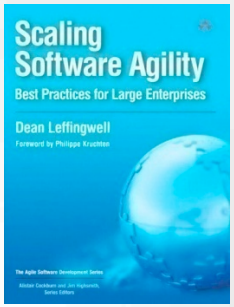
- *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley 2007
- Blog and Resources
  - www.scalingsoftwareagility.wordpress.com
- Website
  - www.leffingwell.org
- Reach me at DeanLeffingwell@gmail.com



If you accept the premise that market needs change faster than the software industry's traditional ability to develop solutions, you're left with the question "what can we do about it?" For me, the answer is Agile.

Israel Gat, Vice President,
Infrastructure Management, BMC Software, Inc.

BMC Results

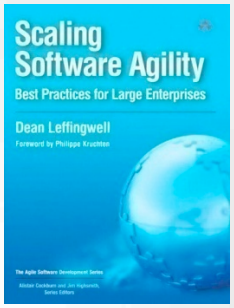


QSM Associates press release

- *… remarkable levels of time-to-market and quality*
- *… produce large scale enterprise software in 4-5 months, compared to typical one year*
- *… exceptional time-to-market without sacrificing quality*
- *… especially noteworthy - BMC 'Secret Sauce' enables process to succeed in spite of geographically dispersed teams*
  - *"Other companies experience higher defects and longer schedules with split teams, BMC does not. I've never seen this before. The low bug rates also result in very low defect rates post-production"*
- *… clearly ahead of more than 95 percent of all the software projects captured in the SLIM metrics database,* ***they're among the best I've seen***

Source: QSM Associates Press Release, Sep 10, 2007

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?

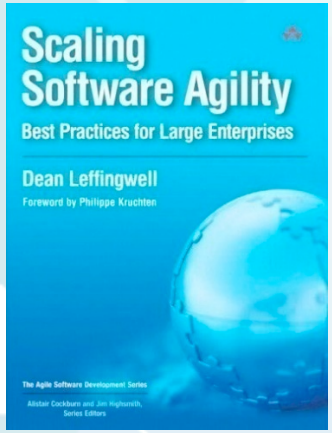
You realize how little you know, and you face your own failures and redo it again, and at the second trial you realize another mistake . . . So you can redo it once again.

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."

Fuijo Sho, President, Toyota

What Is Software Agility?



Team Agility

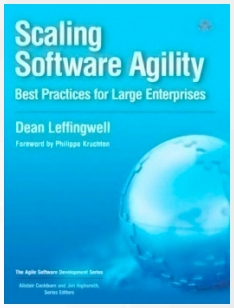
A disciplined set of

- enhanced software engineering practices
- empirical software project management practices
- modified social behaviors

That empowers teams to:

- more rapidly deliver quality software
- explicitly driven by intimate and immediate customer feedback

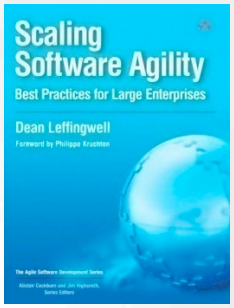
Achieving Team Agility



Seven Agile Team Practices that Scale

1. The Define/Build/Test Team
2. Mastering the Iteration
3. Two-levels of Planning and Tracking
4. Smaller, More Frequent Releases
5. Concurrent Testing
6. Continuous Integration
7. Regular Reflection and Adaptation

Enterprise Agility



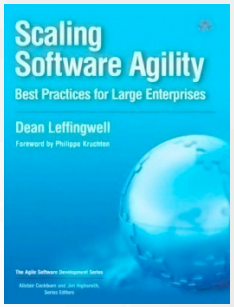
A set of

- organizational best practices
- core values and beliefs

That harness large numbers of agile teams to build and release quality enterprise-class software more rapidly than ever before

Explicitly driven by intimate and immediate customer feedback

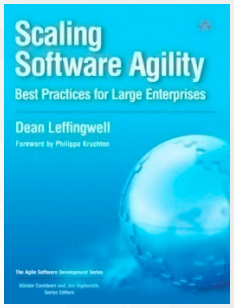
Achieving Enterprise Agility



Seven Enterprise Practices

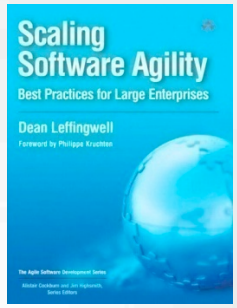
1. Intentional Architecture
2. Lean Requirements at Scale
3. Systems of Systems and the Agile Release Train
4. Managing Highly Distributed Development
5. Impact on Customers and Operations
6. Changing the Organization
7. Measuring Business Performance

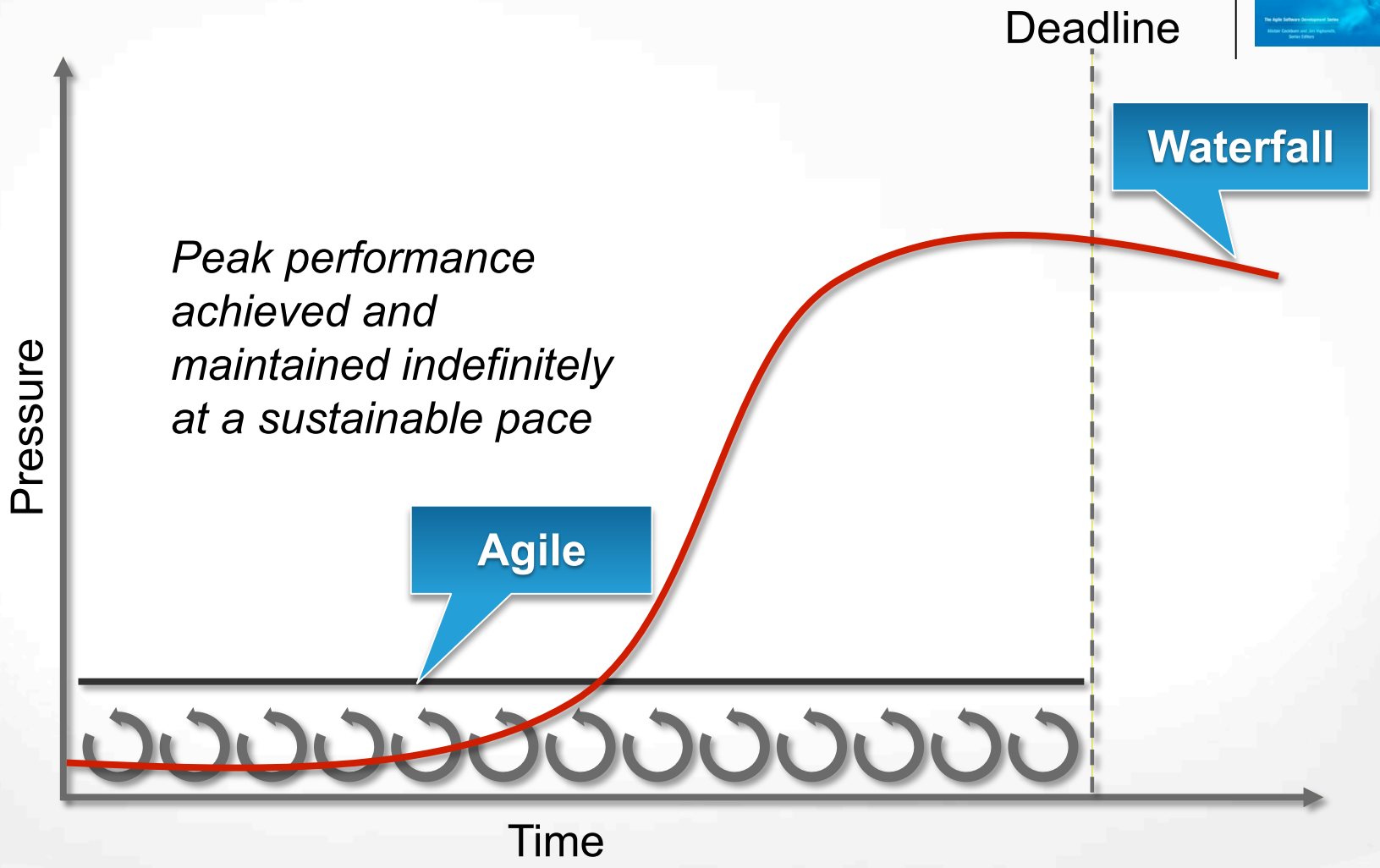
Agile Turns Tradition Upside-Down



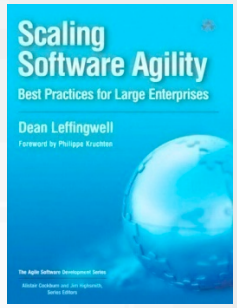
	Predictive Process (Waterfall)	Adaptive Process (Agile)
Constraints	Requirements	Cost / Schedule
	Plan Driven	Value/Vision Driven
Estimates	Cost / Schedule	Features
	The plan creates cost/schedule estimates	*The vision creates feature estimates*

