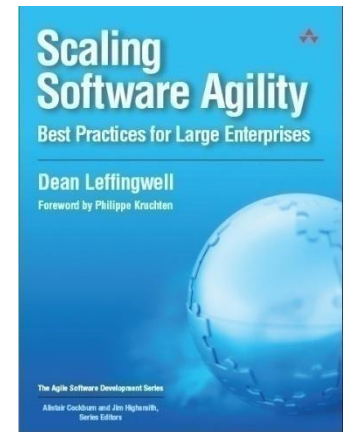
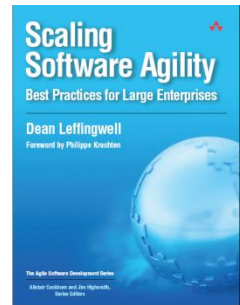


Scaling Software Agility: Best Practices for Large Enterprises

By Dean Leffingwell

Denver SQUAD
Oct 10, 2007

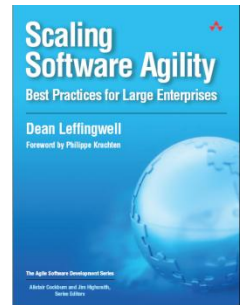




If you accept the premise that market needs change faster than the software industry's traditional ability to develop solutions, you're left with the question "what can we do about it?" For me the answer is Agile.

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

Approaching Agile 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.

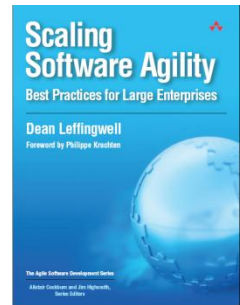
So by constant improvement one can rise to the higher level of practice and knowledge.

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

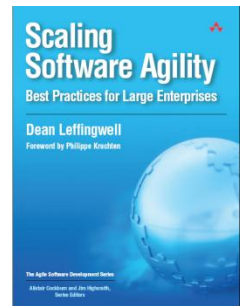
Fujiio Sho, President, Toyota

Agenda

- **Why Enterprise Agility?**
- **What IS Software Agility?**
- **What's Different About Agile?**
- **Achieving Enterprise Agility**
 - **Seven Agile Team Practices that Scale**
 - **Seven Enterprise Practices**

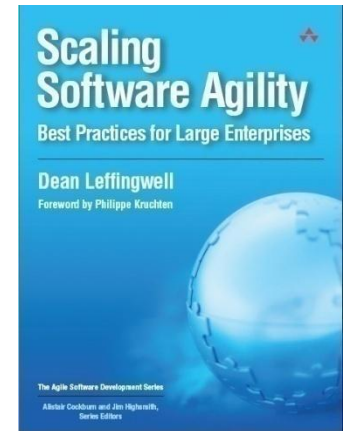


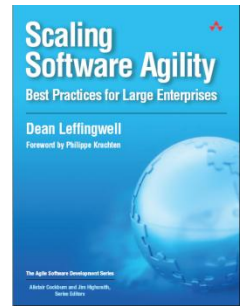
Dean Leffingwell's Background



- Independent software business advisor and software enterprise coach
- Former founder/CEO of Requisite, Inc., makers of RequisitePro requirements management tool
- Former SVP of Rational Software. Responsible for the Rational Unified Process and promulgation of UML
- Methodologist/advisor to Rally Software Development Corporation
- Author of “Scaling Software Agility: Best Practices for Large Enterprises”, Addison-Wesley 2007
- Author of “Managing Software Requirements: A Use-Case Approach”, Addison-Wesley 2003

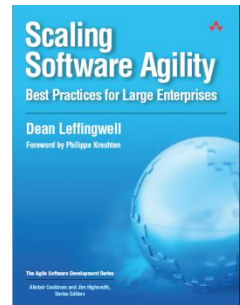
Why Enterprise Agile?





- Promote rapid delivery of value to customers
- Provide timely and regular solution visibility to customers *and* other stakeholders
- Delivers increases in quality, productivity, morale
- Increases Return on Investment for software development

The BMC Story



- Led by (then newly hired) Israel Gat, BMC VP of Infrastructure Development
- Executive Coaches: Dean Leffingwell and Ryan Martens
- Team Coaches and Agile Tooling: Rally Software Development

Press Release

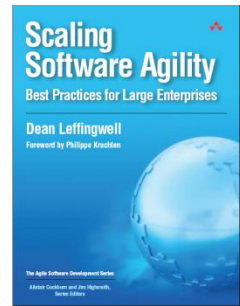
Source: QSM Associates

Agile Excels at BMC, Programming Technique Provides Customers Higher Quality Products in Record Time

Monday September 10, 8:08 am ET

SLIM Analysis Shows Agile Development Can Bring Positive Results for both Developers and the Bottom Line

BMC Results, Cutter Consortium

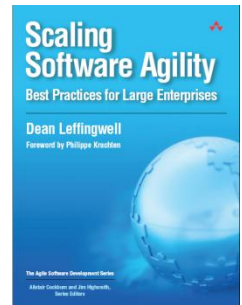


Press Release Sept 10, 2007

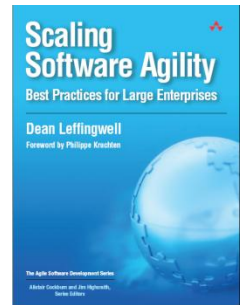
- “remarkable levels of time-to-market and quality”
- “produce large scale enterprise software in 4-5 months, compared to typical one year”
- "exceptional time-to-market without sacrificing quality”
- "Especially noteworthy - BMC 'Secret Sauce' enables process to succeed in spite of geographically dispersed 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

Agile Delivers Higher



Quality



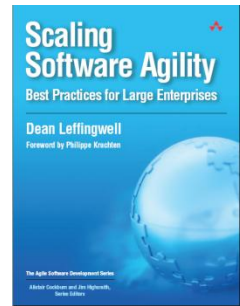
- *Our implementation of agile practices . . . helps us find bugs earlier, helps us achieve higher quality, and helps us work well with SW QA.*

—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



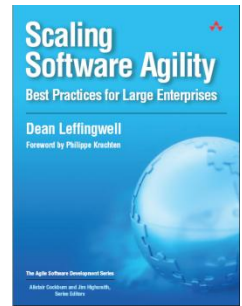
- *Last year, we had 22 releases across 3 major product lines, and not a one of them was late. We support hundreds of Fortune 1000 enterprises with a single person dedicated to support -- the software is that solid.*

—Andre Durand, CEO, Ping Identity Corp.

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

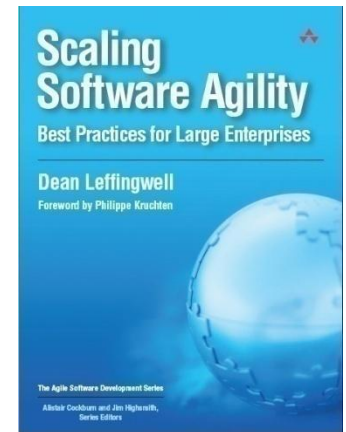
—BMC Software

Morale

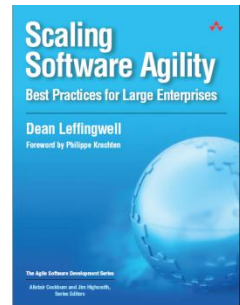


- *Development teams are more engaged, empowered and highly supportive of the new development process.*
—BMC Software
- *Our implementation of agile practices . . . (1) makes the work more enjoyable, (2) helps us work together, and (3) is empowering.*
—Jon Spence, Medtronic

What Is Software Agility?

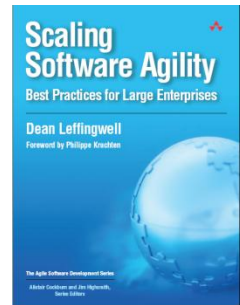


Agile Methods



- Adaptive Software Development (Highsmith)
 - Crystal Methods (Cockburn)
 - Dynamic System Development Method (Faulkner et al)
 - Feature Driven Development (Coad & DeLuca)
 - Lean Software Development (Poppendiecks)
 - SCRUM (Schwaber, Beebe, Sutherland)
 - Extreme Programming (XP) (Beck, Gamma)
-
- Iterative/Agile – RUP and Open Unified Process (Jacobson, Kruchten, Royce, Kroll)

Agile Principles—*The Agile Manifesto*



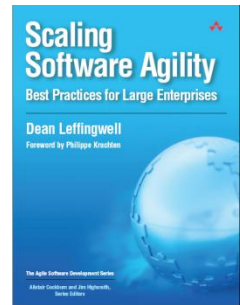
“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

<http://www.agilemanifesto.org/>

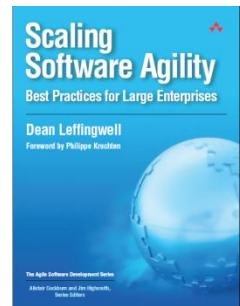
Agile Manifesto Principles



- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Working software is the primary measure of progress.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

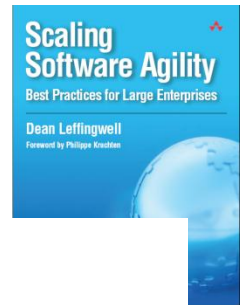
<http://agilemanifesto.org/principles.html>

Principles (continued)

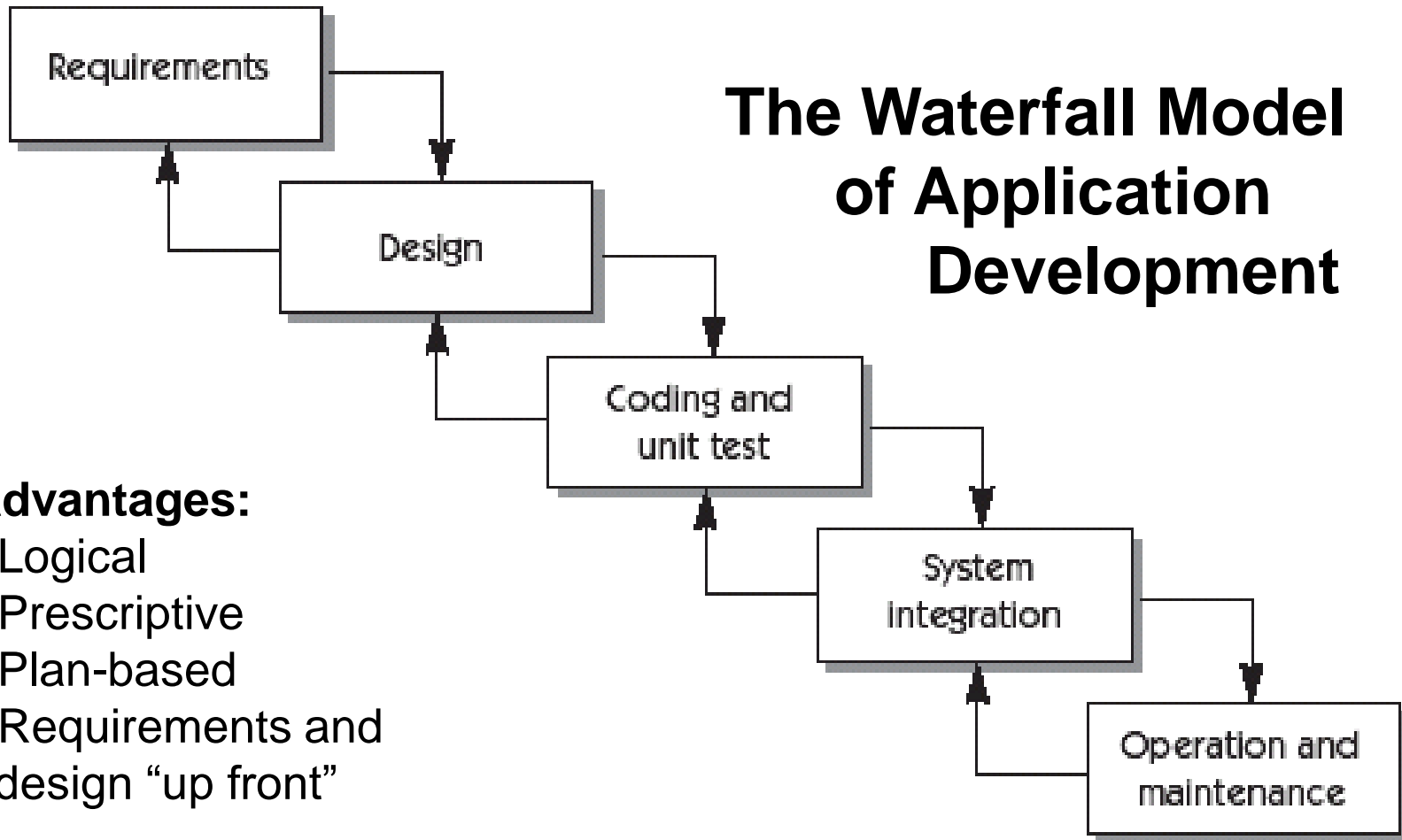


- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Motivation- A Brief Look Back



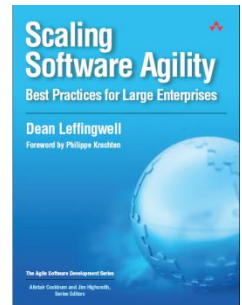
The Waterfall Model of Application Development



Advantages:

- Logical
- Prescriptive
- Plan-based
- Requirements and design “up front”

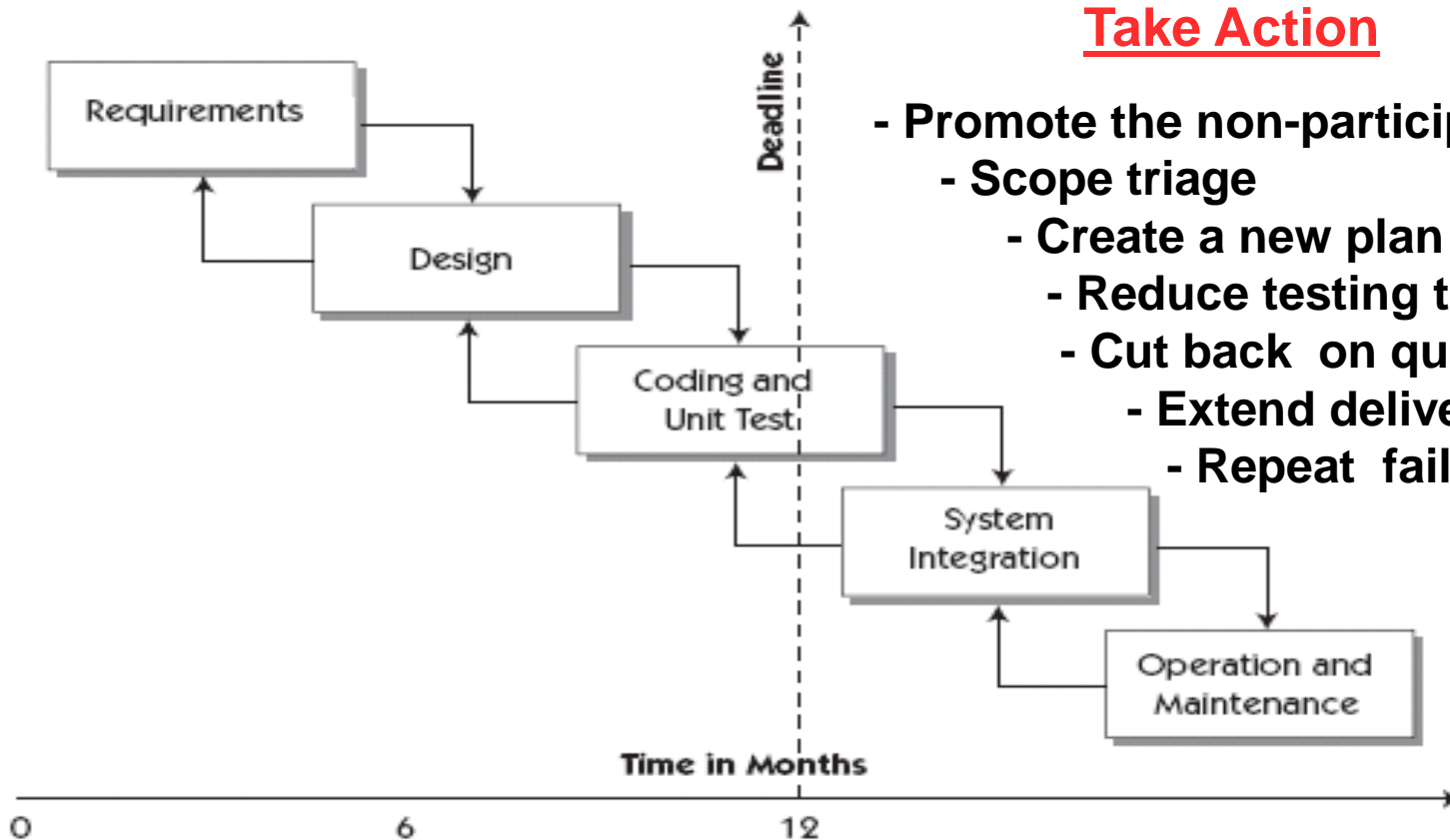
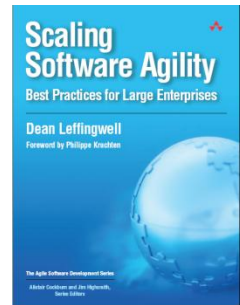
Assumptions Behind the Waterfall Model