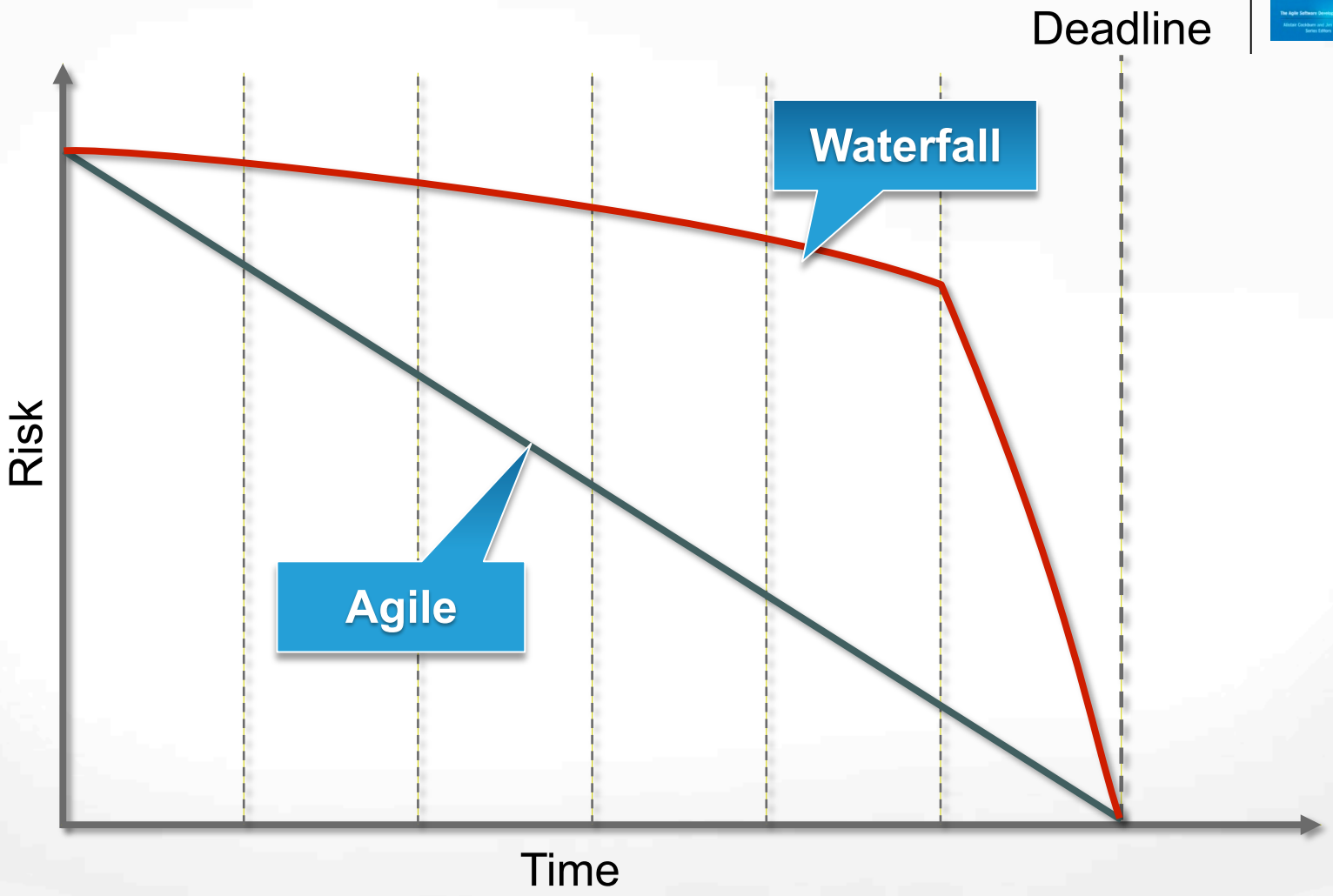
Helps Avoid the Death March



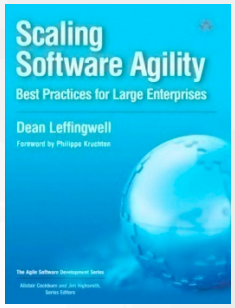


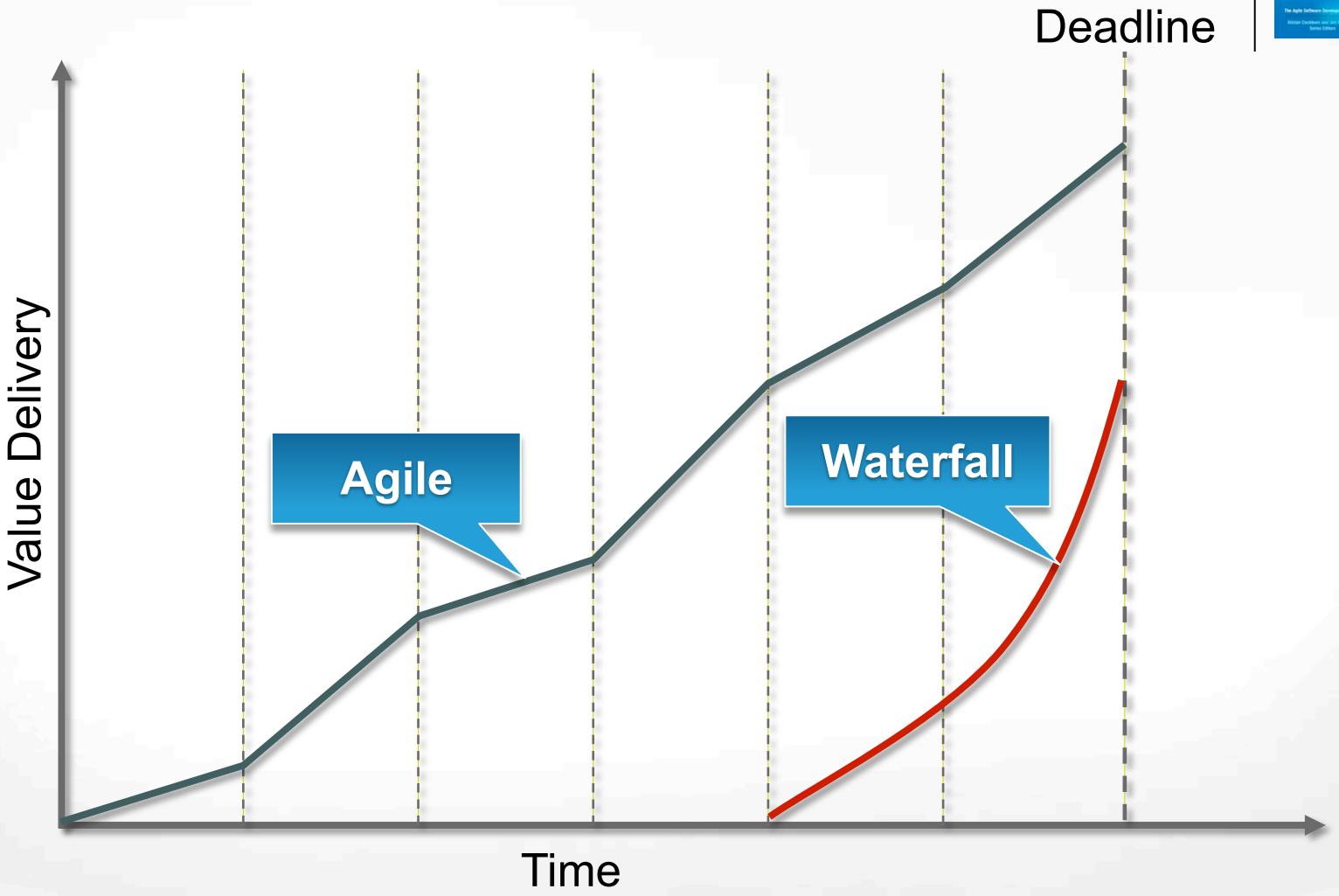
Reduces Risk



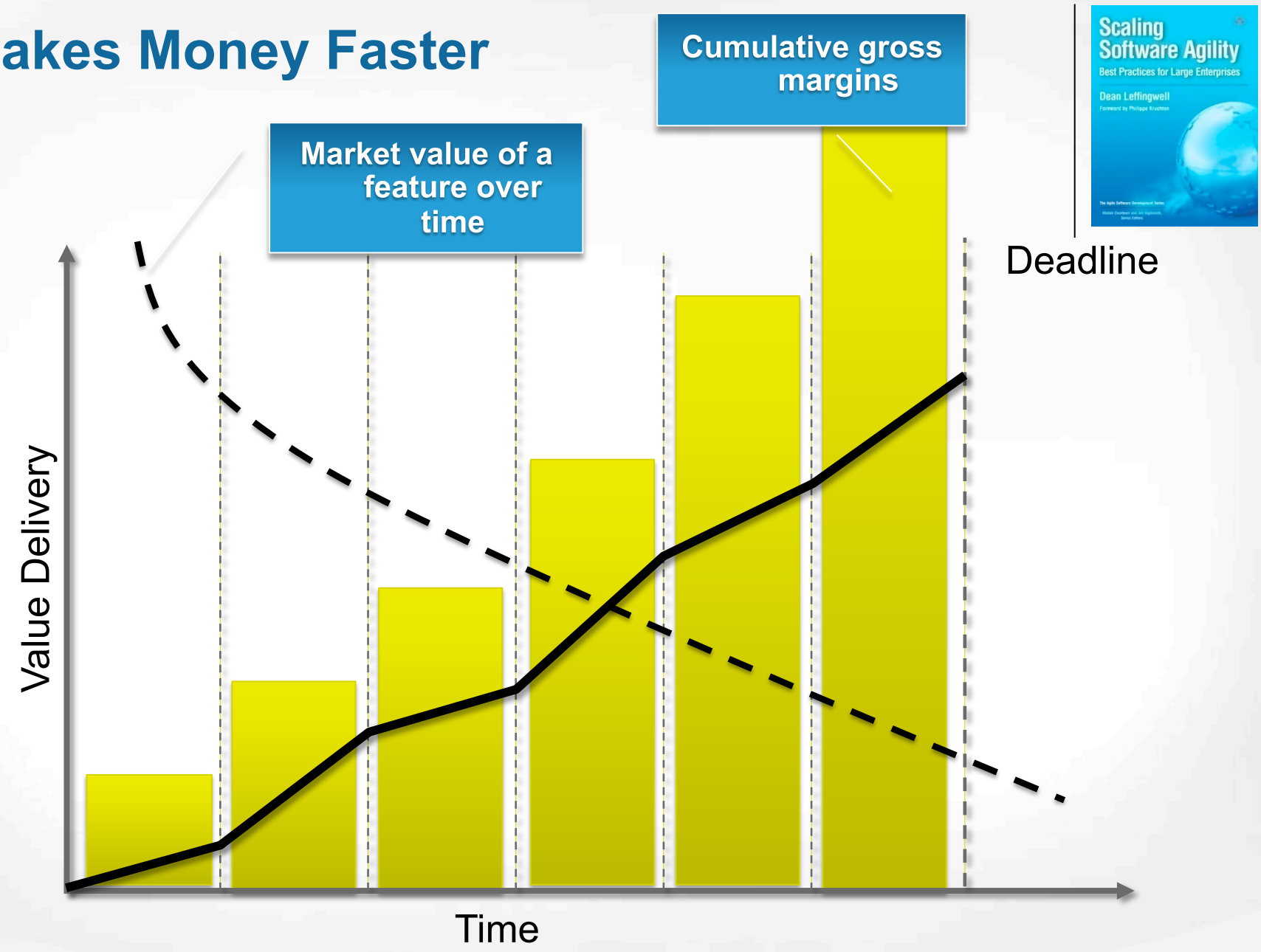


Starts Delivering Immediately

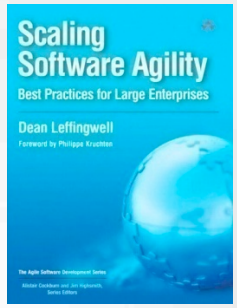


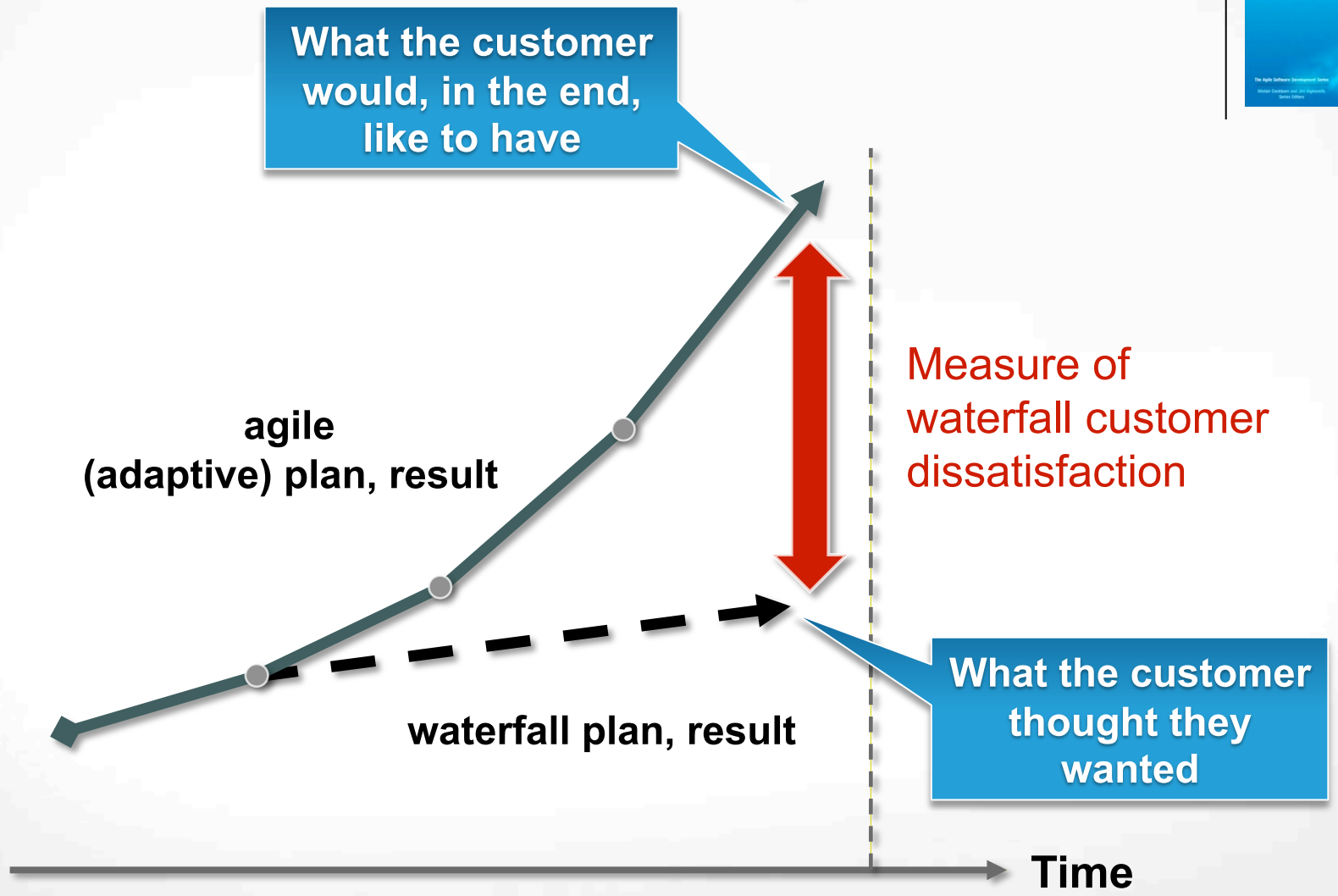


Makes Money Faster

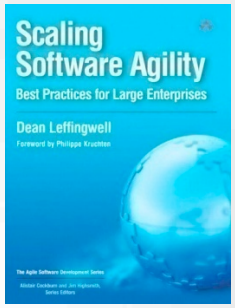


Delivers Better Fit for Purpose

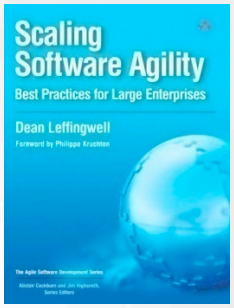




Agile Delivers Higher



Quality



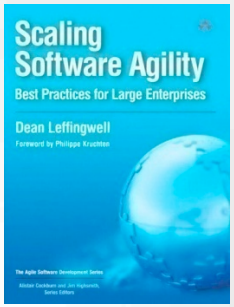
- *Our implementation of agile practices . . . helps us find bugs earlier, helps us achieve higher quality, and helps us work well with SW QA*

  Jon Spence, Medtronic

- *I measure quality by the life of a defect, time measured from injection to finding and fixing. Agile gives us solid results with most defects living no longer than one to two iterations. Agile delivers higher quality than anything I've found with the waterfall model*

  Bill Wood, VP, Ping Identity Corp.

Productivity



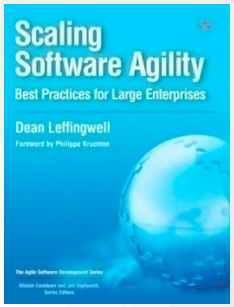
- *Last year, we had 22 releases across 3 major product lines, and not a one of them was late. We support hundreds of Fortune 1000 enterprises with a single person dedicated to support -- the software is that solid*

  Andre Durand, CEO, Ping Identity Corp.

- *We increased individual developer and team productivity by an estimated 20 percent to 50 percent*

  BMC Software

Morale



- *Development teams are more engaged, empowered and highly supportive of the new development process*

  BMC Software

- *Our implementation of agile practices . . . (1) makes the work more enjoyable, (2) helps us work together, and (3) is empowering*

  Jon Spence, Medtronic

Seven Agile Team Practices That Scale

The Define/Build/Test Team

Mastering the Iteration

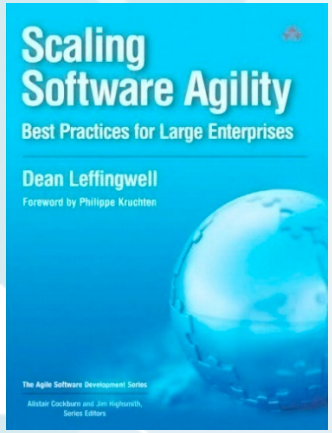
Two-level Planning and Tracking

Smaller, More frequent releases

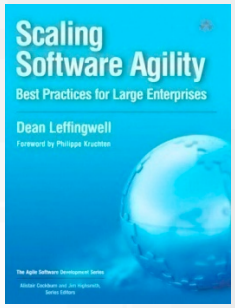
Concurrent Testing

Continuous Integration

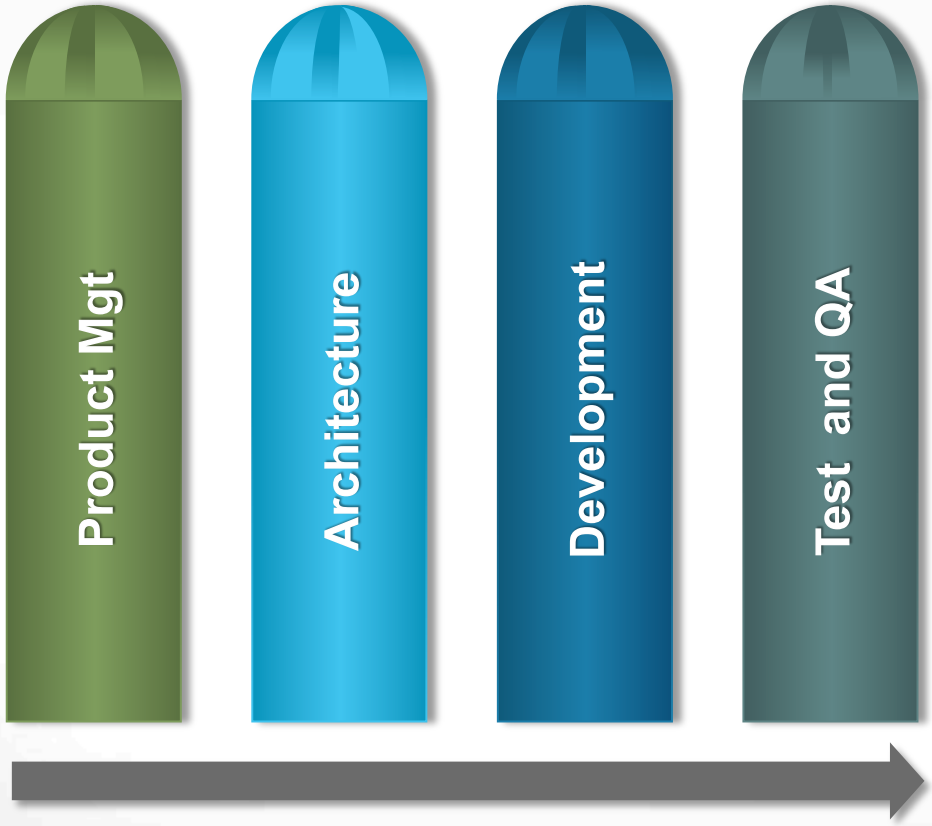
Regular Reflection and Adaptation



1. Define/Build/Test Team



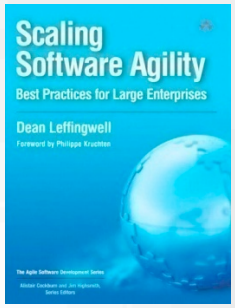
Before Agile: Typical Functional Silos



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

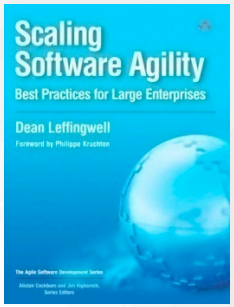
Management Challenge:
Connect the Silos

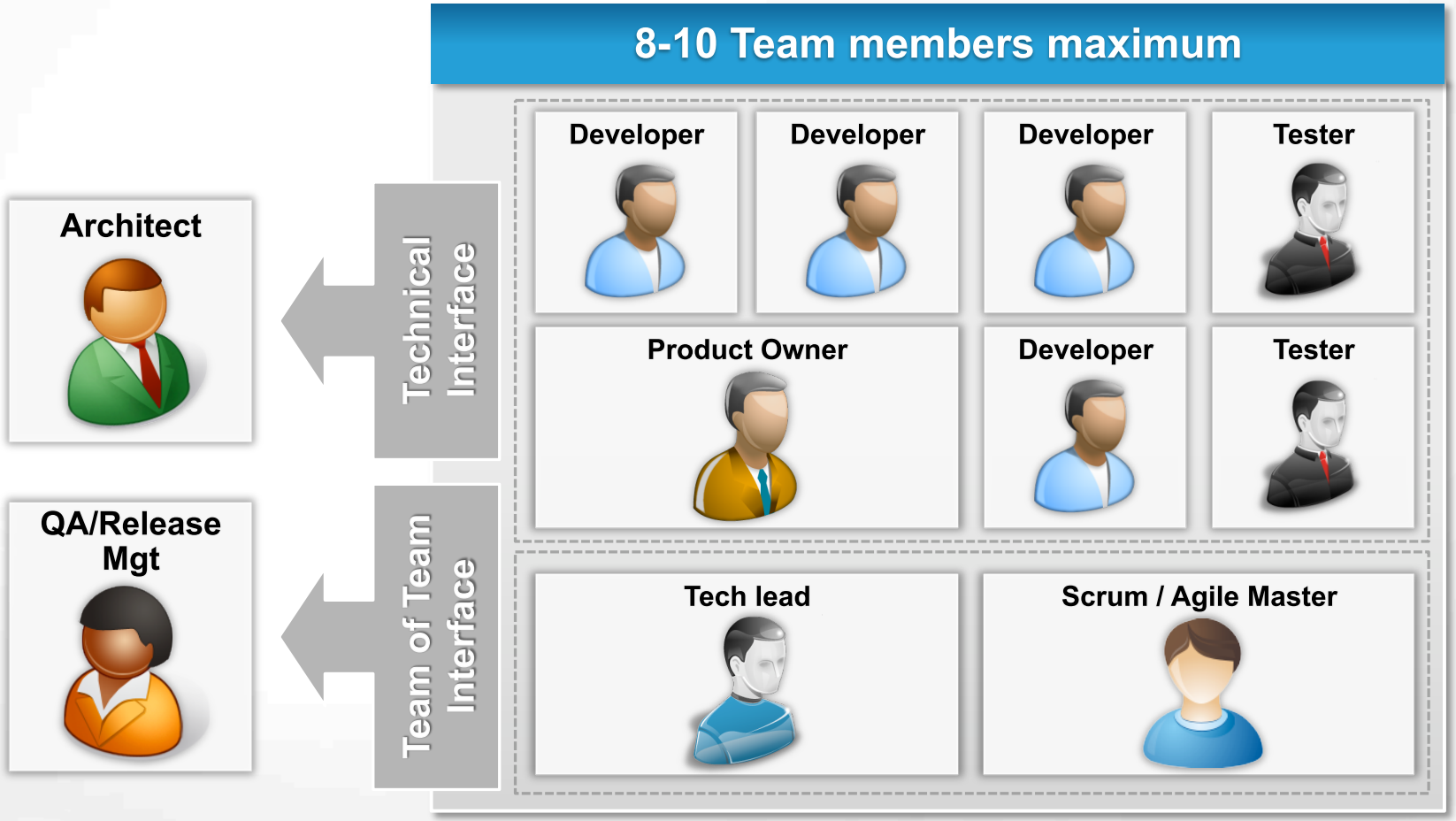
D/B/T Team – Agile Fractal



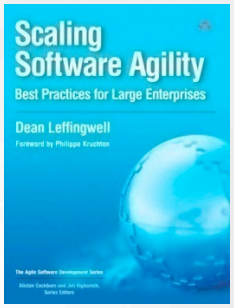
- A self-organizing team that can ***Define***, ***Build*** and ***Test*** a thing of interest
- Optimized for communication about the thing
- Repeated at larger scales to produce larger systems
- Teams can be based on
  - Components
  - Subsystems
  - Features
  - Interfaces
  - Products

D/B/T Teams Have the Necessary Skills



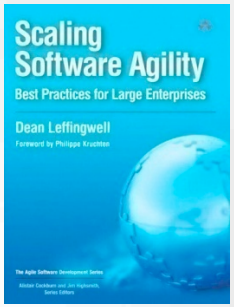


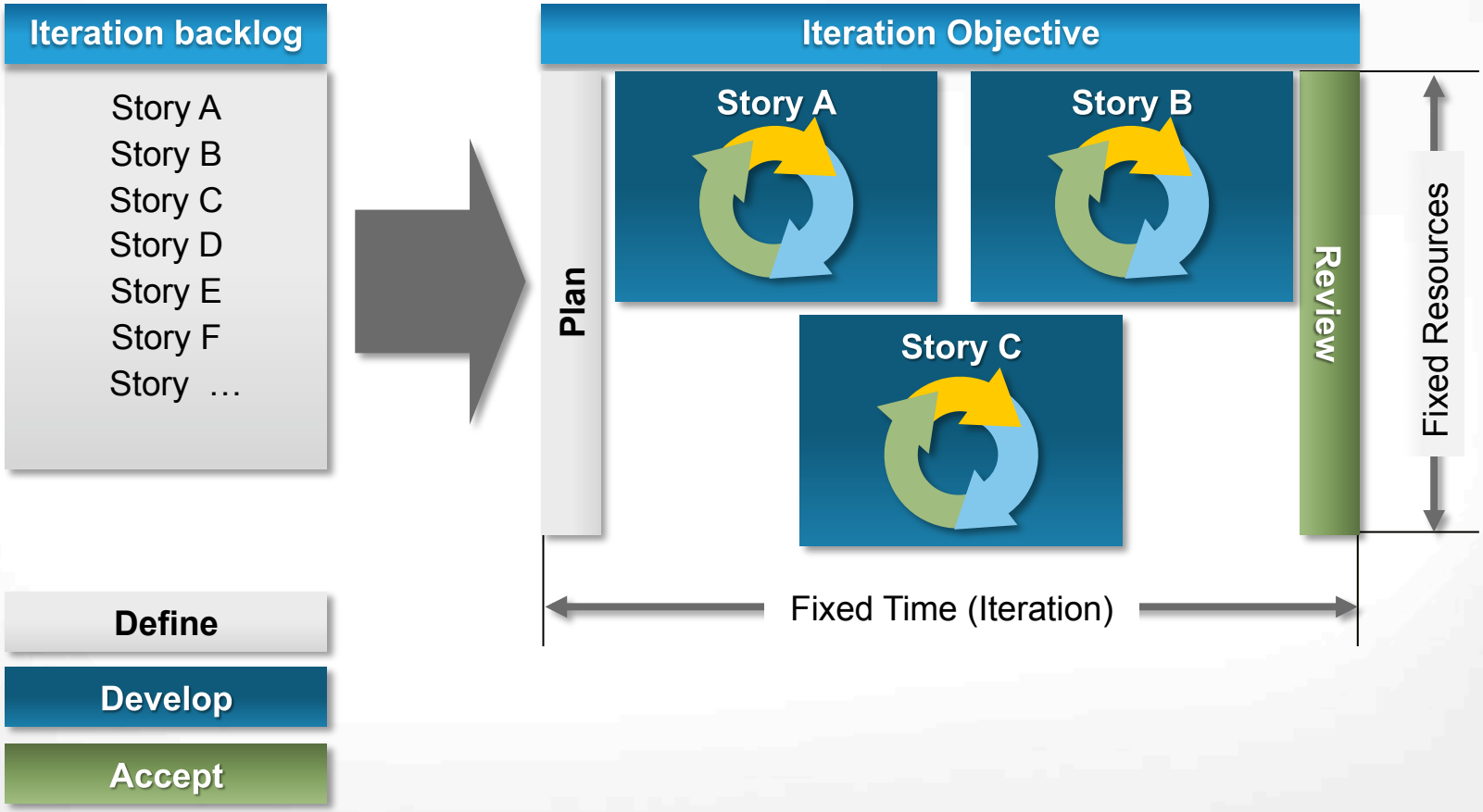
2. Mastering the Iteration



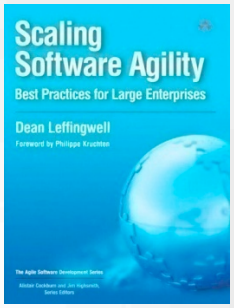
The iteration is the heartbeat of agility. Master that, and most other things agile tend to naturally fall into place.