1. There exists a reasonably well defined set of requirements, if we only took the time to discover them
2. We will limit change and/or change will be small and manageable
3. System integration is a stage in the lifecycle and we can predict how that will go
4. Software innovation can be done on a predictable schedule and budget

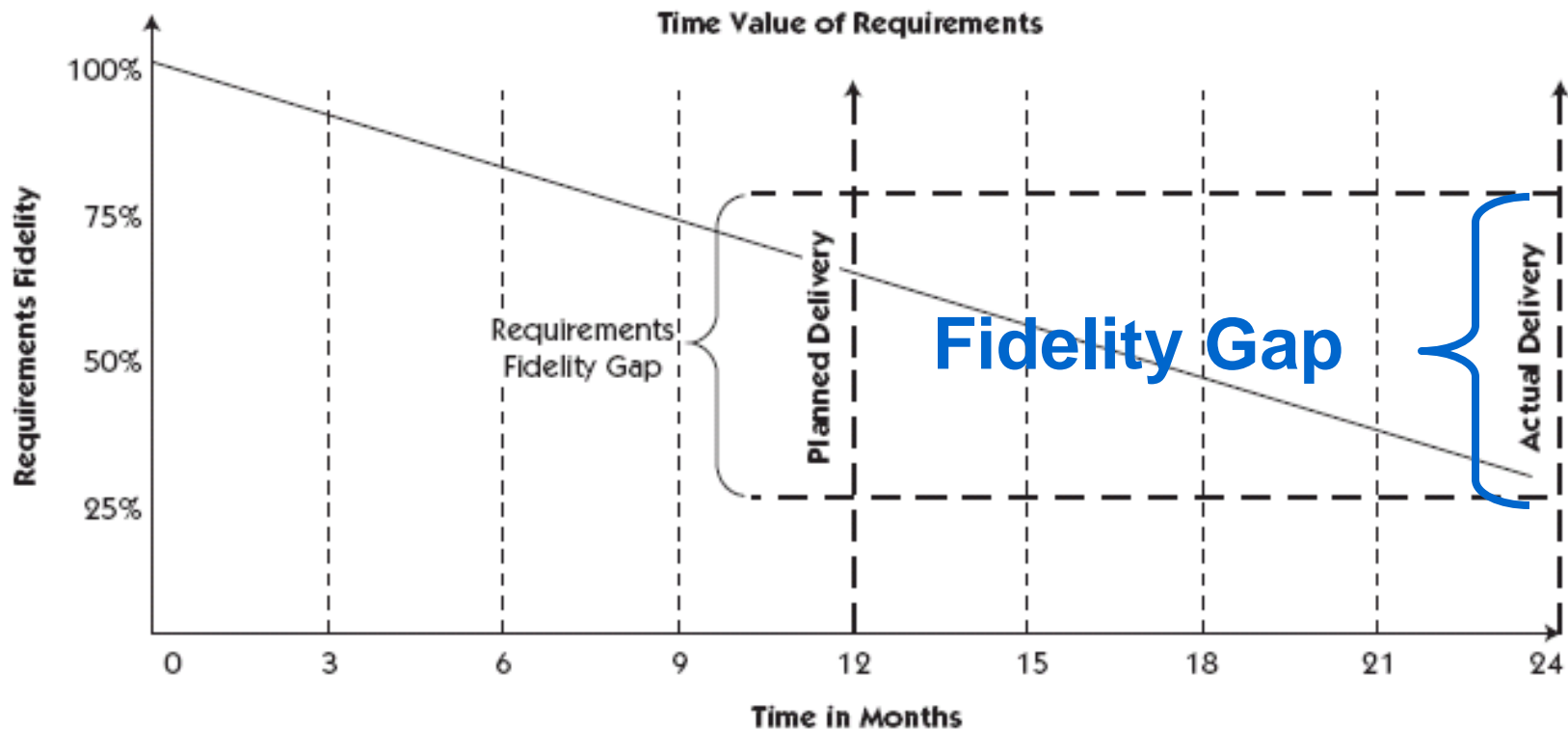
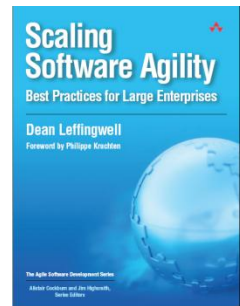
What Really Happens

Most software projects are 50-100% late (Standish Group)
At deadline time, nothing works completely
What do we do now?

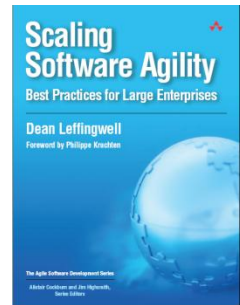


Wait, it gets worse-

The solution we don't even have is a poor fit to current requirements



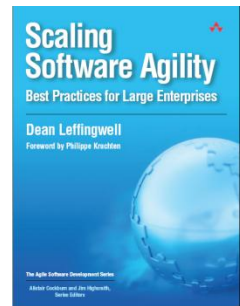
Looking Backwards - Conclusion



- We failed to deliver the application as intended to the customer in the predicted time frame.
- We now understand that the solution that we have in process is off the mark. (requirements decay)
- We likely don't have anything holistic to ship to the hold the customer's confidence.
- We haven't even driven the risk out of the project, because integration is not complete.

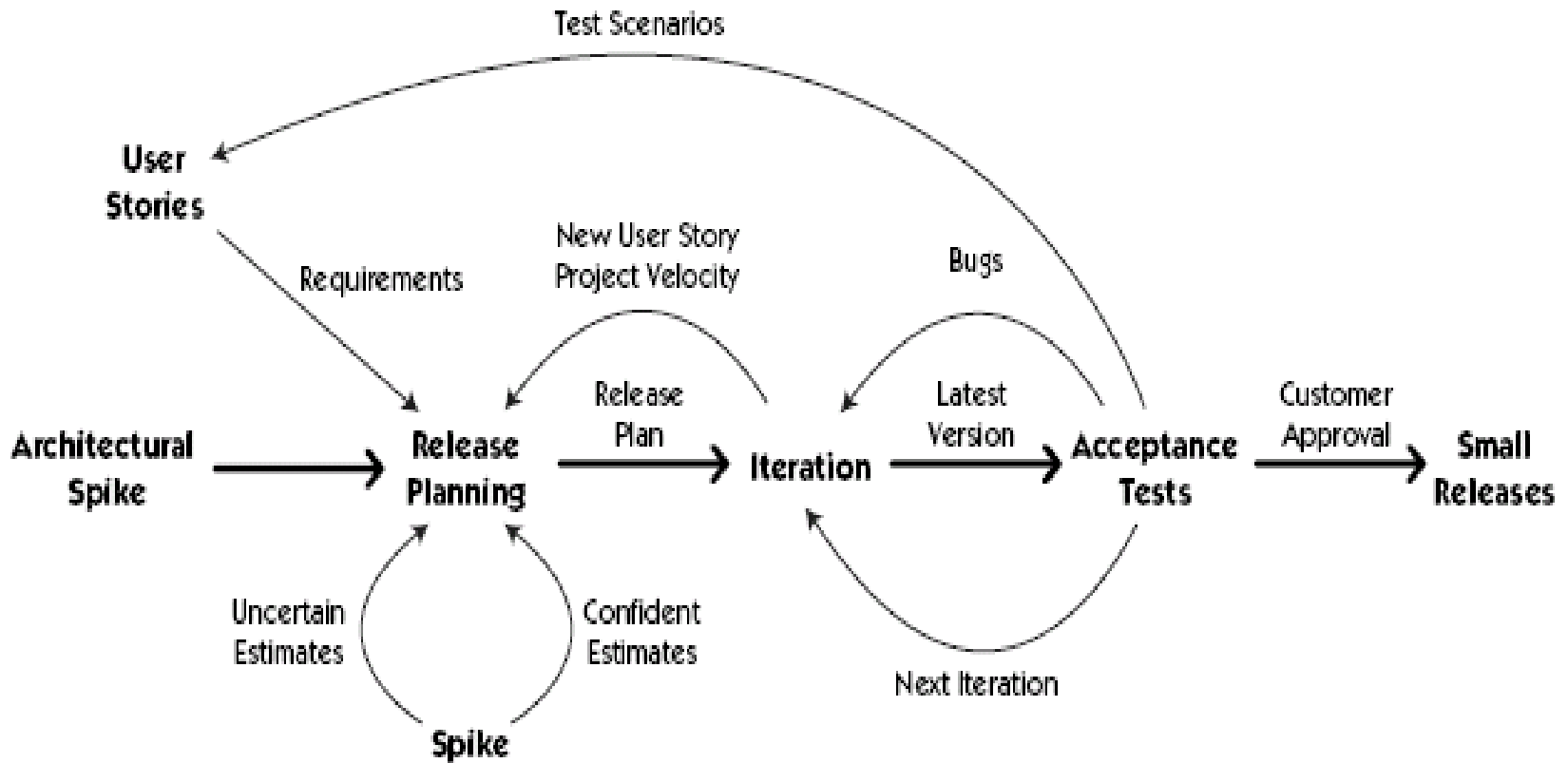
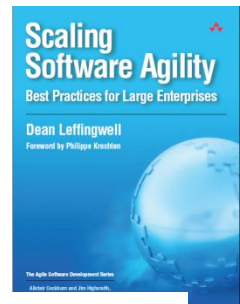
There has to be a better way

Agile Methods – New Assumptions



1. We **do not** assume that we or our customers can fully understand all the requirements up-front.
2. We **do not** assume that change will be small and manageable.
 - We assume change will be constant
 - We deliver in small increments to better track change.
3. We **do** assume that system integration is important and is integral to reducing risk. Therefore:
 - We integrate from the beginning and we integrate continuously.
 - We live under the mantra “the system always runs,” and strive to assure that there is always product available for demonstration and potential shipment.
4. We **do not** assume that we can develop new, state-of-the-art, software projects on a fixed function and schedule basis.
Indeed, we assume cannot.
 - Instead, we assume that we can deliver the most important features to our customer earlier, rather than later, than might have expected.
 - In so doing, we can get immediate feedback whether we are building the right solution.

Popular Agile Methods - XP



Extreme Answers to Serious Questions?

*“If code reviews are good, we’ll review code all the time
(pair programming).*

*If testing is good, everybody will test all the time (unit testing), even the
customers (acceptance testing).*

If design is good, we’ll make it part of everybody’s daily business (refactoring).

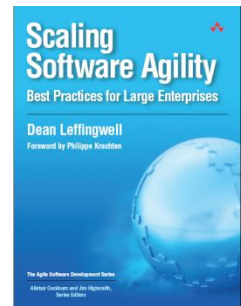
*If simplicity is good, we’ll leave the system with the simplest design that
supports its current functionality
(the simplest thing that could possibly work).*

*If architecture is important, everybody will work at defining and refining
architecture all the time. (emergent architecture)*

*If integration testing is important, we will integrate and test several times a day
(continuous integration).*

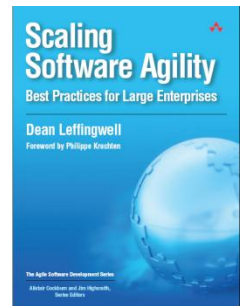
*If short iterations are good, we will make the iterations really, really short—”
(daily build, weekly release)*

Beck [2001]

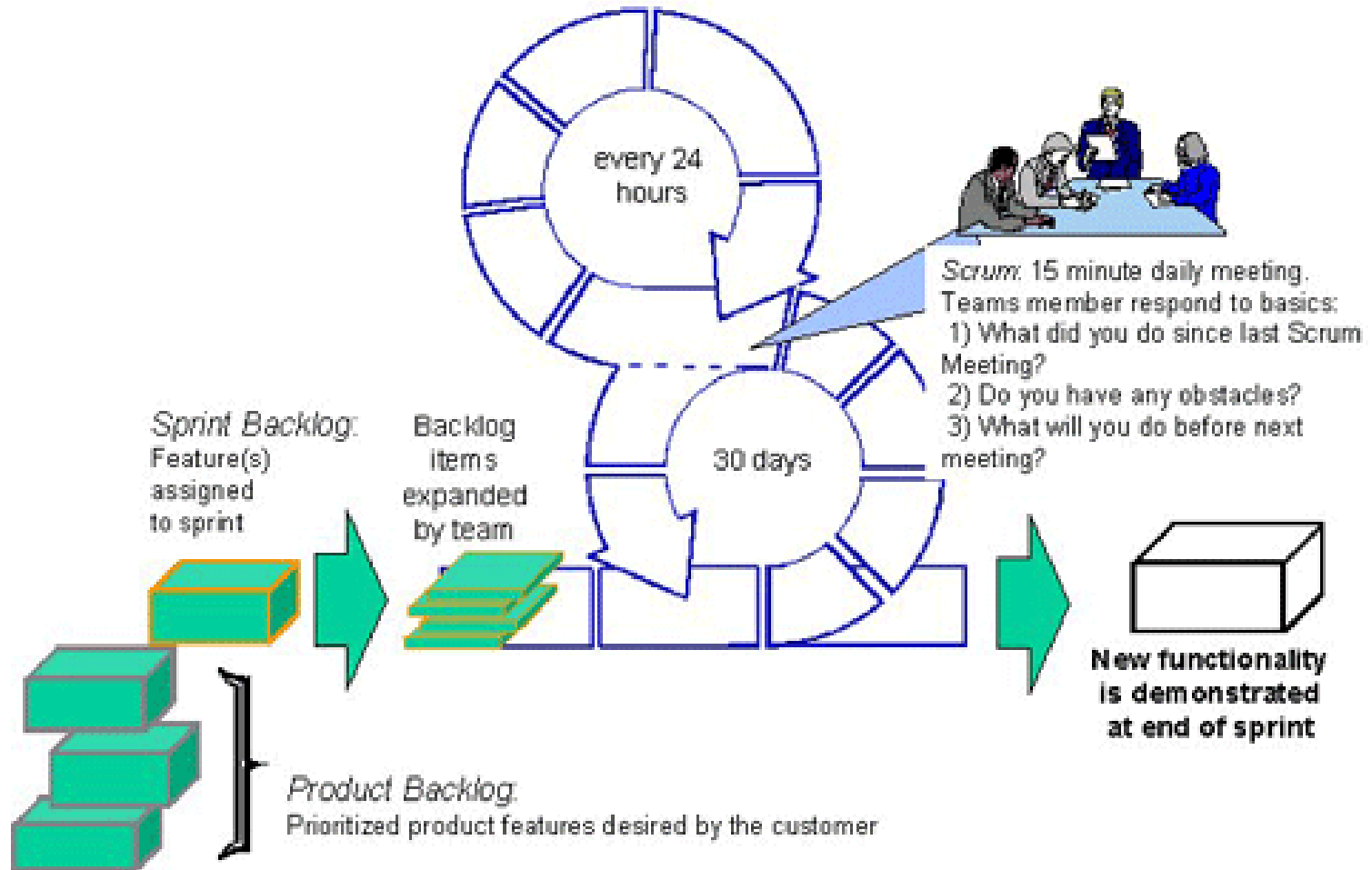
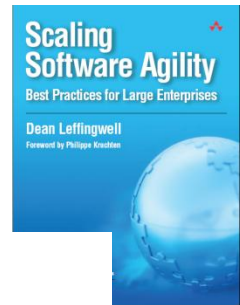


XP Lessons Learned

- Small, co-located teams are (way) more productive
- The customer is part of the team
- Two eyes (pair programming, dev-tester pairs) are better than one
- Nothing works until it is tested
- Agile is enabled by constant refactoring
- Constant refactoring is enabled by good design, good tooling, OO, testing and test automation....