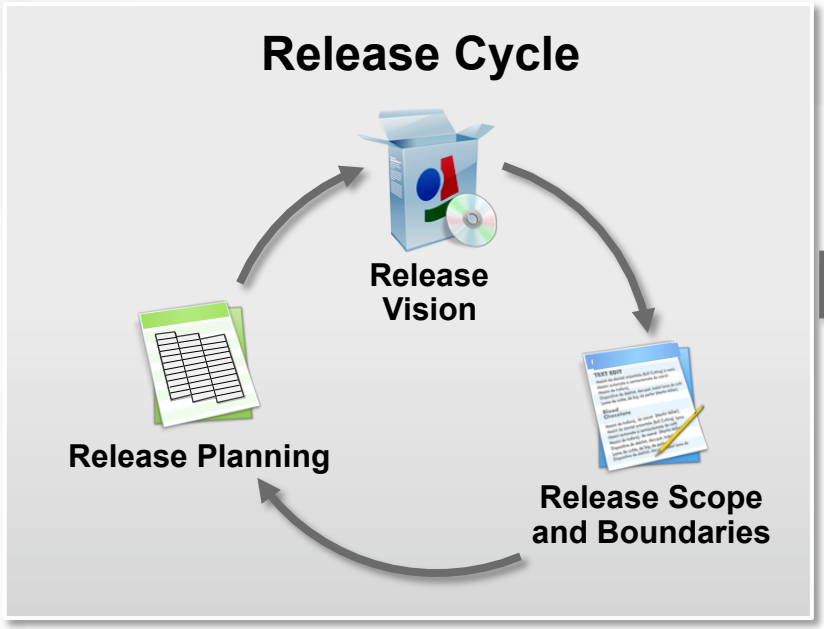
Iteration Pattern





3. Two-Level Planning and Tracking

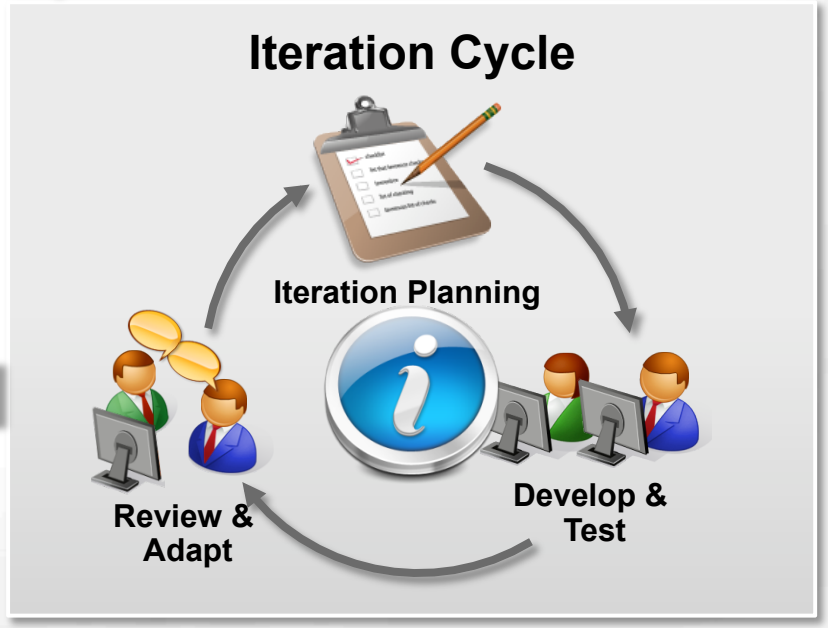




Drives

Plan iterations at the ***component/feature l***evel

- 2-4 iteration visibility
- Currency: user stories

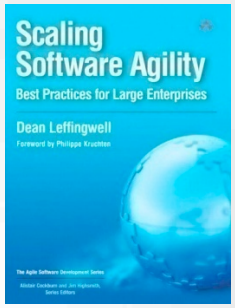


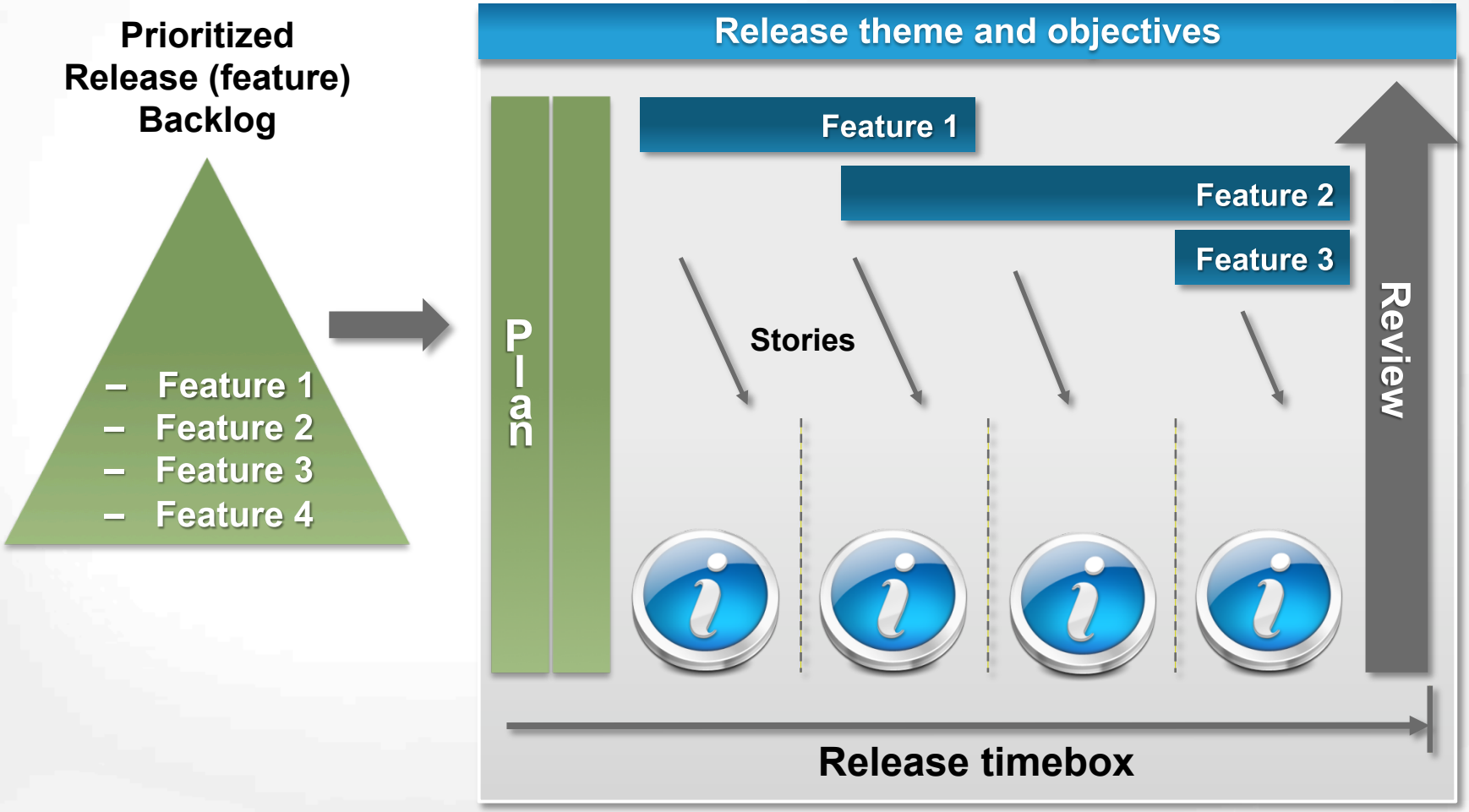
Plan Releases at the ***System*** Level

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

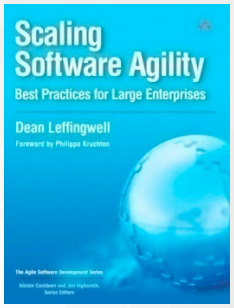
Feedback - Adjust

Release Pattern

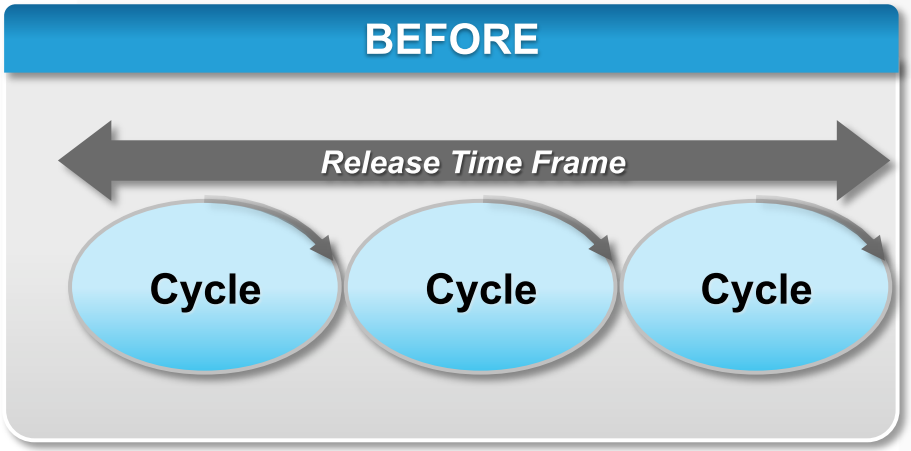


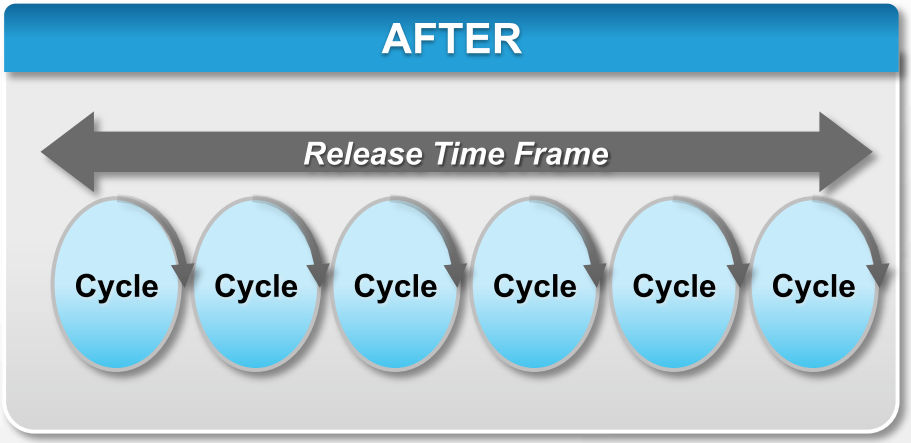


4. Smaller, More Frequent Releases

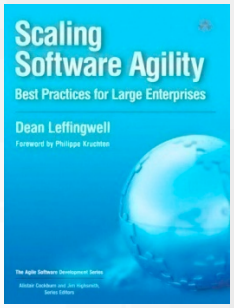


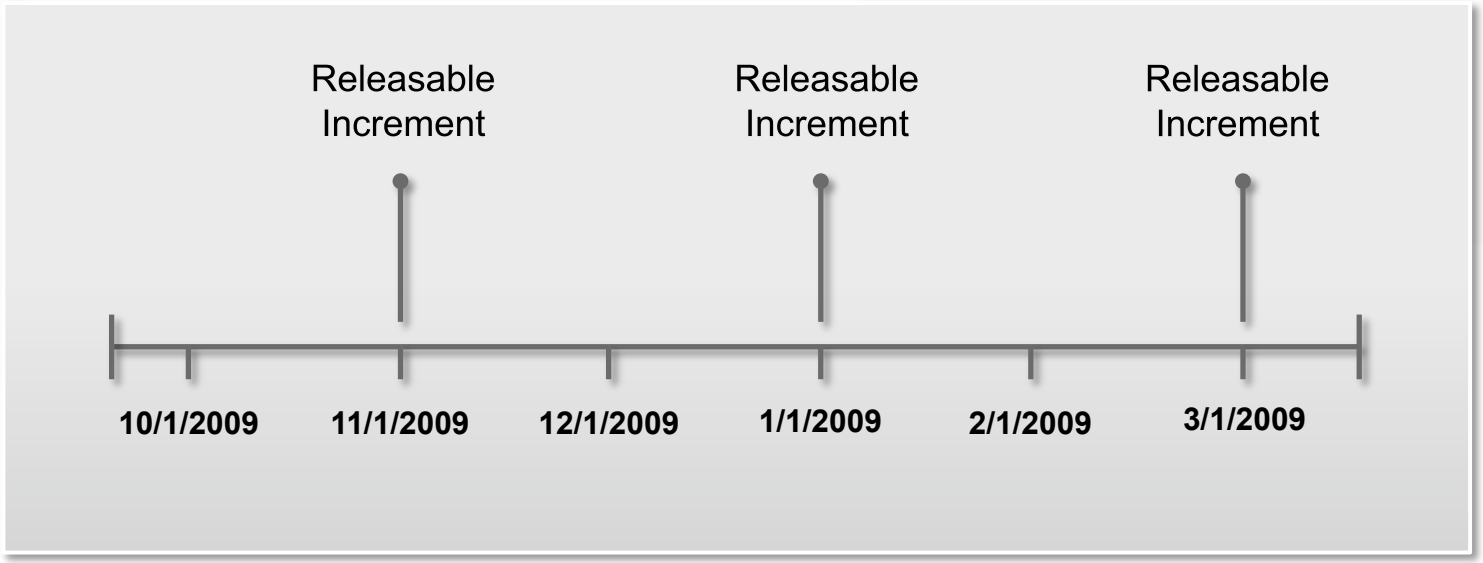
- Shorter release dates
  - 60-120 days
- Releases defined by
  - Date, theme, planned feature set, quality
- Scope is the variable
  - Release date and quality are fixed





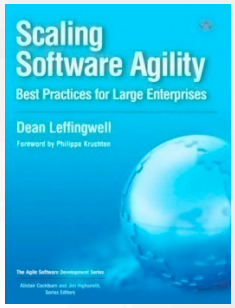
Fix the Dates - Float the Features





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

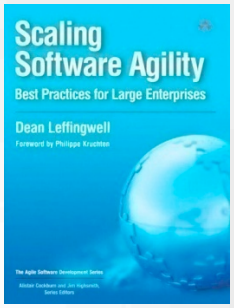
5. Concurrent Testing



Philosophy of Agile Testing

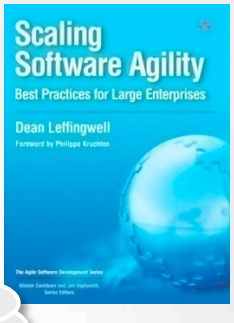
- All code is tested code. Teams get no credit for delivering functionality that is coded, but not tested.
- Tests are written before, or concurrently with, the code itself.
- Testing is a team effort. Testers and developers all write tests.
- Test automation is the rule, not the exception.

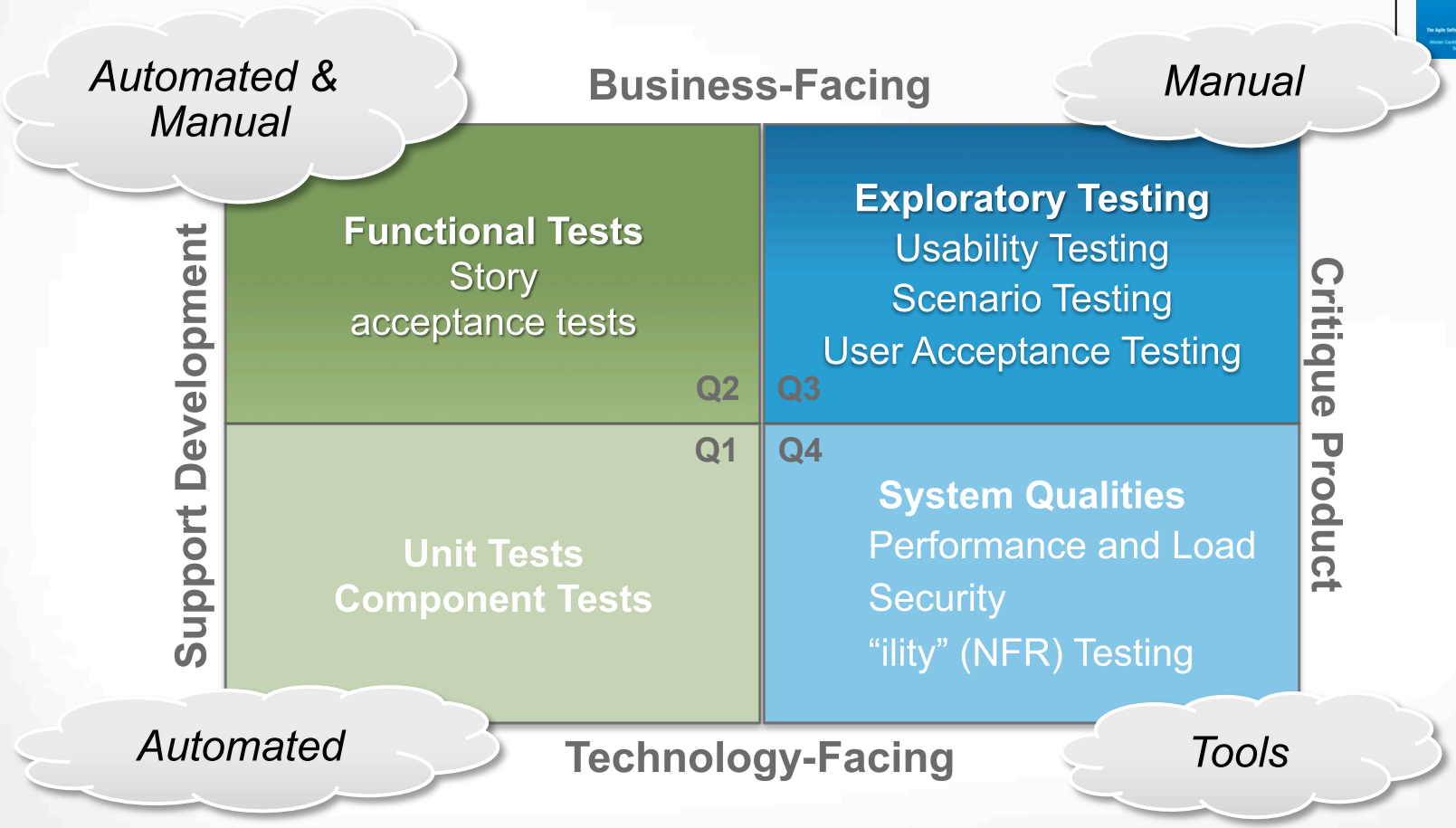
Concurrent



- 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

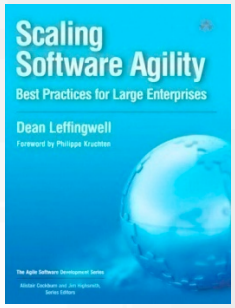
Agile Testing Quadrants





Adapted from Brian Marick, Crispen and Gregory

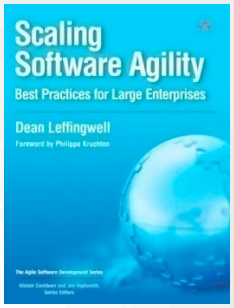
On Test Automation



Automate Now

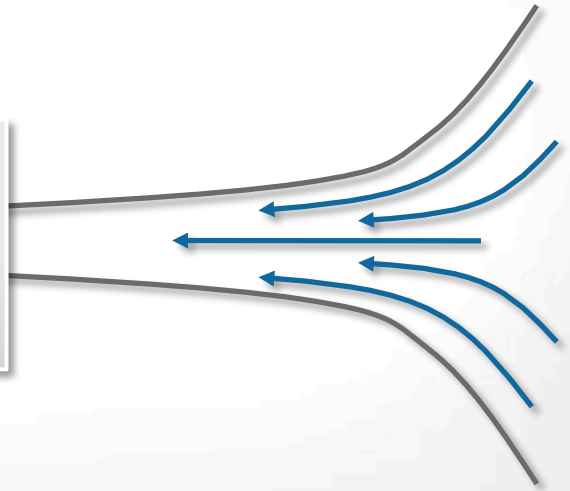
- You have no choice
- Manual tests bottleneck velocity
- You can't ship what you can't test

6. Continuous Integration

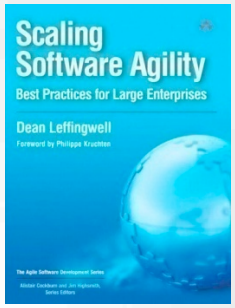


- Continuous integration is neither new nor invented by agile
- It has been applied as a best practice for at least a decade
- However, continuous integration is mandatory with agile

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



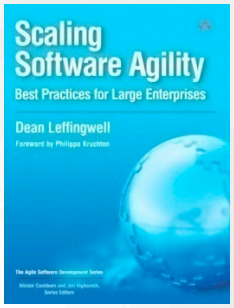
Continuous Integration Success



- 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

Source: Martin Fowler

Memo from an XP Shop

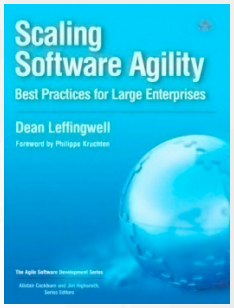


"The XP environment provides us with many benefits, not the least of which is the incredible pace of progress we are so proud of. Lately we have had a rash of build failures, some related to infrastructure issues, but more related to carelessness. Broken builds destroy the "heartbeat" of an XP team. Each of you has a primary responsibility to ensure that this doesn't happen . . . but here are a few tips to ensure that you aren't the one who broke the build:

- *Write your test cases before you write the code*
- *Build and test the code on your desktop BEFORE you check it in*
- *Make sure you run all of the cases that the build does*
- *Do not comment-out inconveniently failing unit tests. Find out why they are broken, and either fix the test or fix your code*
- *If you are changing code that may affect another team, ASK before you check it in*
- *Do not leave the building until you are SURE your last check-in built successfully and the unit tests all ran*

The Build master is there to make sure that broken builds get addressed, not to address them. The responsibility for a broken build is yours. Breaking the build ***will have an affect on your standing within the team*** *and, potentially, your review, so let's be careful out there."*

7. Regular Reflection and Adaptation



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

Agile Manifesto, Principle 12

- Periodically, the entire team including owners/end users
  - reflects on the results of the process
  - learn from that examination
  - adapt the process - and organization - to produce better results
- The team decides what is *working well*, what *isn't*, and what one thing to *do differently* next time

The shorter the iterations and releases – the faster the learning

Achieving Enterprise Agility

Intentional Architecture

Lean Requirements at Scale

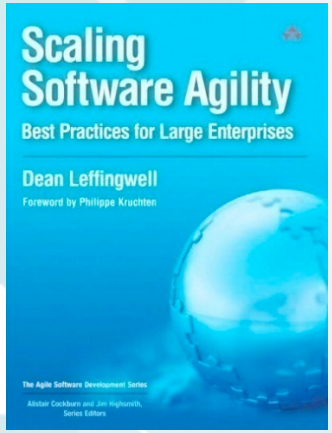
Systems of Systems and the Agile Release Train

Managing Highly Distributed Development

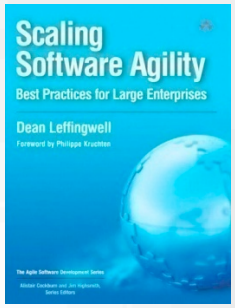
Impact on Customers and Operations

Changing the Organization

Measuring Business Performance



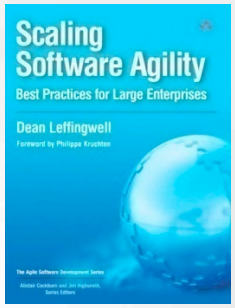
1. Intentional Architecture



- 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.

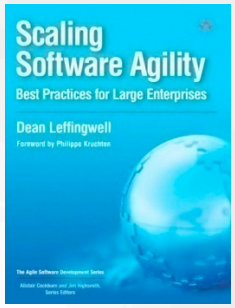
Principle #4 They build it, they test it.

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



Customers

Product Manager

Product Vision

Product Owner

System/ Portfolio Team

Value Stories
+ Arch. Spikes

Component Team

Design Spike

System Architect

Tech lead/ architect

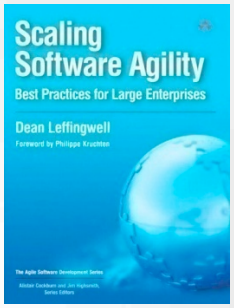
Agile Master

Architectural
- Inputs
- Ideas
- Constraints

Architectural Evolution

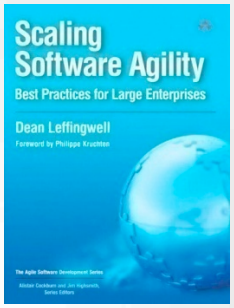
Stakeholders

2. Lean Requirements at Scale

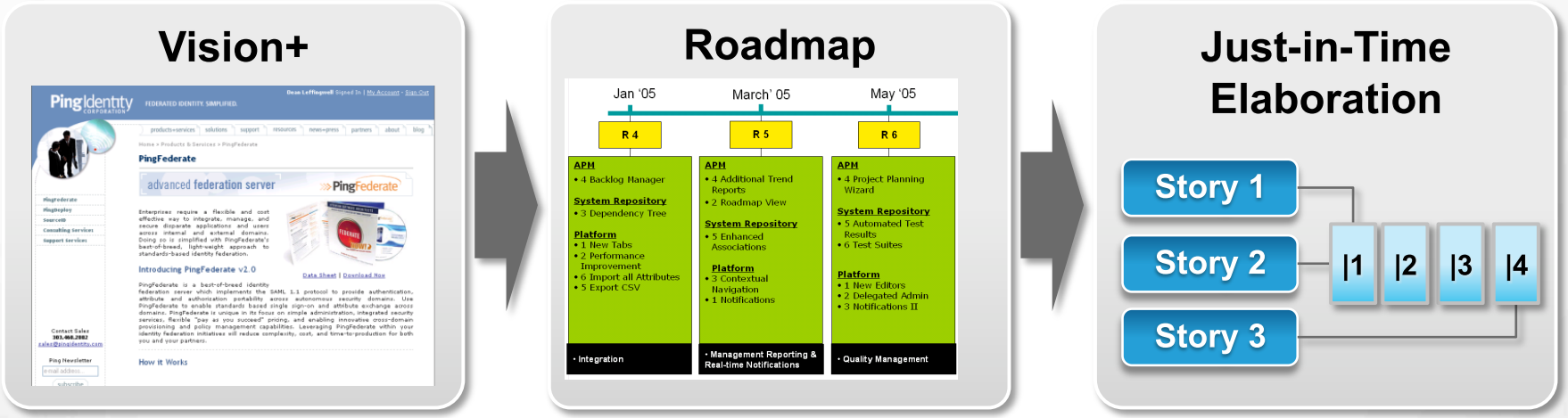


Requirements still matter in agile. At scale, lean and more extensible requirements practices can be applied.