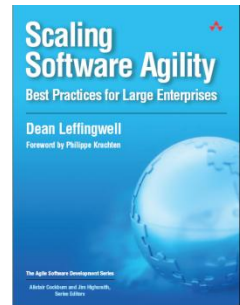


Popular Agile Methods - Scrum

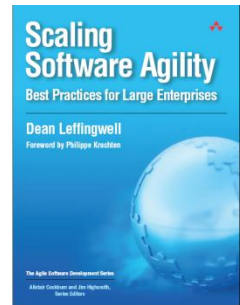


Scrum Philosophy

- Based, in part, on Toyota's product development Process
 - Concurrent Engineering
 - The New, New Product Development Game
- Based on empirical process control
- Lightweight process, just 3 roles
- Harnesses the power (Ba) of the team

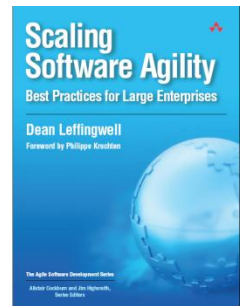


Scrum Practices

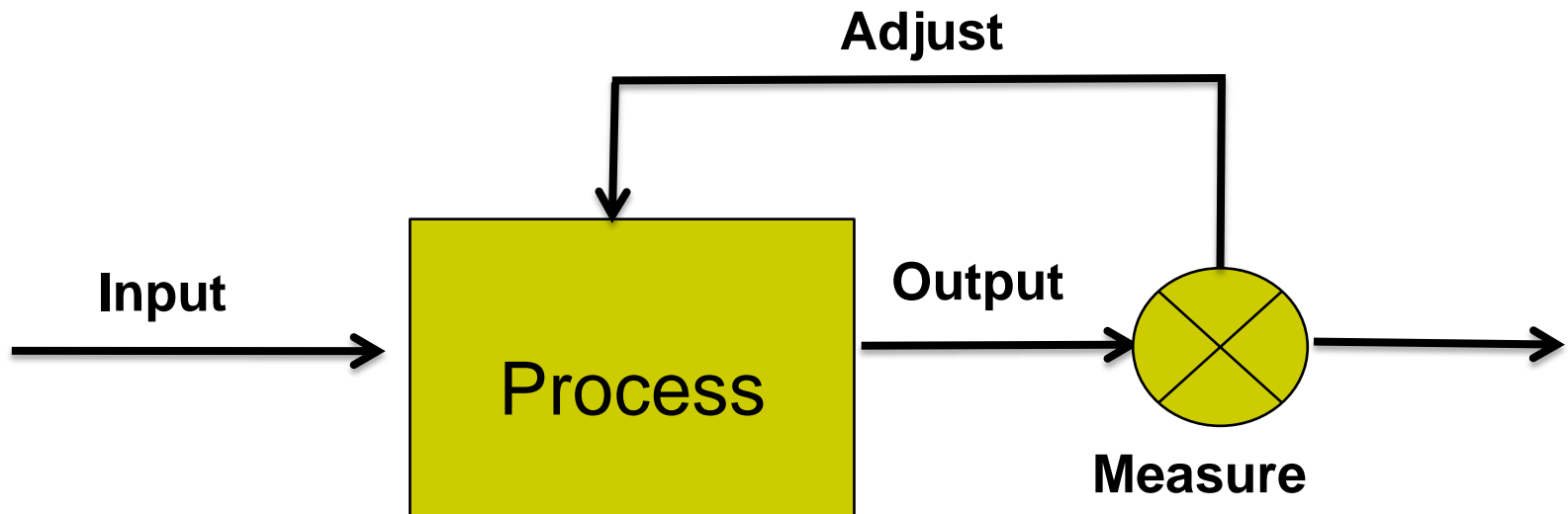


- Collocated, cross-functional teams of eight or fewer develop software in 30 day “Sprints.”
- Each sprint delivers incremental, tested, user value.
- Work within a sprint is fixed.
- The Scrum Master mentors and manages the self-organizing and self managing teams
- All work to be done is carried as Product Backlog, developed, managed, and prioritized by the Product Owner, who is integral to the team
- A daily 15-minute stand-up meeting, or “Daily Scrum,” is a primary communication method.
- Everything is time boxed.
- Requirements, architecture, and design emerge over the course of the project.

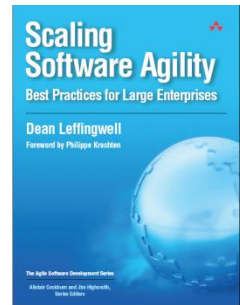
Scrum – Lessons Learned



If a process is unpredictable or too complicated for the **planned**, (predictive) approach, then the **empirical** approach (measure and adapt) is the method of choice.

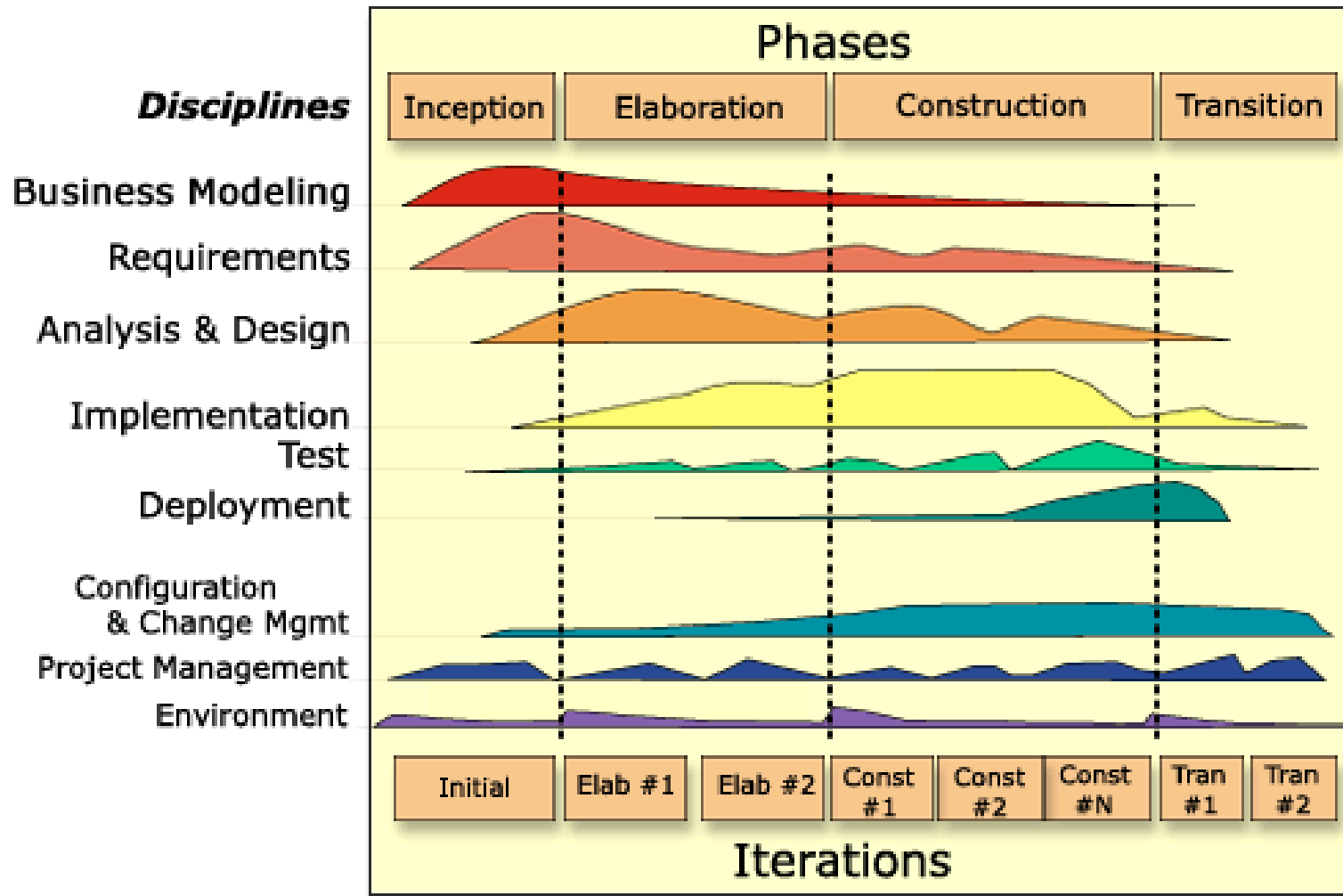


Scrum – Lessons Learned

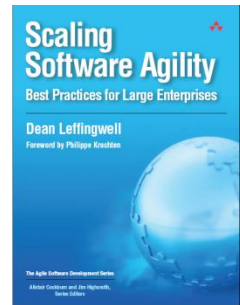


- Unleash “ba”, the energy of the self-organizing team
- Team work is facilitated by a Scrum Master
 - Fosters changed practices
 - Eliminate impediments, reinforces agile discipline
- “backlog” is a simple organizing technique for requirements and priorities
- Measure and communicate every day
- Scrum drives team and organizational change – better prepare

RUP – Iterative and Incremental



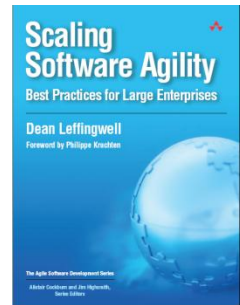
RUP Lessons Learned



- No amount of planning will get it right the first time: Iterate!
- *System* level use cases help understand desired *system* behavior
- For systems of scale, architecture matters
- Modeling is a way to reason about the behavior of a system
- Architecture must be built, not just planned
- There is no one-size-fits-all software process

RUP Leans to Agile with Open UP

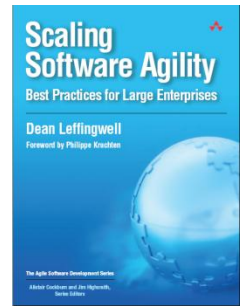
IBM donated the *Basic Unified Process*, a scaled-down RUP, to the Eclipse open source project



Result: OpenUP - a hybrid of agile methods and RUP

- Lighter weight process
 - Still iterative, use case–driven, and architecture–centric, still follows the lifecycle phases of the RUP
- Incorporates ideas from agile methods:
- **Scrum**
 - – Product backlog
 - – Iteration planning, assessments, and daily status meetings
 - – Agile modeling (no big upfront design)
- **XP**
 - – TDD: Tests happen sooner rather than later (test first)
 - – Refactoring
 - – Continuous integration
- **DSDM**
 - Stakeholder collaboration

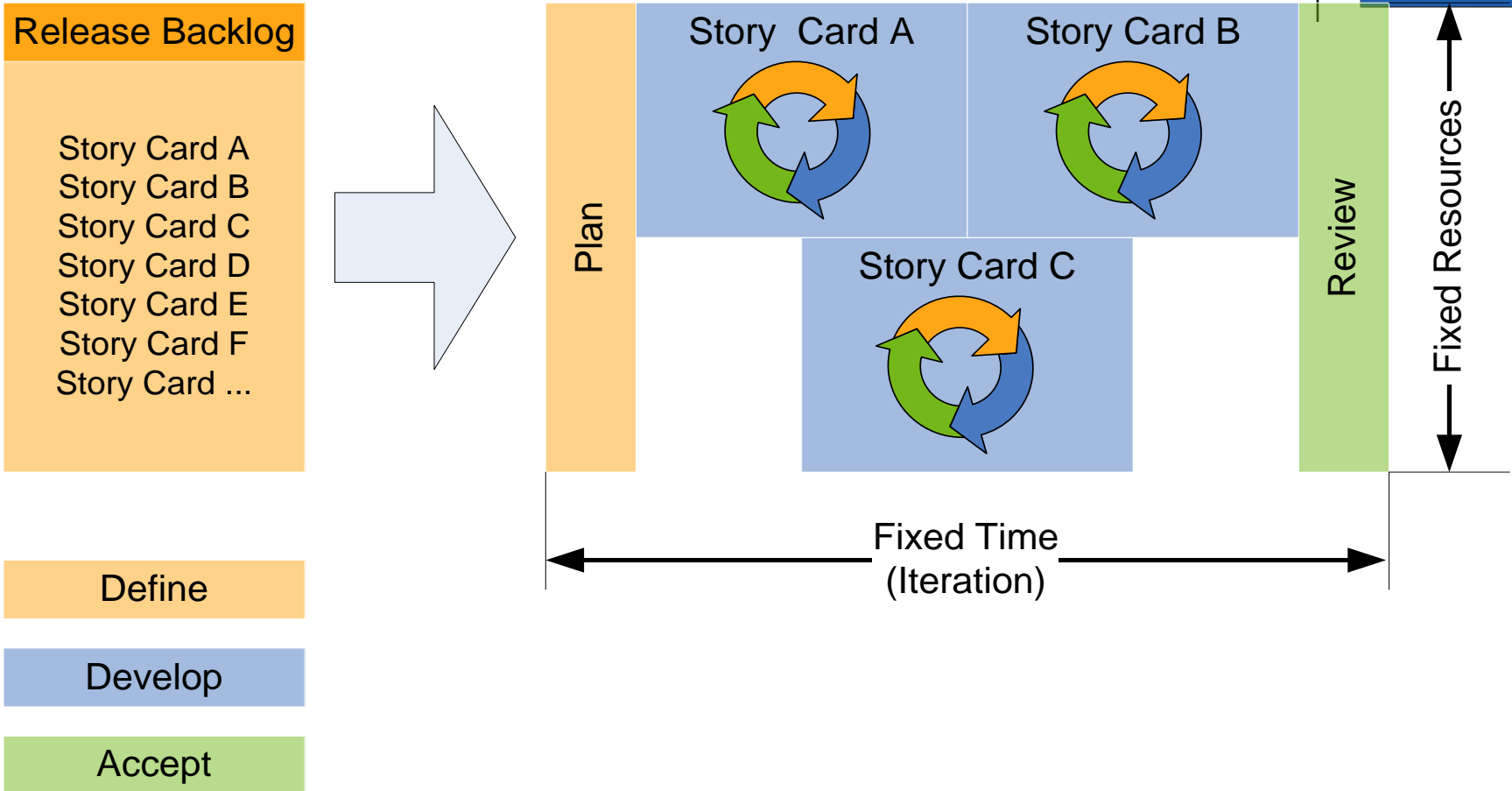
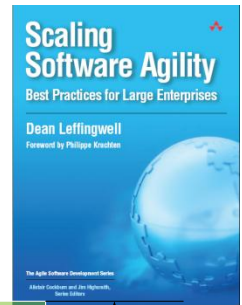
Question



- What do all these agile techniques have in common?
- Answer:
 - Iterations: deliver working tested increments in a short time box

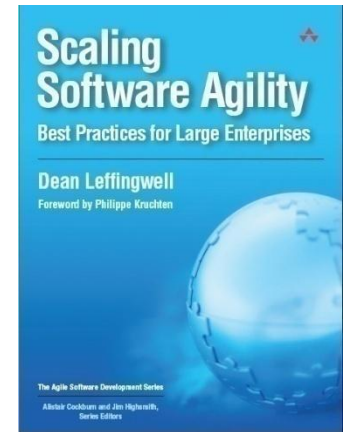
Short iterations - the heartbeat of agility

Agile Iteration

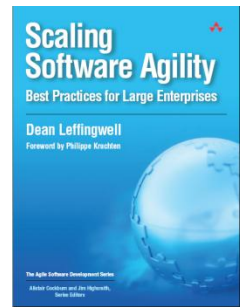


What's Different About Agile?

Conceptually, agile is simple.
Most everything is different.

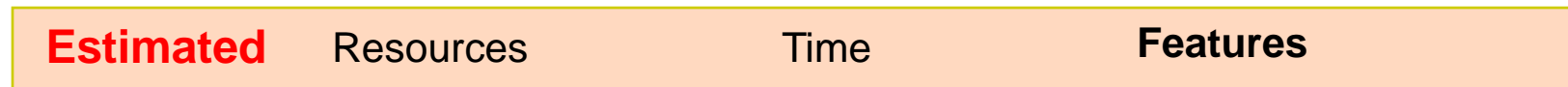
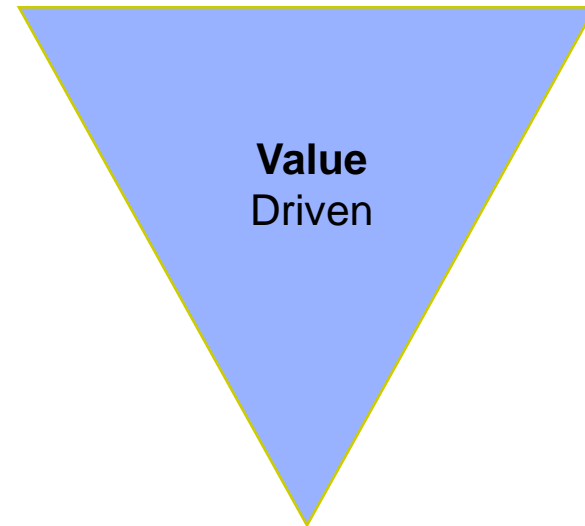
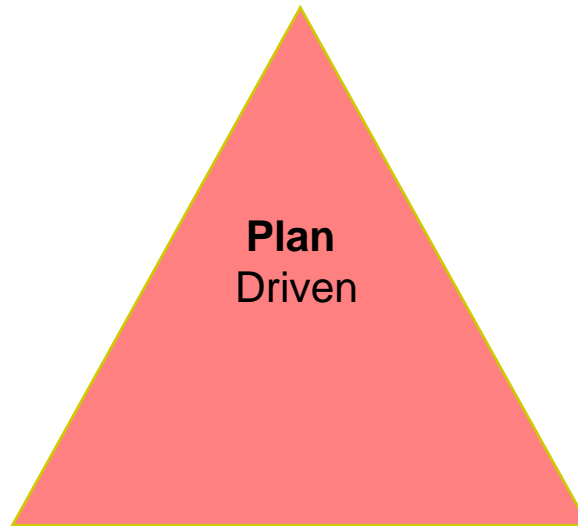
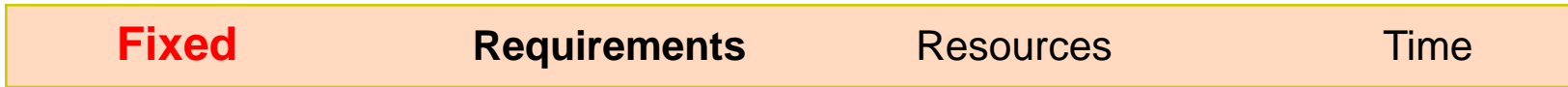


The Agile Paradigm Shift



Waterfall

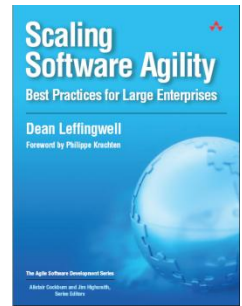
Agile



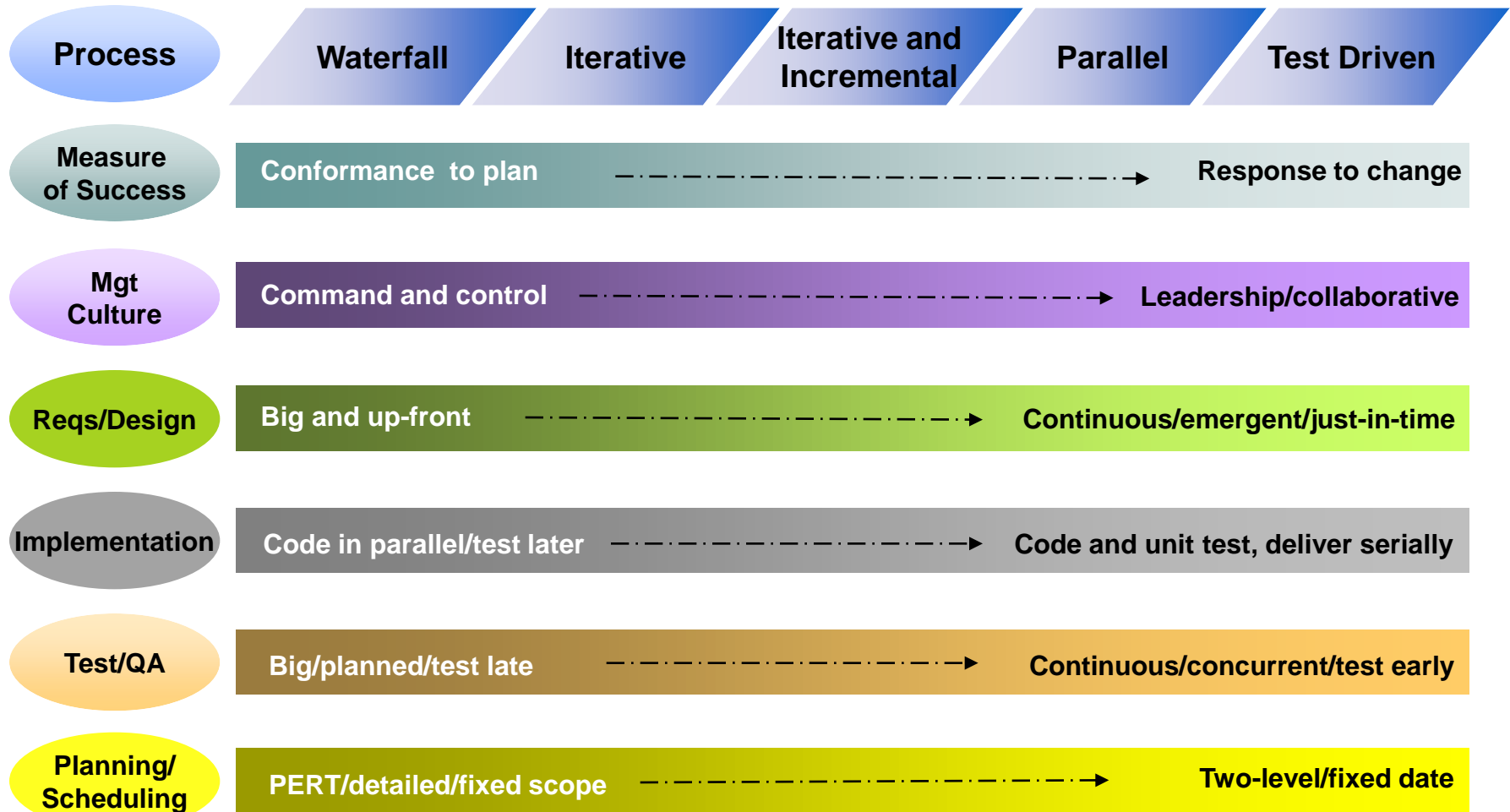
The Plan creates cost/schedule estimates

Release themes & feature intent drive estimates

What Paradigms Are We Breaking?

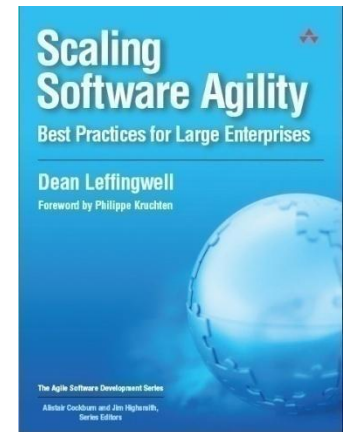


Agile Development

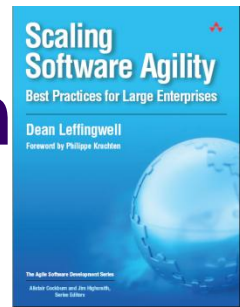


Seven Agile Team Practices That Scale

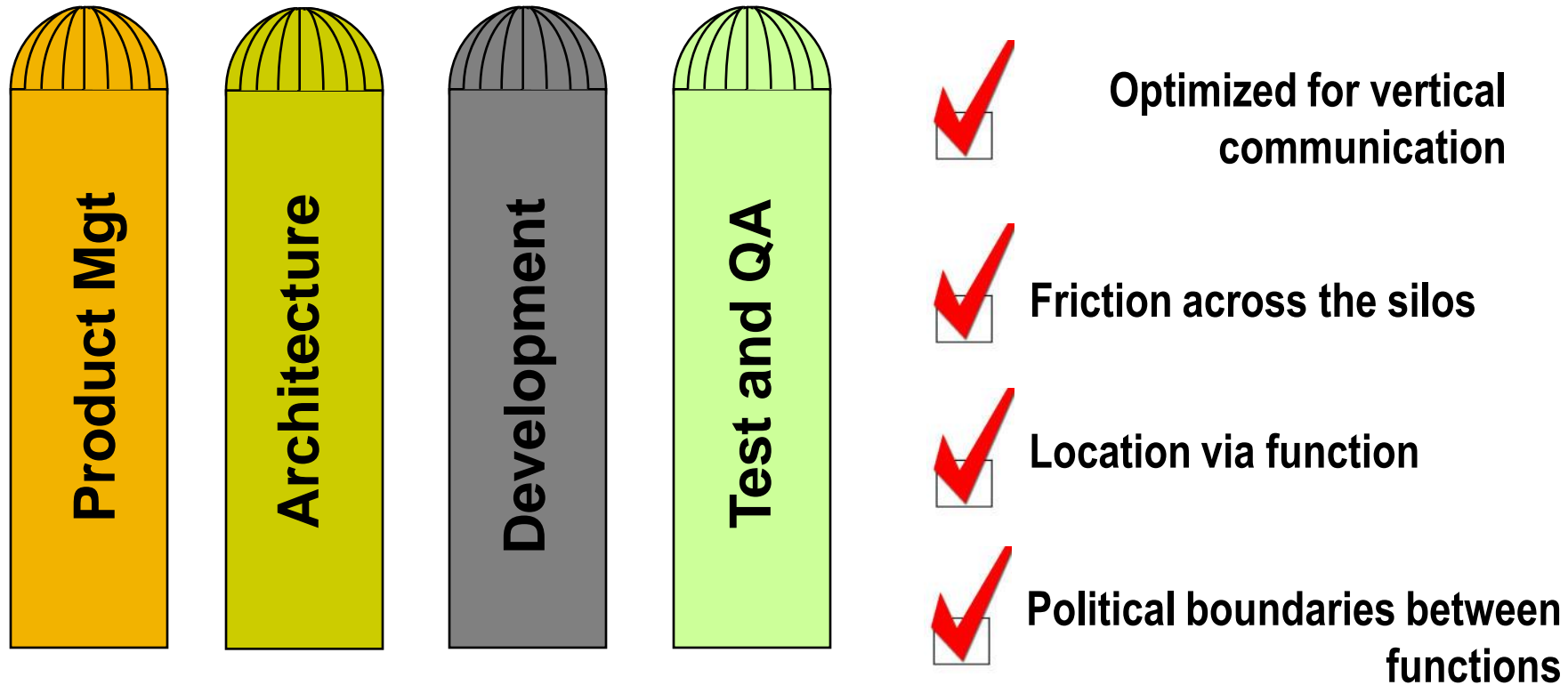
The Define/Build/Test Component Team
Two-level Planning and Tracking
Mastering the Iteration
Smaller, More frequent releases
Concurrent Testing
Continuous Integration
Regular Reflection and Adaptation