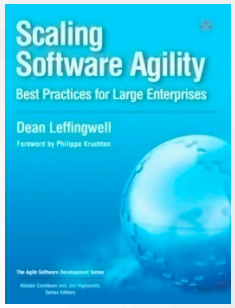
Lean Requirements at Scale



A scalable requirements practice with three elements

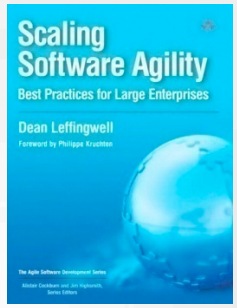


Vision - Management's responsibility

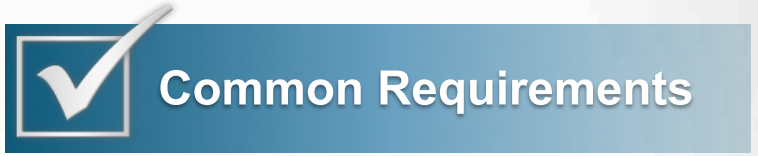


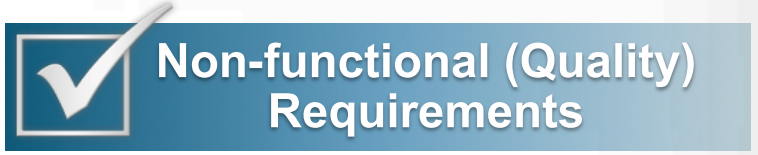
- Where are we headed as a business?
- What problem does this product solve?
- What **features** and benefits does it provide?
- For whom does it provide it?
- What performance does it deliver?

Common and Non-functional Requirements



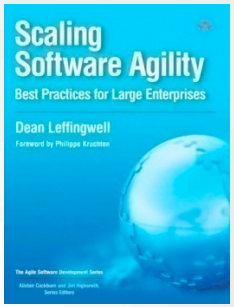
- Some requirements must be known by all teams
  - Common components, common behaviors
  - Internationalization, accessibility
  - Performance, reliability and security requirements
  - Industry/Regulatory/Customer standards/specifications
  - Corporate standards: copyright, logo, graphics, legal

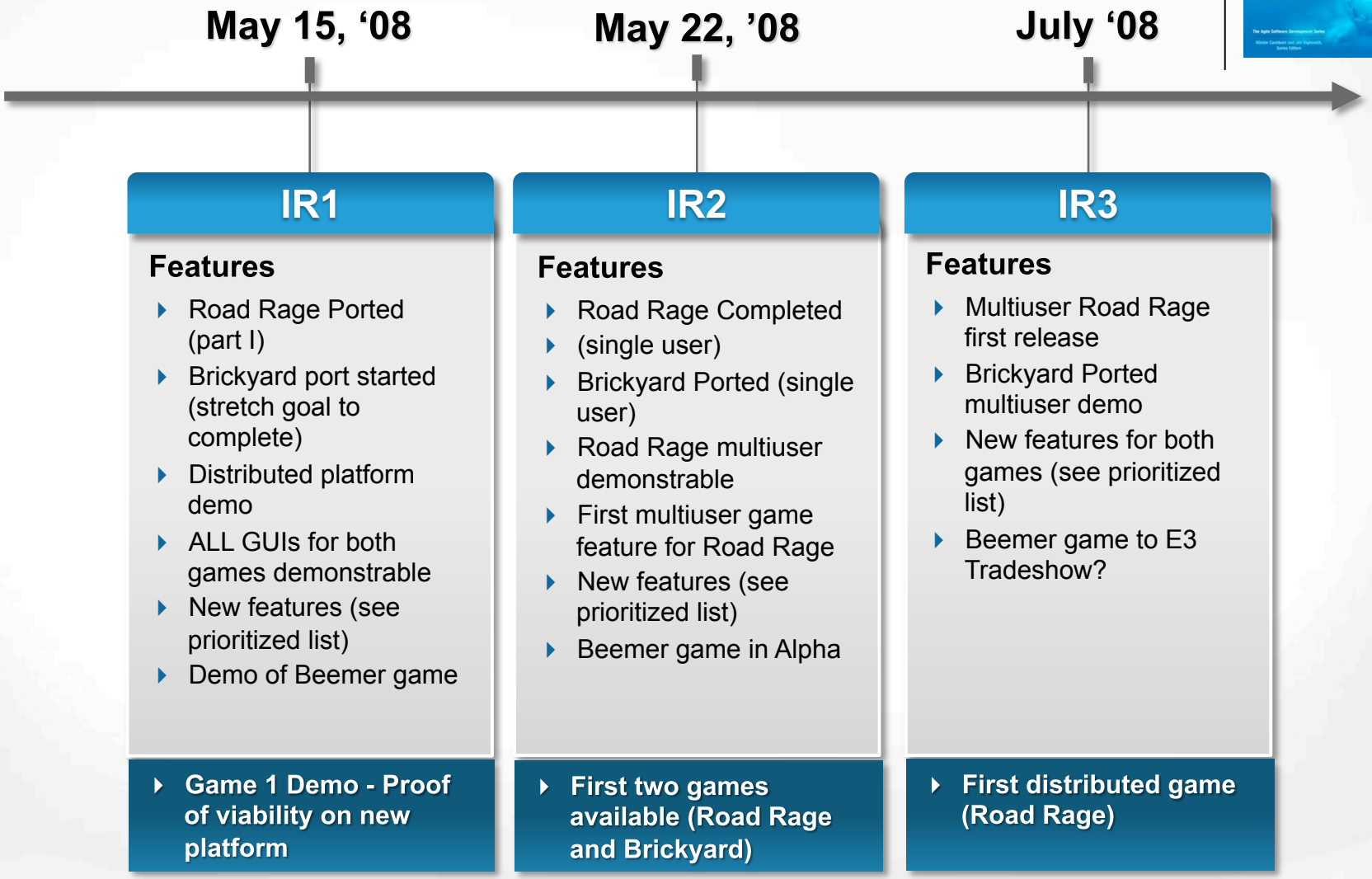




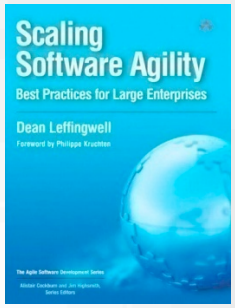
Documented and available to all affected teams.

Roadmap – System Team's Responsibility

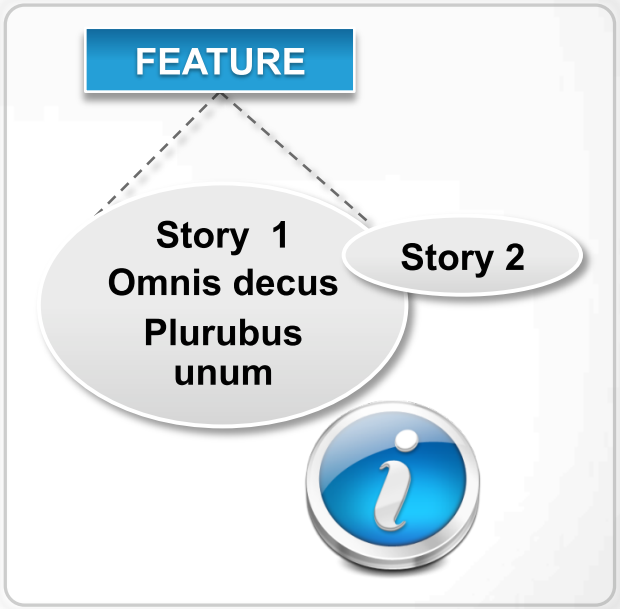




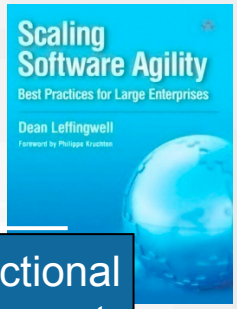
Just-In-Time Elaboration – Agile Team's Responsibility



- 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

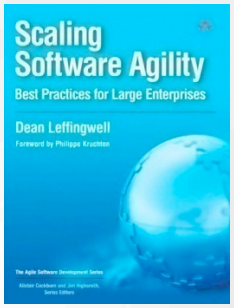


At scale, not everything is a user story



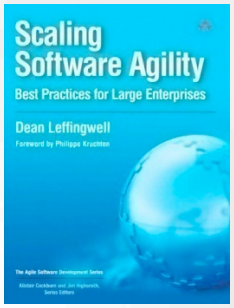
Backlog Item
Constrained by
0..
0..*
Non-functional Requirement
Compliant when passes
0..*
1..*
System Validation Test
Is one of
Investment Themes
Realized by
0, 1
1..*
Epic
Realized by
0,1
1..*
Feature
Realized by
0,1
1..*
Story
Implemented by
1
1..*
Task
*
version
1..*
1..*
Is one of
User Story
Other Work Item
Done when passes
1..*
Acceptance Test
Consists of
1
1..*
1..*
Unit Test
Functional Test

3. Systems of Systems and the Agile Release Train



- Scaling agile requires managing interdependencies amongst teams of developers
- Only the teams themselves can plan and manage this complexity
- Only the teams can commit to the schedule
- Systematic enterprise delivery requires an “agile release train” delivery model
- Rolling-wave Enterprise Release Planning drives release train vision and execution

Component Agile is not System Agile



Time when you discover you are not

Planned system release date

…….time spent thinking you are on track…….

System

Integrate and slip!

Agile teams

A

Internal Release

External Release

Release docs

Iterate | Iterate | Iterate | Harden | Iterate | Iterate | Iterate | Harden

Port and certs

B

Internal Release

External Release

Release docs

Iterate | Iterate | Iterate | Harden | Iterate | Iterate | Iterate | Harden

Port and certs

C

Internal Release

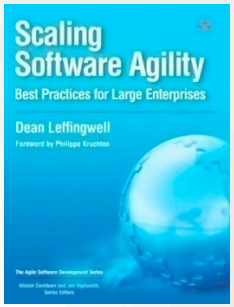
External Release

Release docs

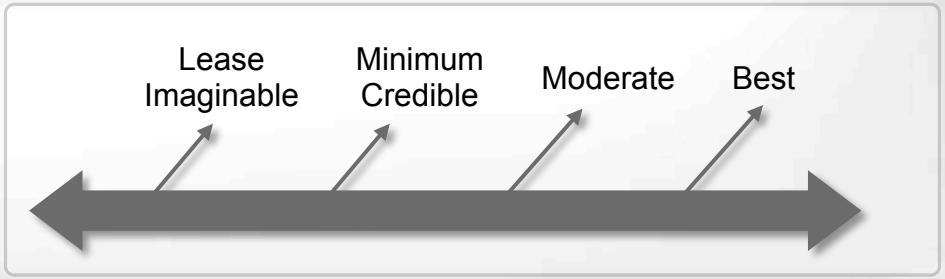
Iterate | Iterate | Harden | Iterate | Iterate | Harden