1. Define/Build/Test Component Team

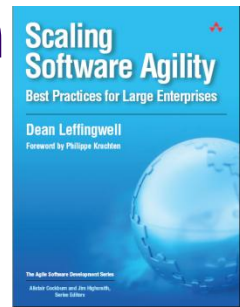


Typical Functional Silos

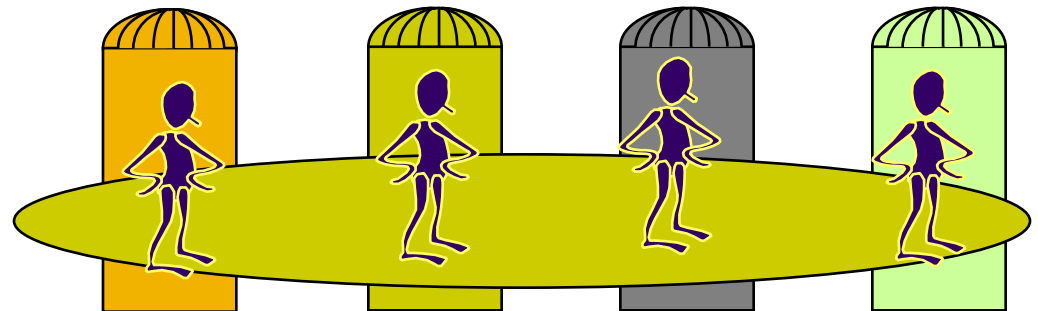


Management Challenge: Connect the Silos

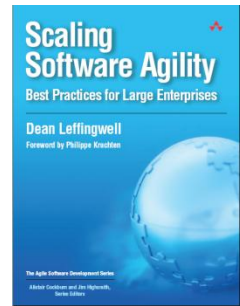
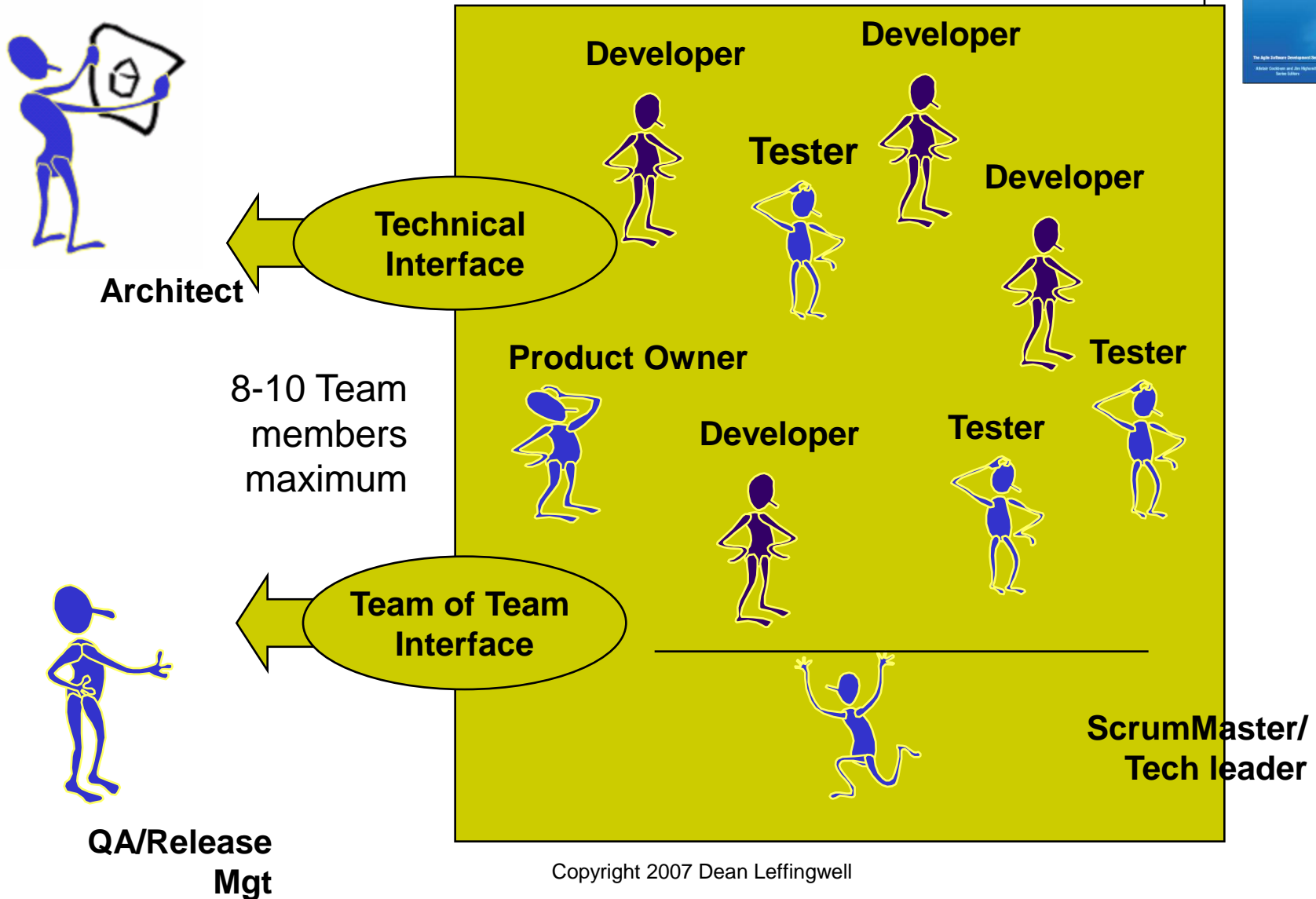
Component Team – The Agile Organization Fractal



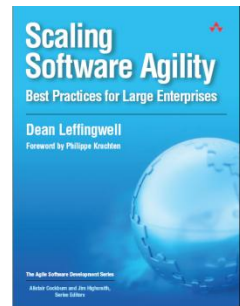
- Agile Fractal – a team pattern that is repeated at larger scales to produce optimum outcomes that cannot be predicted by a plan-based system
- Fractal Teams can be based on
 - Components
 - Subsystems
 - Features
 - Interfaces
 - Products



Agile Team

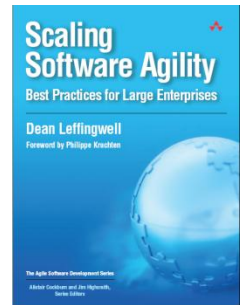
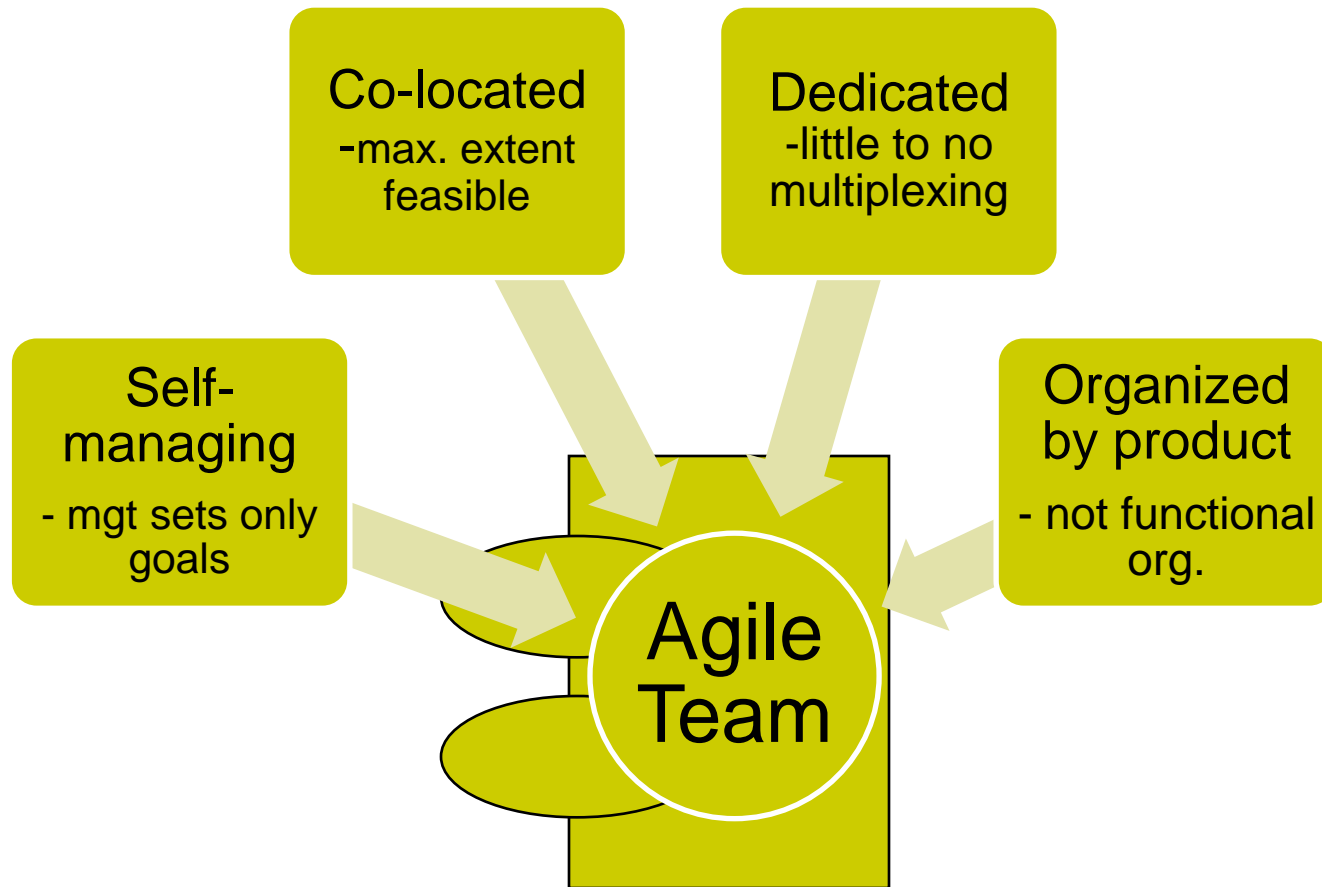


Agile Component Team Rationale

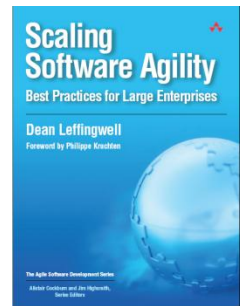


- Communication is optimized *across* disciplines
- Teams have all disciplines necessary
 - define>build>test
 - Bid, elaborate, execute
 - Adjust scope and reprioritize
 - Integrate, test, accept
- The functional and social network optimized for *one* purpose
 - To define, design, build and test a potentially shippable increment of software

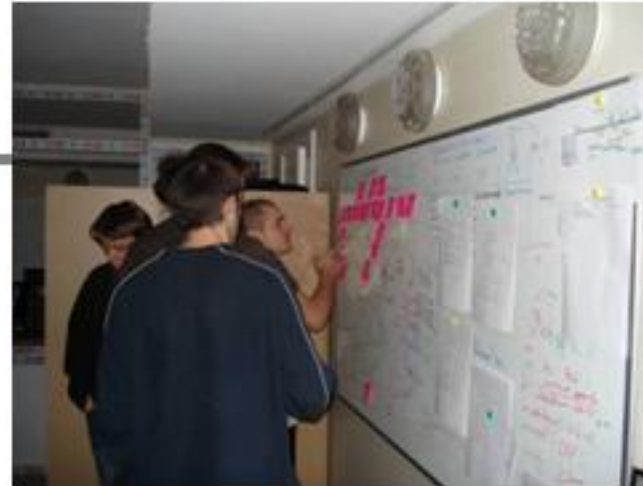
Agile Team Characteristics



Open Environment, Constant Communication



Daily Standup

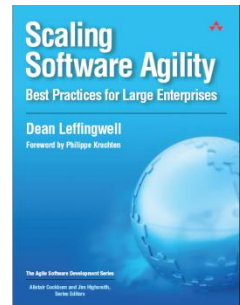


Meet after

Spontaneous Pair Programming

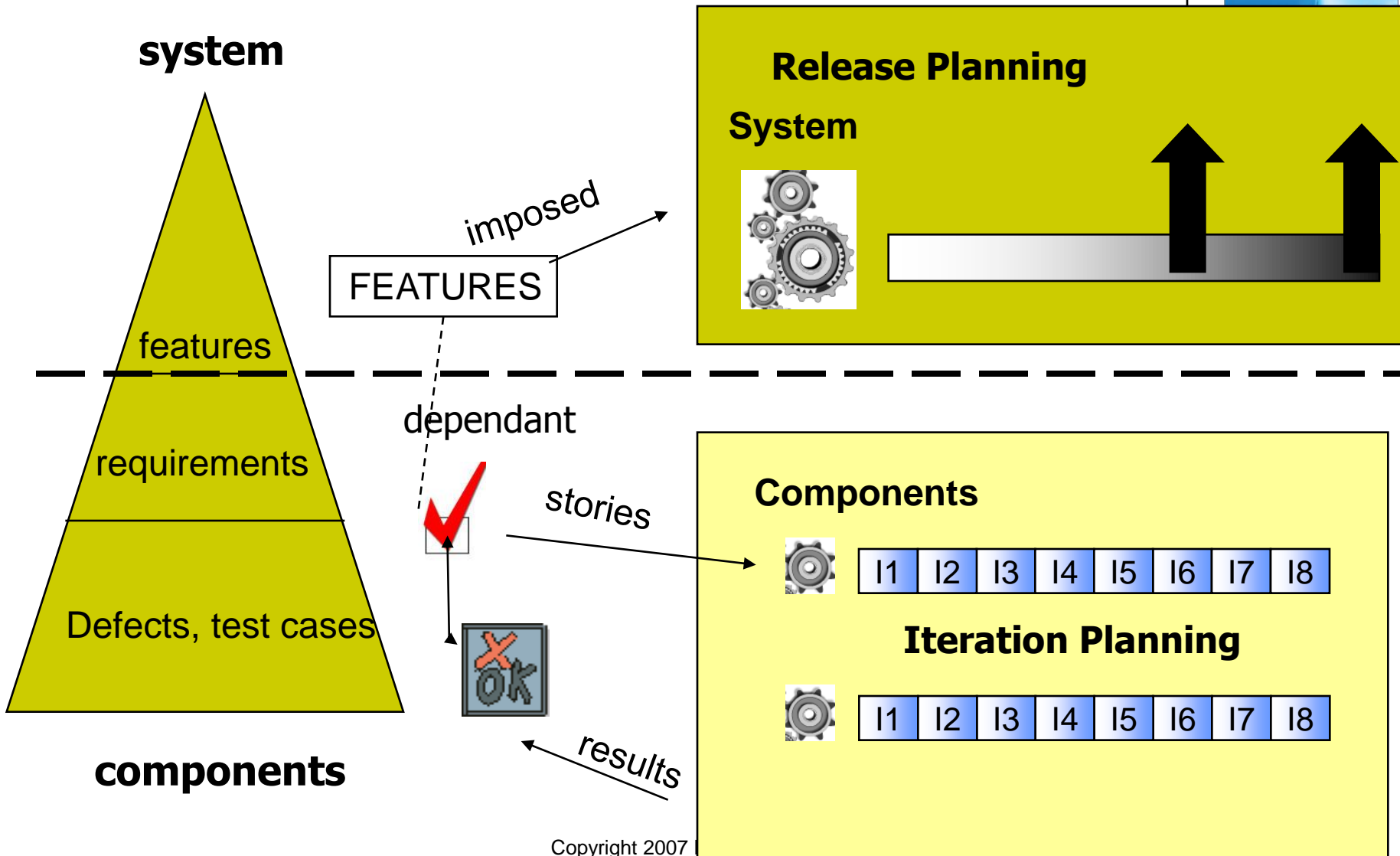


2. Two-Level Planning and Tracking

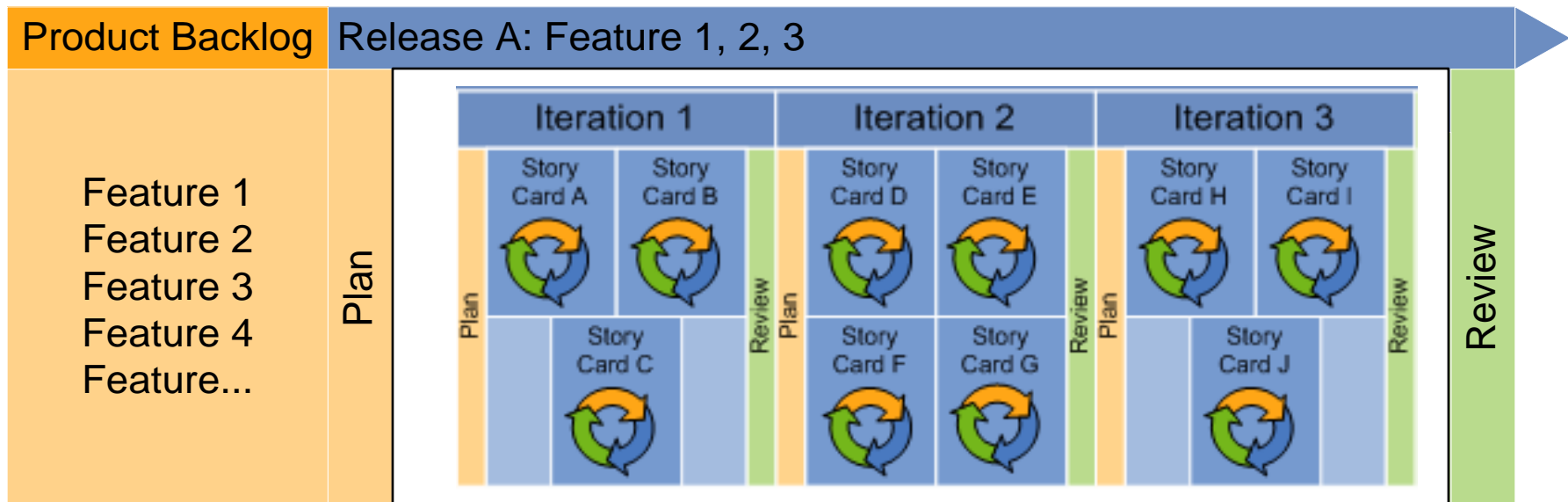
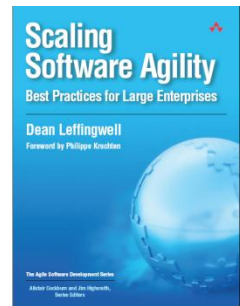


- Release Plan at the System Level
 - Release theme and prioritized feature sets for each release
 - High visibility and confidence near term (Release “next” and “next+1”). Lower visibility longer term
 - Six to nine months planning horizon
- Iteration Plan at the Component Level
 - Define story cards and tasks for next 1-3 iterations
 - Adapt and adjust at each iteration
 - 4-6 weeks planning horizon

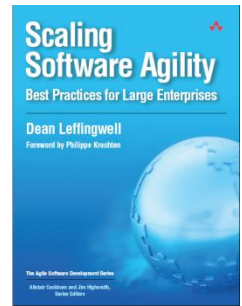
Release and Iteration Planning



Agile Iteration and Release Flow

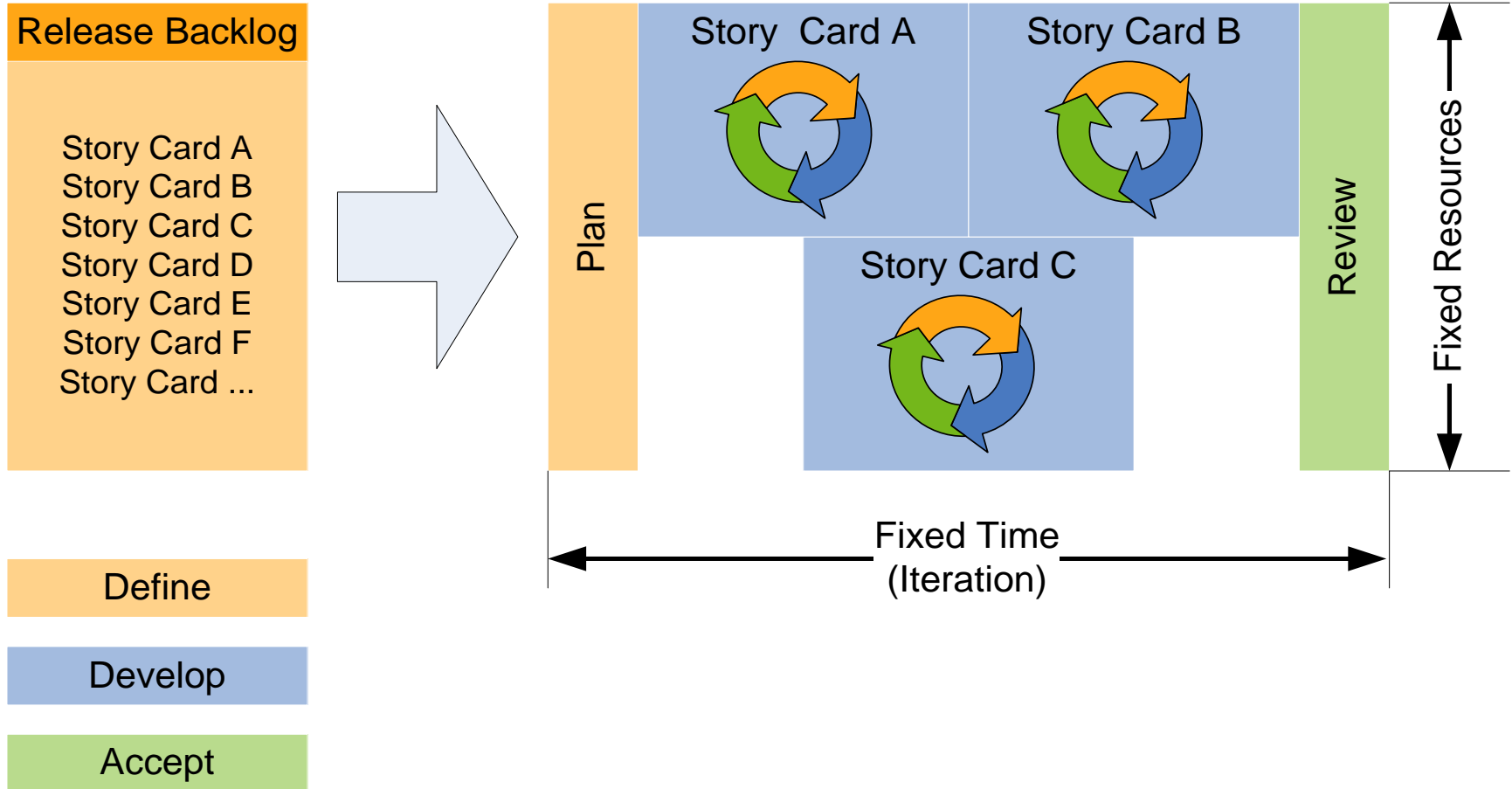
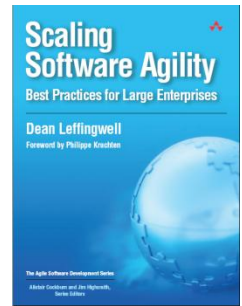


3. Mastering the Iteration



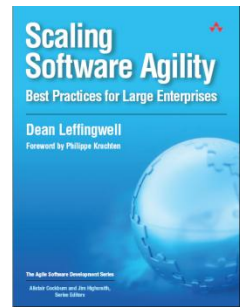
The iteration is the heartbeat of agility. Master that, and most other things agile tend to naturally fall into place.

Iteration Pattern



Iteration Cadence Calendar

Routine meeting schedules simplify planning, management, and assessment

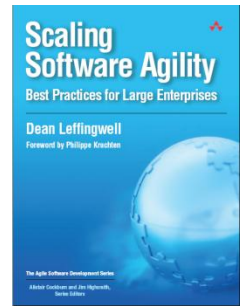


Prioritize and elaborate backlog



Day 1	Day 2-9	Day 10
Iteration planning	Daily standup	Daily standup
	Define, design, develop, test, accept	Demo
Iteration commitment		Retrospective
	Day 6 Mid iteration review	

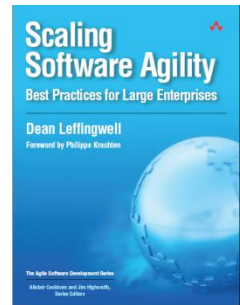
Commitment



- *If you ask a team to choose items from a (prioritized) list that the members believe they can do in a short time-box, the team will probably choose and commit to a reasonable set of features.*
- *Once the team members have committed to a set of features that they think they can complete, they will probably figure out how to get those features done within the time-box.*

Mary Poppendiek – Lean Software Development

Short Iterations

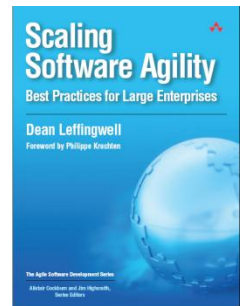


- Systems are build as increments in short iterations
- Literature counsels lengths of 1 week (XP) to 30 days (Scrum Sprint) to 2-6 wks (RUP)
- Author's experience has shown that an optimum iteration length is often **two weeks**
 - More than one week to build real functionality
 - Short enough to limit procrastination
 - Twice as many times to fail in a month
 - Natural calendar cadence

Daily Standups

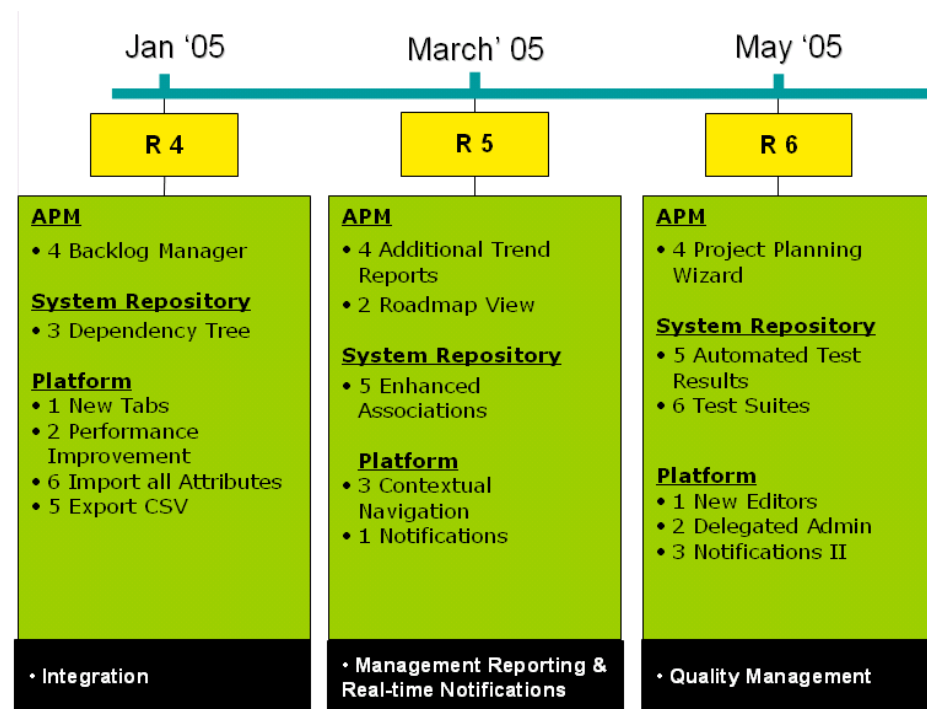
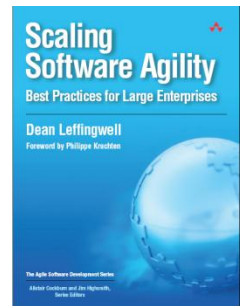
- 15 minute meeting - same time and place every day; the team decides when
- Make sure everyone stands up; design and problem solving discussions *after* the stand-up.
- 15-minute time box; stick to 1 to 2 minutes per report
- Start and stop on time; late-comers seriously affect ability to keep the meeting to 15 minutes
- ALL team members Participate

-
- “Meet After” – keep only needed participants for design and problem solving
 - Attendance and length varies daily

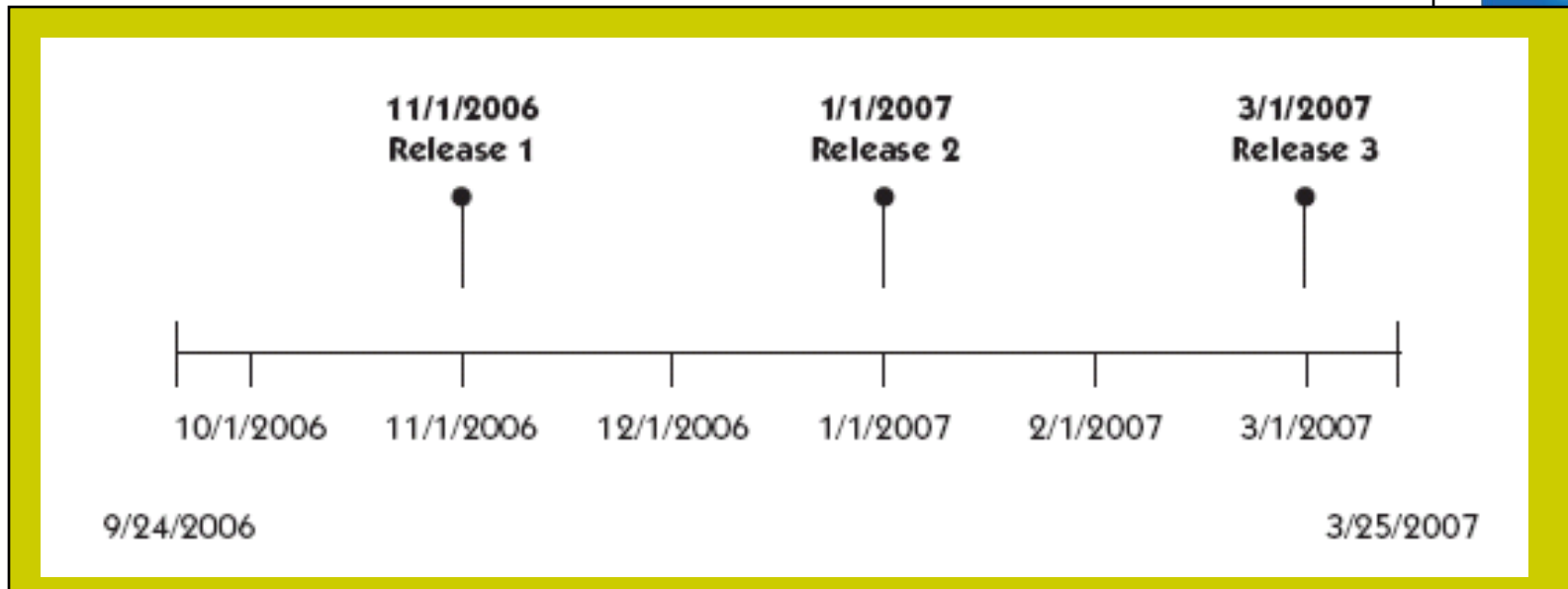
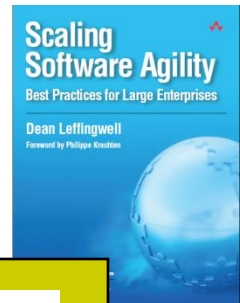


4. Smaller, More Frequent Releases

- Fix release dates
 - 60-120 days, 90 days typical
- Releases defined by
 - Date, theme, planned feature set, quality
- Scope is the variable
 - Release date and quality are fixed

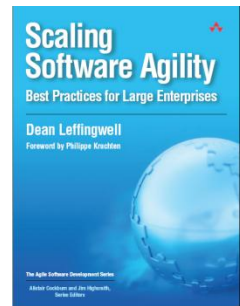


Fix the Dates - Float the Features



- Teams learn the discipline that dates **MATTER**
- Product owners learn the discipline that priorities **MATTER**
- Managers learn that agile development teams **MEET** their commitments

Benefits

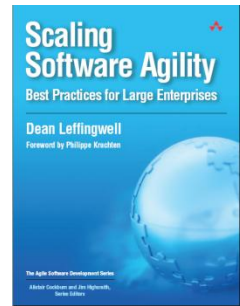


- Rapid customer feedback reduces waste
- Earlier value delivery against customer's highest needs (no fluffy features)
- Frequent, forced system integration improves quality and lowers risk
- Low cost to change
 - Accepts new, important customer features
 - Reprioritize backlog at every iteration & release
 - Reduced patching headaches
 - "It's only X days the next release, that feature can wait"
 - Or easy, high-confidence patching
- Smaller increments for higher productivity
 - Leaner flow through the entire organization to customer

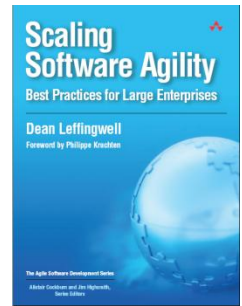
5. Concurrent Testing

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.