The slowest team drags the train

Rules of the Agile Release Train

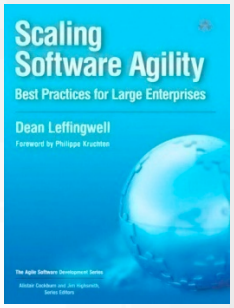


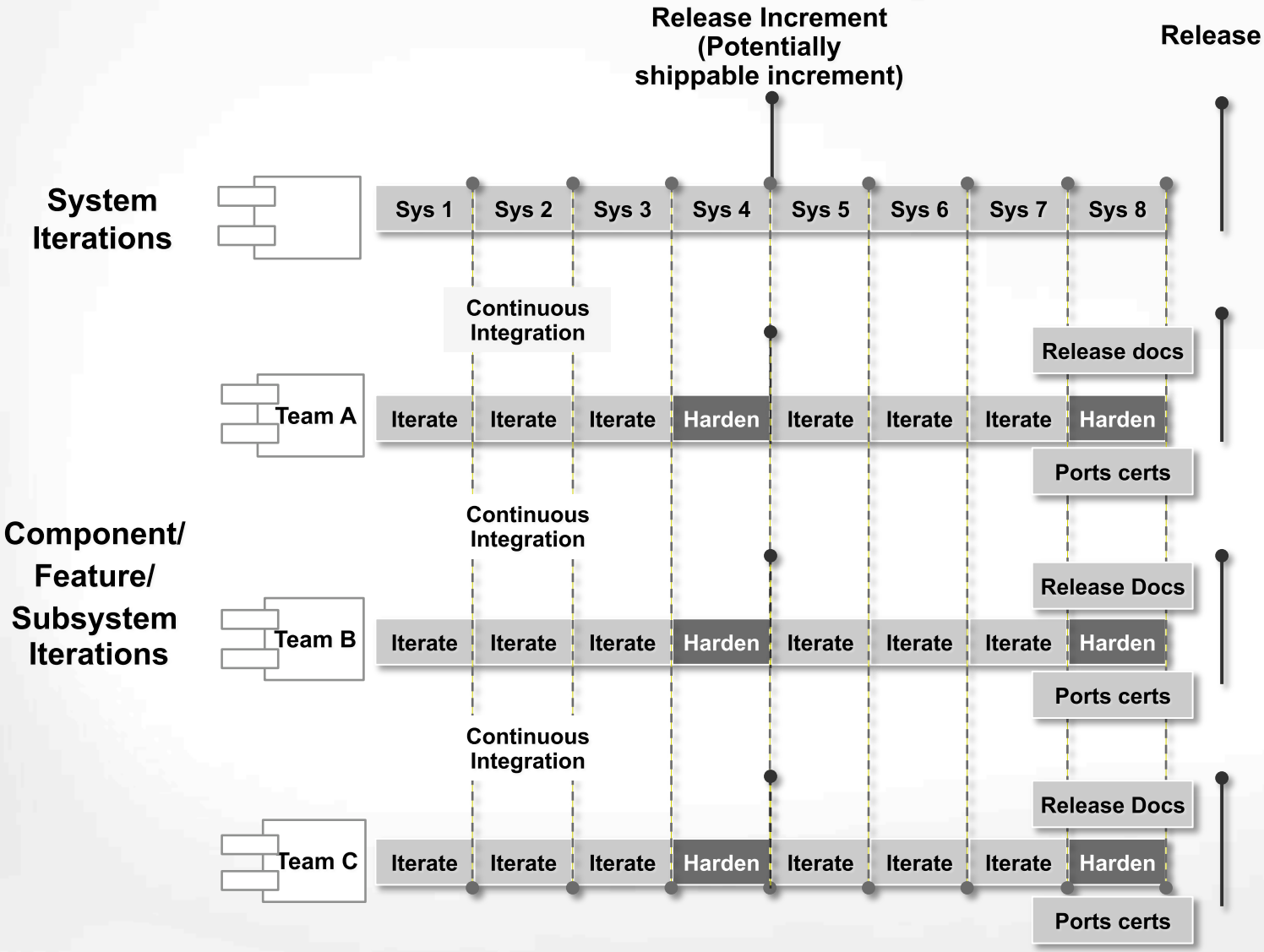
- Periodic release dates for the solution are fixed
- Intermediate, global integration milestones are established and enforced
- Constraining these means that component/feature functionality must flex
- Shared infrastructure must track ahead
- Teams evolve to a flexible model:
  - Design spectrum for new functionality
  - Backup plan to ship less capable version if necessary



Agile release train design continuum

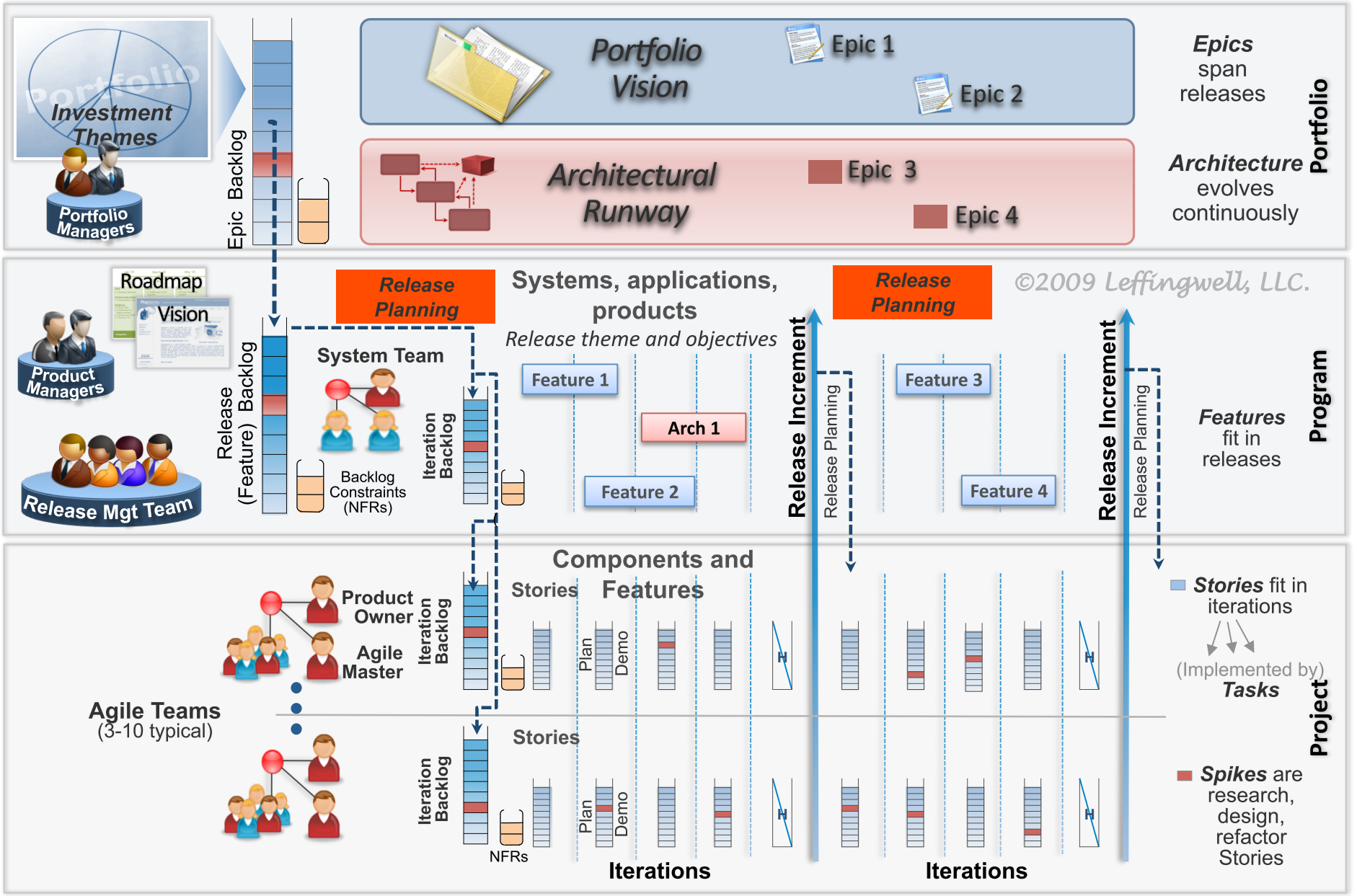
Synchronized Agile Release Train



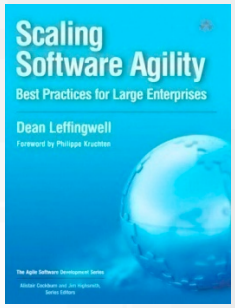


The Agile Enterprise Big Picture

For discussion, see www.scalingsoftwareagility.wordpress.com



Rolling Wave Release Planning Drives the Train



9-10 Business Context

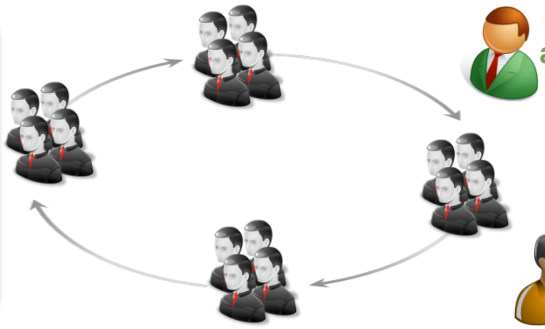
- State of the business
- Objectives for upcoming periods

10-11 Product Vision

PMs

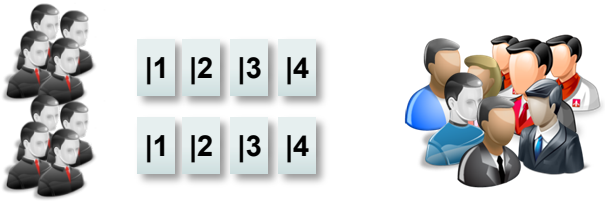
- Objectives for release
- Prioritized feature set

11-12 / **12-1** / **1-2** Team Breakouts



- Teams plan stories for iterations
- Work out dependencies
- Architects and PMs, POs circulate

2-3 / **3-4** Draft Release Plan Review

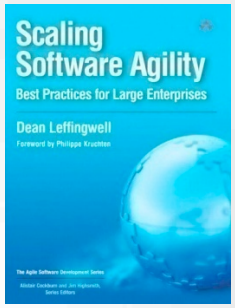


- Each team presents plans to group
- Issues/impediments noted

4-6 Problem Solving / Scope Management

- Issues/impediments assigned
- Release commitment vote?

Rolling Wave Release Planning Day 2



9-10 Revise Objectives?

Eng mgrs/PMs

- Objectives for release
- Prioritized feature set

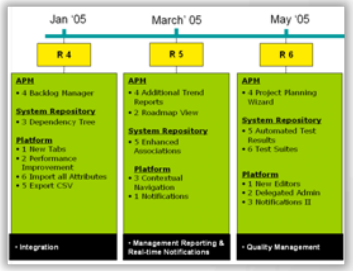
10-11

11-12 Plan/Re-plan as necessary

Dev teams

Product Managers

- What did we learn?
- Update Product Roadmap



12-1

1-2 Final Plan Review

|1 |2 |3 |4

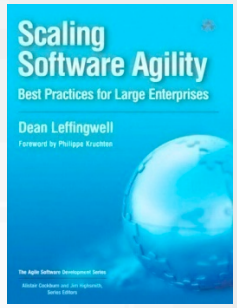
|1 |2 |3 |4

2-3 Commitment

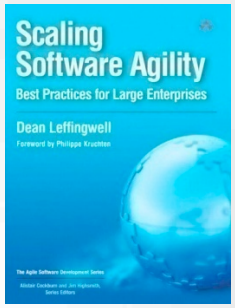
Eng mgrs

- All Issues/impediments assigned
- Release commitment vote

Release Commitment

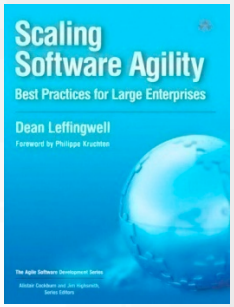


4. Managing Highly Distributed Development



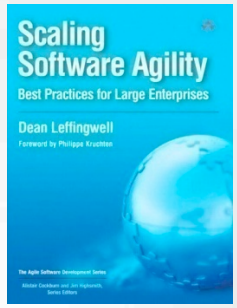
- Co-locate team often – at least at Release Planning
- Establish core hours, with overlap required
- Apply high cohesion and low coupling to sites (organize and reorganize around features/components)
- Don't let anyone go dark - apply daily Integration Scrums
- Establish a single global instance of project assets
- Invest in tools that support distributed, but shared view of status

5. Changing the Organization



- Transition as a Project
- “All in” or Incremental Rollout
- Eliminating Impediments
- Moving to Agile Portfolio Management

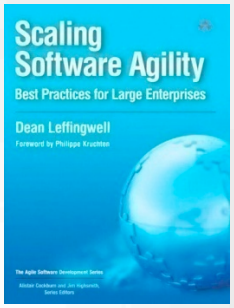
Transition as a Project



- Establish an Agile Enterprise Transition Team
  - Drives the enterprise vision and facilitates implementation
  - Cross-functional involvement
  - Cross-level involvement
  - Executive leadership
- Create a transition backlog
- Run project in iterations
  - Commit to weekly iteration goals
  - Meet at least weekly
  - Report to other executive stakeholders
  - Experience agile project management

- Executive sponsors
- Cross functional

All-In or Incremental?