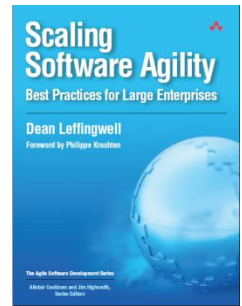


Concurrent



- Unit Testing
 - Developer written
- Acceptance Testing
 - Customer, product owner, tester written
- System, Performance and Reliability Testing
 - QA and Developer Written

On Test Automation

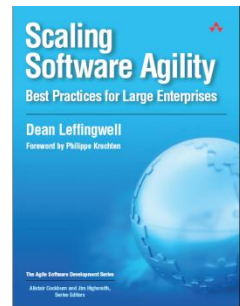


Automate Now.

You have no choice

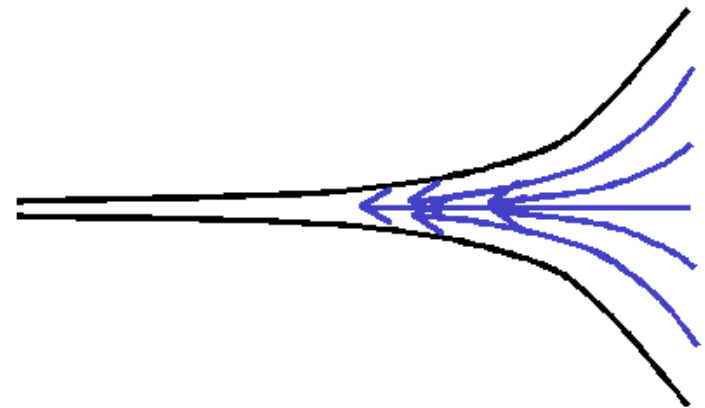
- **Manual tests bottleneck velocity**
- **You can't ship what you can't test**

6. Continuous Integration

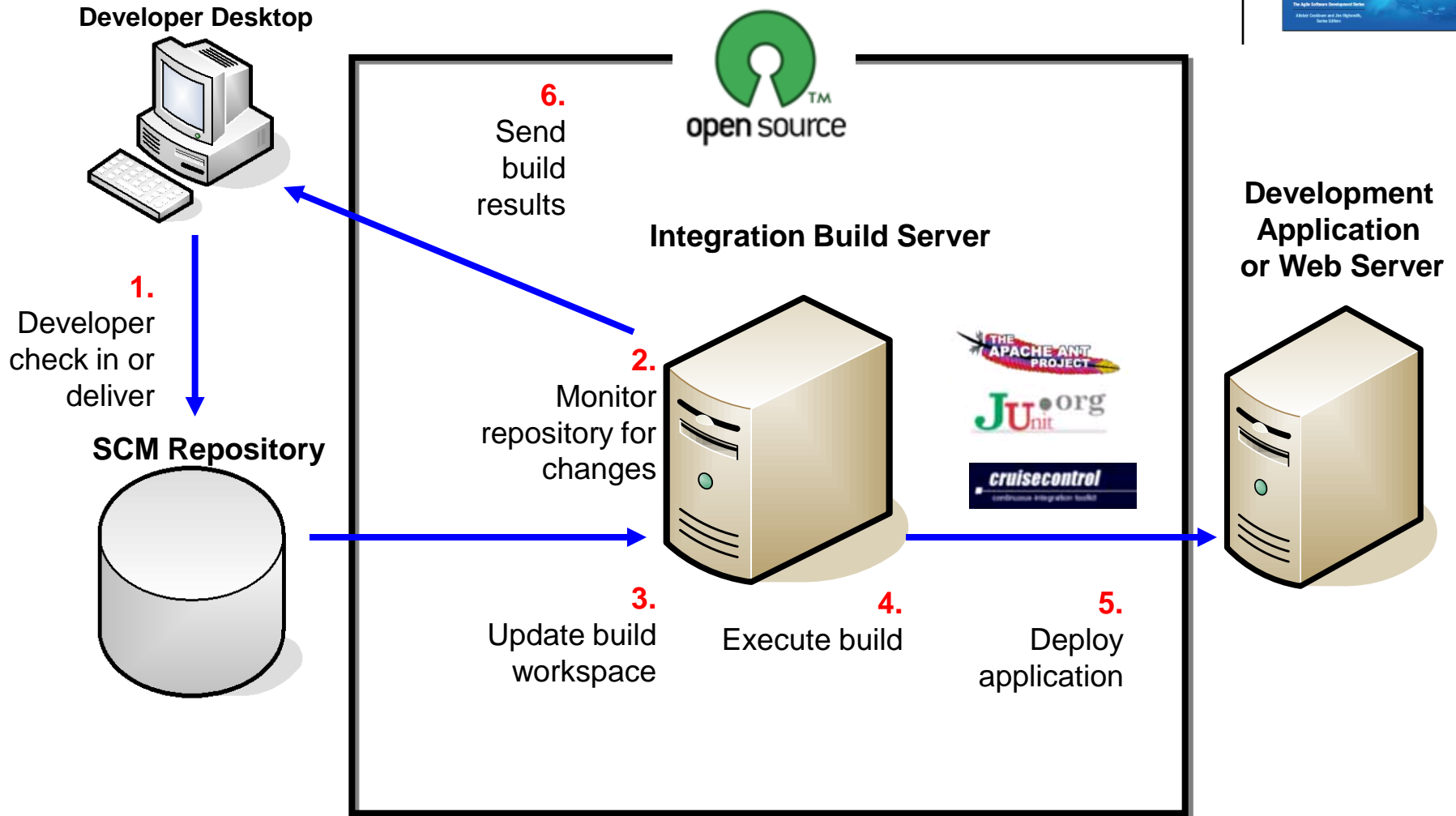
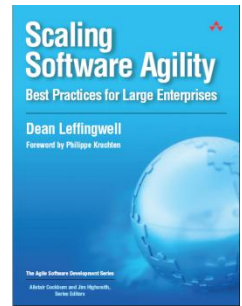


- Continuous integration is neither new nor invented by agile
- It has been applied as a best practice in many methods for at least a decade
- However, continuous integration is mandatory with agile

*the teams ability to build code
a critical bottleneck*

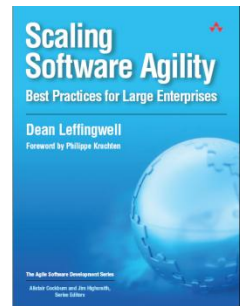


Continuous Integration Model



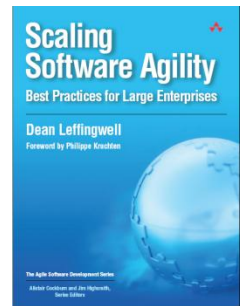
Continuous Integration Success

- 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



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

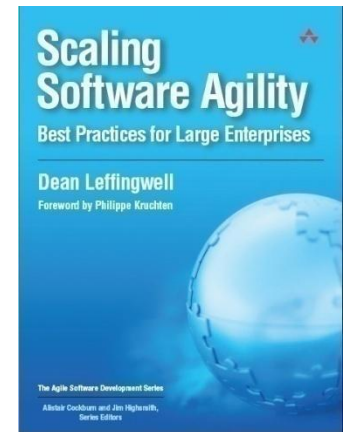


- Periodically, the entire team including owners/end users
 - reflects on the results of the process
 - learn from that examination
 - adapt the process - and organization - to produce better results
- The team decides what to *keep doing*, what to *stop doing*, and what to *do differently* next time

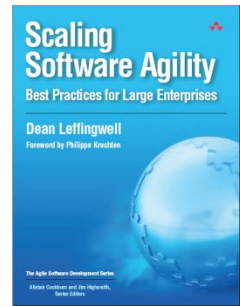
The shorter the iterations and releases the faster the learning and the virtuous cycle repeats

Part III – Seven Enterprise Practices

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



XP

- No Big Up-Front Design, architecture emerges through continuous re-factoring

SCRUM

- Organize teams around a feature-based or component architecture

FDD

- Domain model as simple, central, continuous model

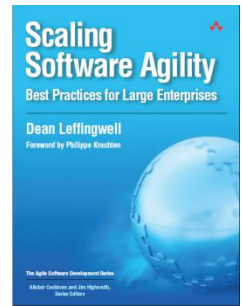
DSDM

- Model as necessary, model guidance

RUP

- Architecture-centric, build architecture in early iterations,

How Much Architecture You Need

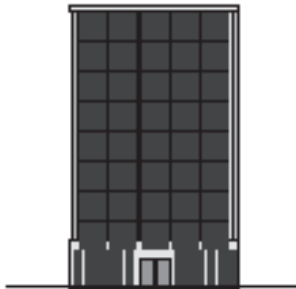


teams of teams,
systems of
systems

small team scale

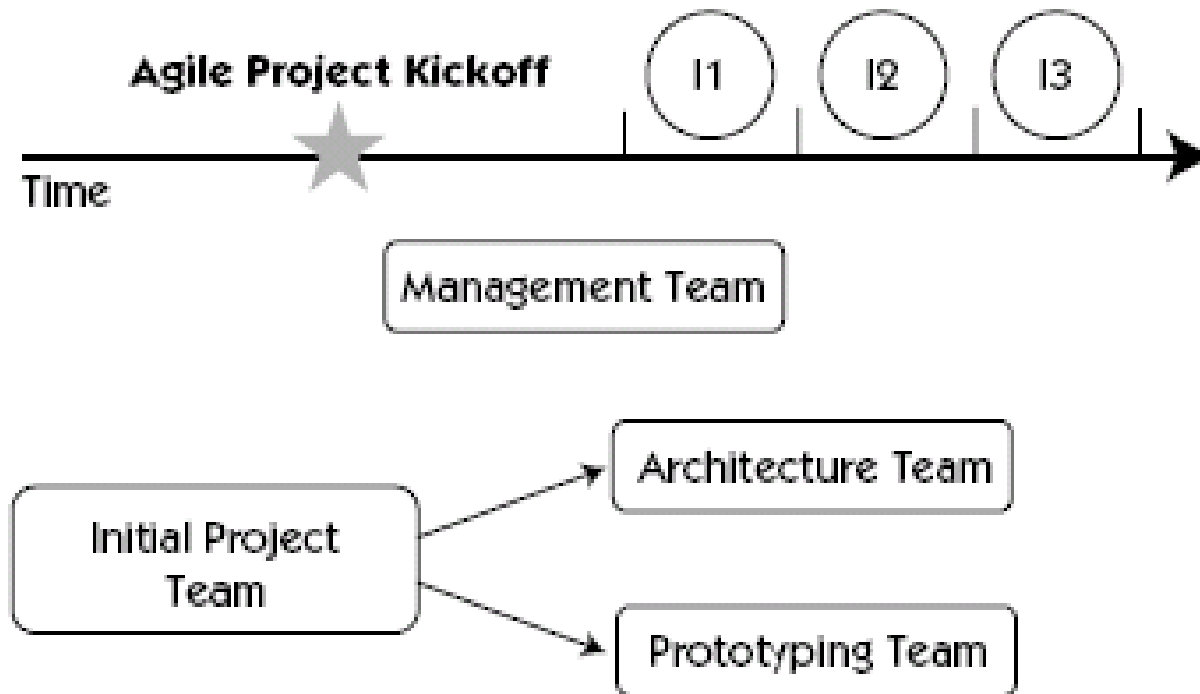
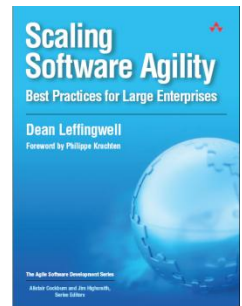


more teams scale

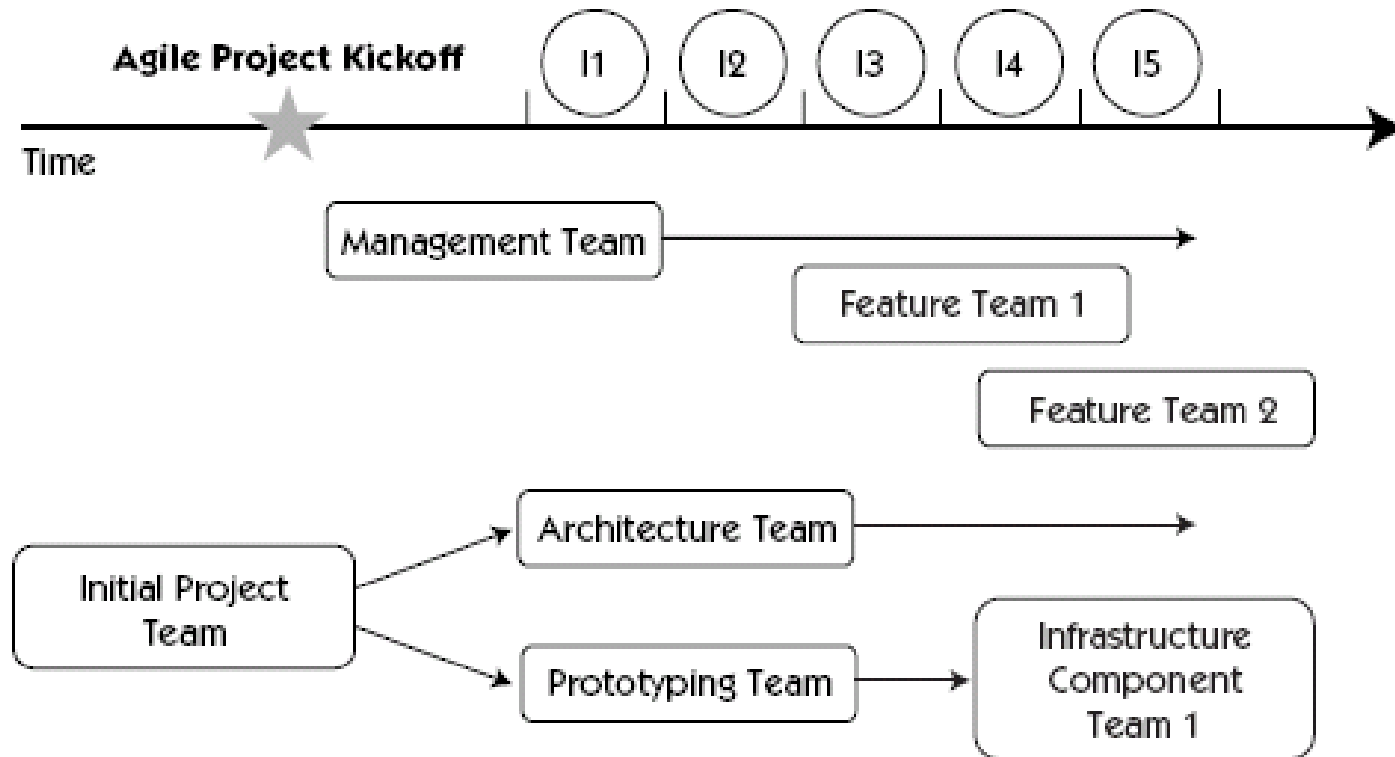
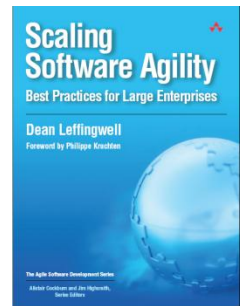


depends on what you are building.