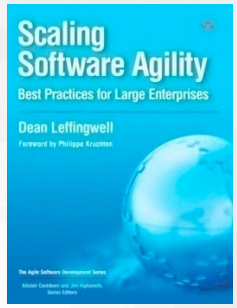
All-in

Advantages	Disadvantages
▸ Failure not an option ▸ All hands on deck ▸ Unified software practices ▸ Enterprise benefits achieved most quickly	▸ Enterprise disruption ▸ Risk of larger scale failure ▸ Risk of organizational buy-in ▸ Training and education resource demands

Incremental

Advantages	Disadvantages
▸ Minimizes adoption risk ▸ More modest training resources ▸ Develop successful organizational patterns ▸ Develop internal mentors	▸ Failure is an option ▸ Dual software processes ▸ Continuously re-factoring process guidance ▸ Delayed enterprise benefits

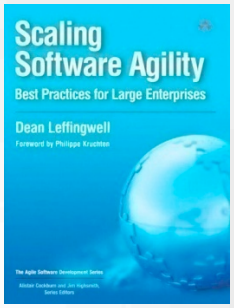
Eliminating Impediments



- Existing rules demand adherence to document-driven, waterfall processes and artifacts
- Software test/ system test not integrated, responsive
- Inadequate build and support infrastructure
- Organization rewards individual over team behavior
- Teams not co-located to maximum extent feasible
- Teams not truly empowered
- Other functions - sales, marketing, customer not supportive of increased delivery pace
- Legacy thinking - Management expectations for fixed-price, fixed-time, fixed-function delivery

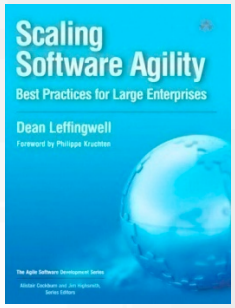
Moving to Agile Portfolio Management

Changing Legacy Mindsets



	Investment Funding	Change Management	Governance and Oversight
From:	"widget engineering" "order taker mentality"	"Maximize utilization" "Get it done"	"Control through milestones/data" "plan out a full year of projects"
To:	Epic based portfolio planning Intense development collaboration	Fixed resources short term only Team commitments Adjust priorities quarterly	Control through empirical release increments Rolling wave release planning

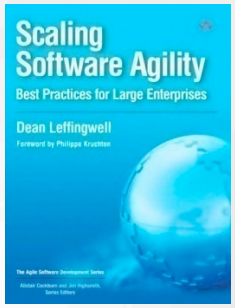
6. Impact on Customers and Operations

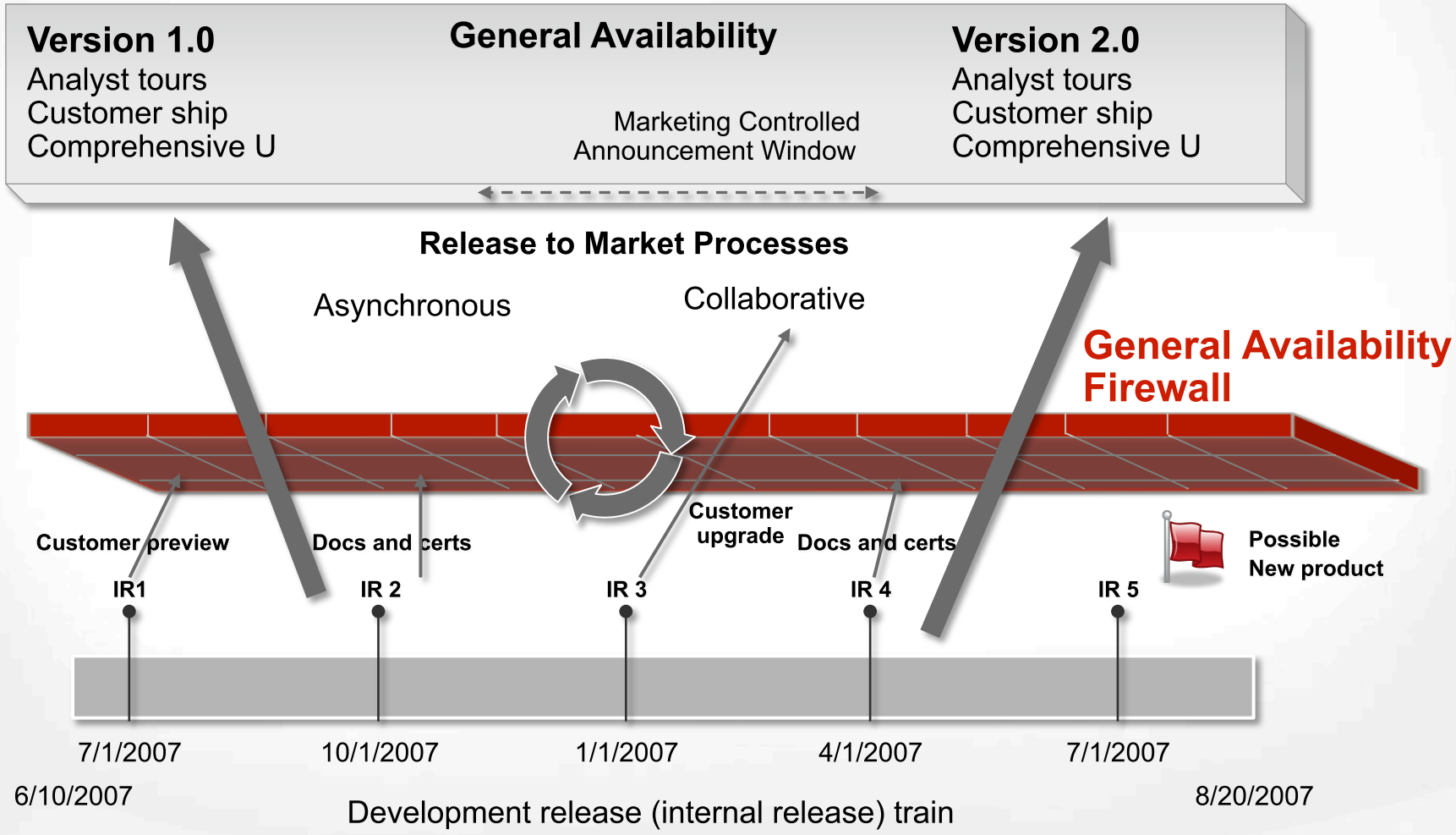


More frequent releases challenge:

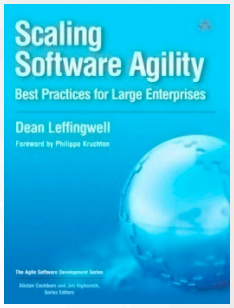
- Customers
- Suppliers
- Marketing and Sales
- Support
- Documentation, certification, localization

Solution: Separation of Concerns





7. Measuring Business Performance

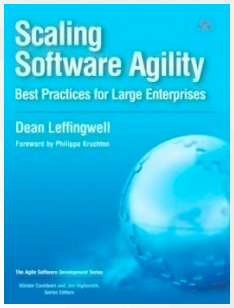


The primary metric for agile is whether or not working software actually exists, and is demonstrably suitable for its intended purpose.

This is determined empirically, by demonstration, at the end of every single iteration.

All other measures are secondary ….. (but not useless)

Process Self-Assessment Metrics



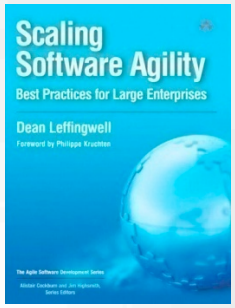
Software Agility Team Self
Backlog prioritized and ranked by busi
Backlog estimated at gross level
Product owner defines acceptance crit
Product owner and stakeholders partic release planning
Product owner and stakeholders partic release review
Product owner collaboration with team
Stories sufficiently elaborated prior to plannir
Total Product Ownership Score

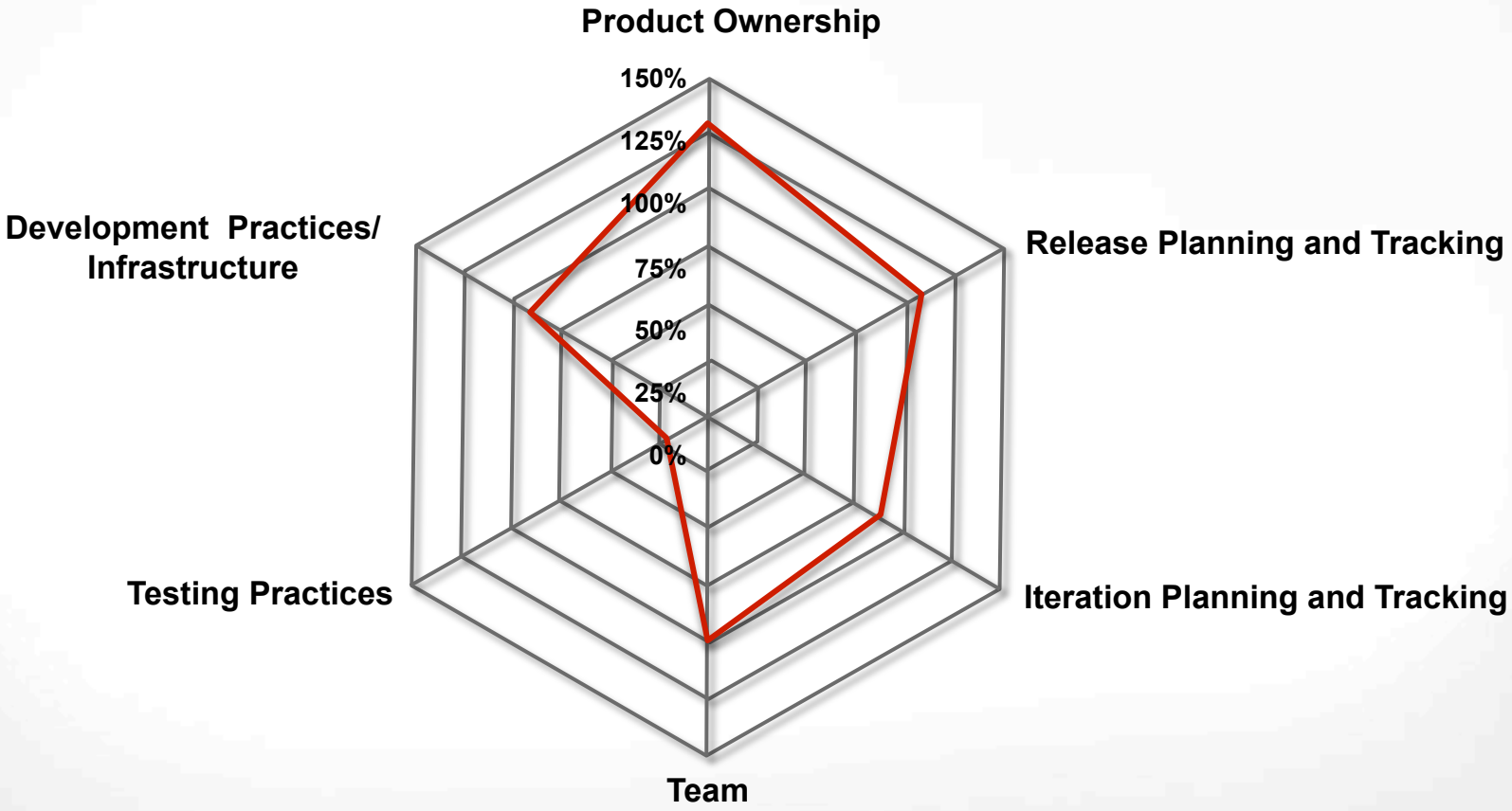
Iteration progress tracked by task to do (burn-down chart) and card acceptance (velocity)
Work is not added by the product owner during the iteration
Team completes and product owner accepts the iteration
Iterations are of a consistent fixed length
Iterations are no more than 4 weeks in length
Iteration review meeting attended and effective
Team inspects and adapts (continuous improvement) the iteration plan
Total Iteration Planning and Tracking Score

release date
Release review meeting attended
Team inspects and adapts (continu plan
Team meets its commitments to release
Total Release Planning and Tracking Score

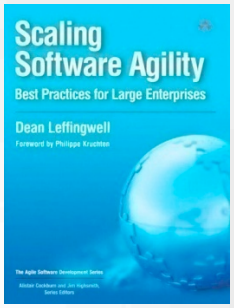
Unit tests are written before development
Acceptance tests are written before development
100% automated unit test coverage
Automated acceptance tests
Total "Testing" Practices Score

Team Agility Assessment Radar Chart



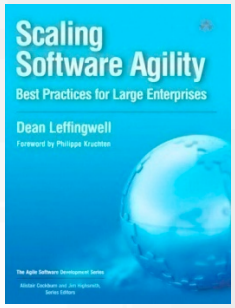


Watch for these Anti-patterns…



- 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 potentially 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

Summary



Agile Teams

1. The Define/Build/Test Team
2. Mastering the Iteration
3. Two-level Planning and Tracking
4. Smaller, More Frequent Releases
5. Concurrent Testing
6. Continuous Integration
7. Regular Reflection and Adaptation

Agile Enterprise

1. Intentional Architecture
2. Lean Requirements at Scale
3. Systems of Systems and the Agile Release Train
4. Managing Highly Distributed Development
5. Changing the Organization
6. Impact on Customers and Operations
7. Measuring Business Performance