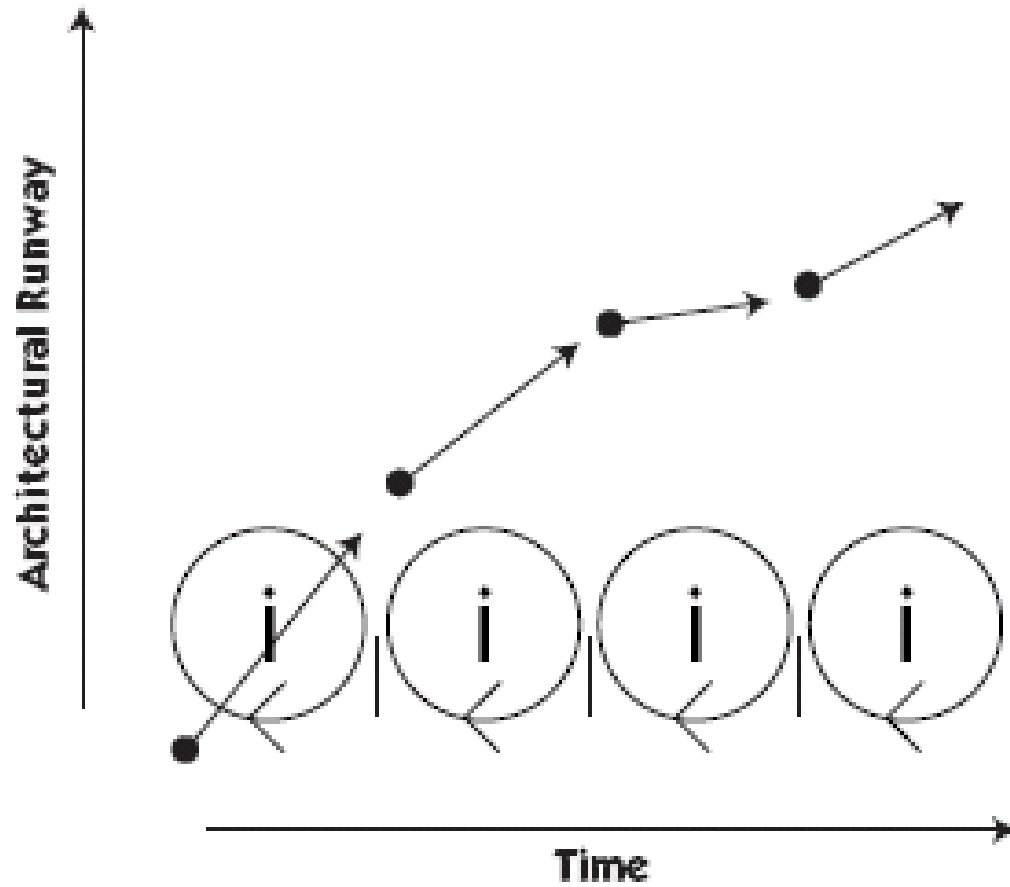
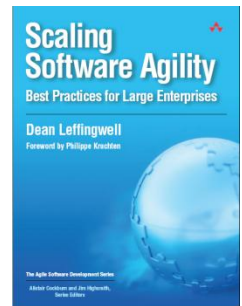
Building Architecture in Early Iterations



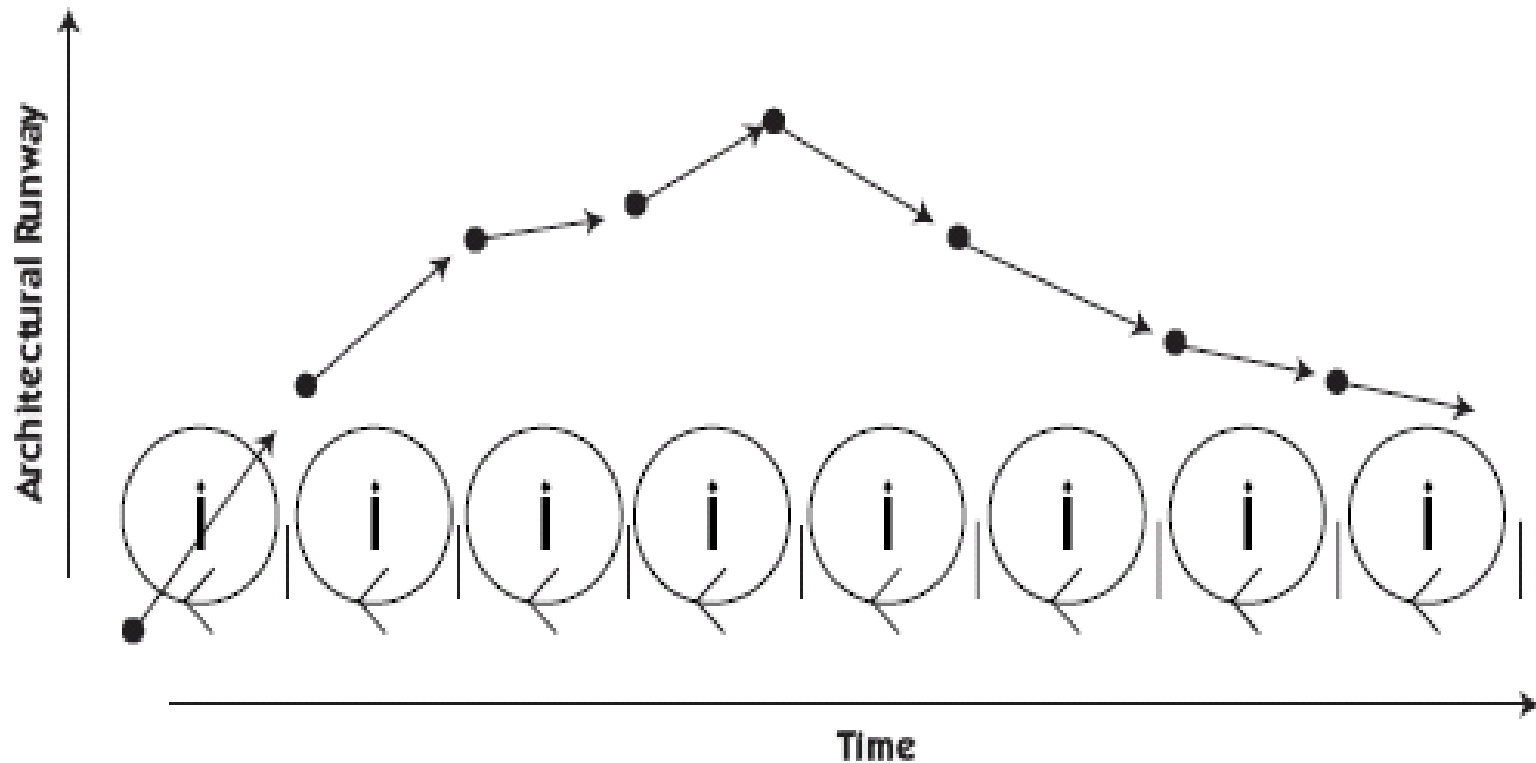
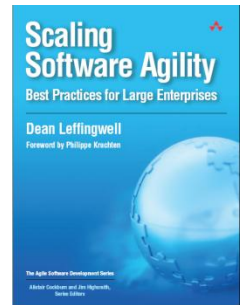
Adding Feature and Component Teams



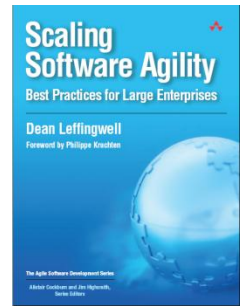
Builds Some Runway



Which Will be Used Over Time....



2. Lean Requirements at Scale



Traditional process is prescriptive:

- Emphasis on getting all the requirements right and written early in the project.

“get it right the first time”

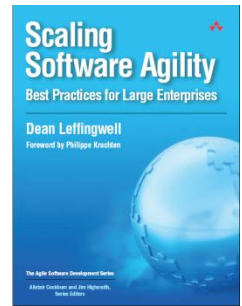
Agile process accepts change

- Requirements gathering ongoing, iterative, process
 - Acknowledge it's impossible to get all of the requirements up front
 - Acknowledge that there is a time and relevance dimension to requirements

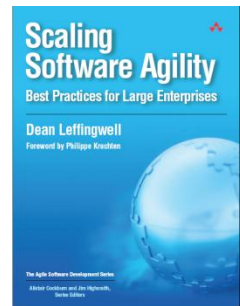
Start now- evolve to “right”

2. Lean Requirements at Scale

- Vision
- Roadmap
- Just-in-Time Elaboration



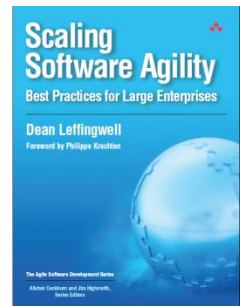
Vision – Data Sheet Approach



- Envision the product “flyer” or product “box” that describes the product to perspective users
 - What problem does your product solve?
 - What features and benefits does it provide?
 - How do you communicate that to the prospect?
 - What performance does it deliver?
 - What platforms, standards, applications, etc does it support?

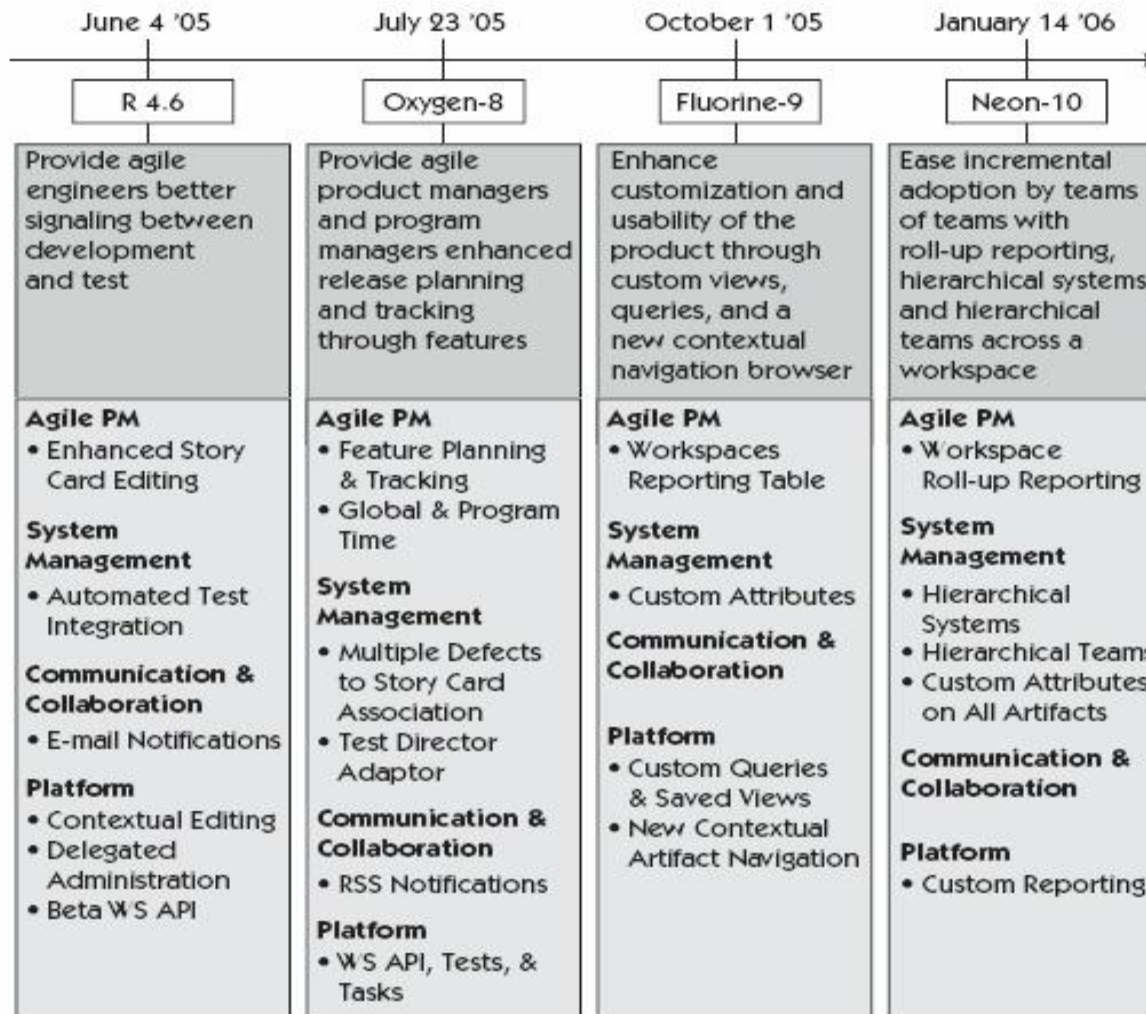


Vision/Supplemental Records Non-Functional Requirements

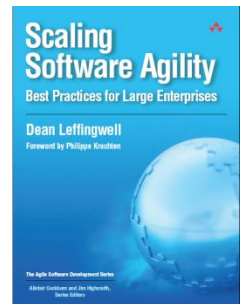


- Some requirements must be known by all teams and cannot simply emerge
 - Performance and reliability requirements
 - Industry/Regulatory/Customer Standards and specifications
 - Common behavior across like components
 - Copyright, logo, graphics, accessibility, other corporate requirements
- These must be documented in an online repository, and made continuously available to all component teams

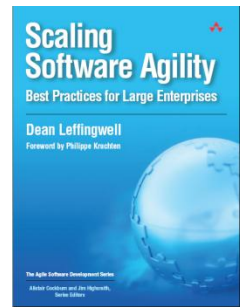
Roadmap



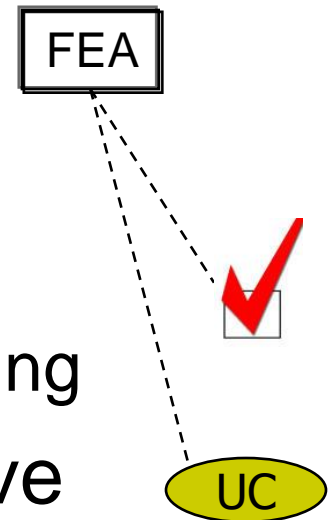
A Product Roadmap Showing Four Releases



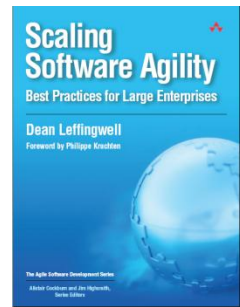
Just-In-Time Elaboration



- Agile investment in “documenting requirements” has been minimal so far
 - Features are very high level, abstract
 - Sufficient to communicate concept
- Elaboration is now required
 - Refine the team’s understanding
 - Support design, implementation and testing
- User Stories, Use Cases or Declarative Requirements

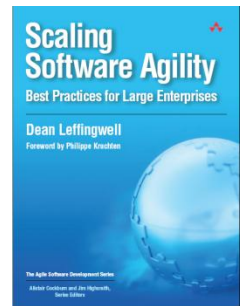


3. Systems of Systems and the Agile Release Train

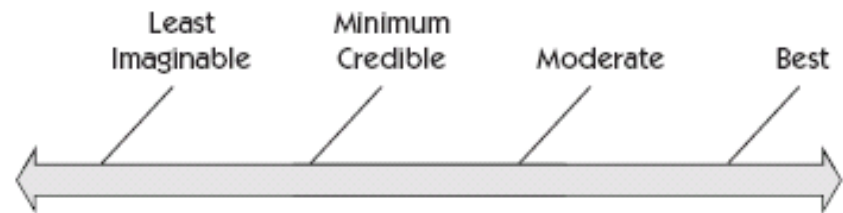


- Scaling agile requires managing interdependencies amongst distributed teams of component developers
- This necessarily requires more predictive planning, and somewhat longer release cycles
- Scaling agile requires a component-based, “agile release train” delivery model

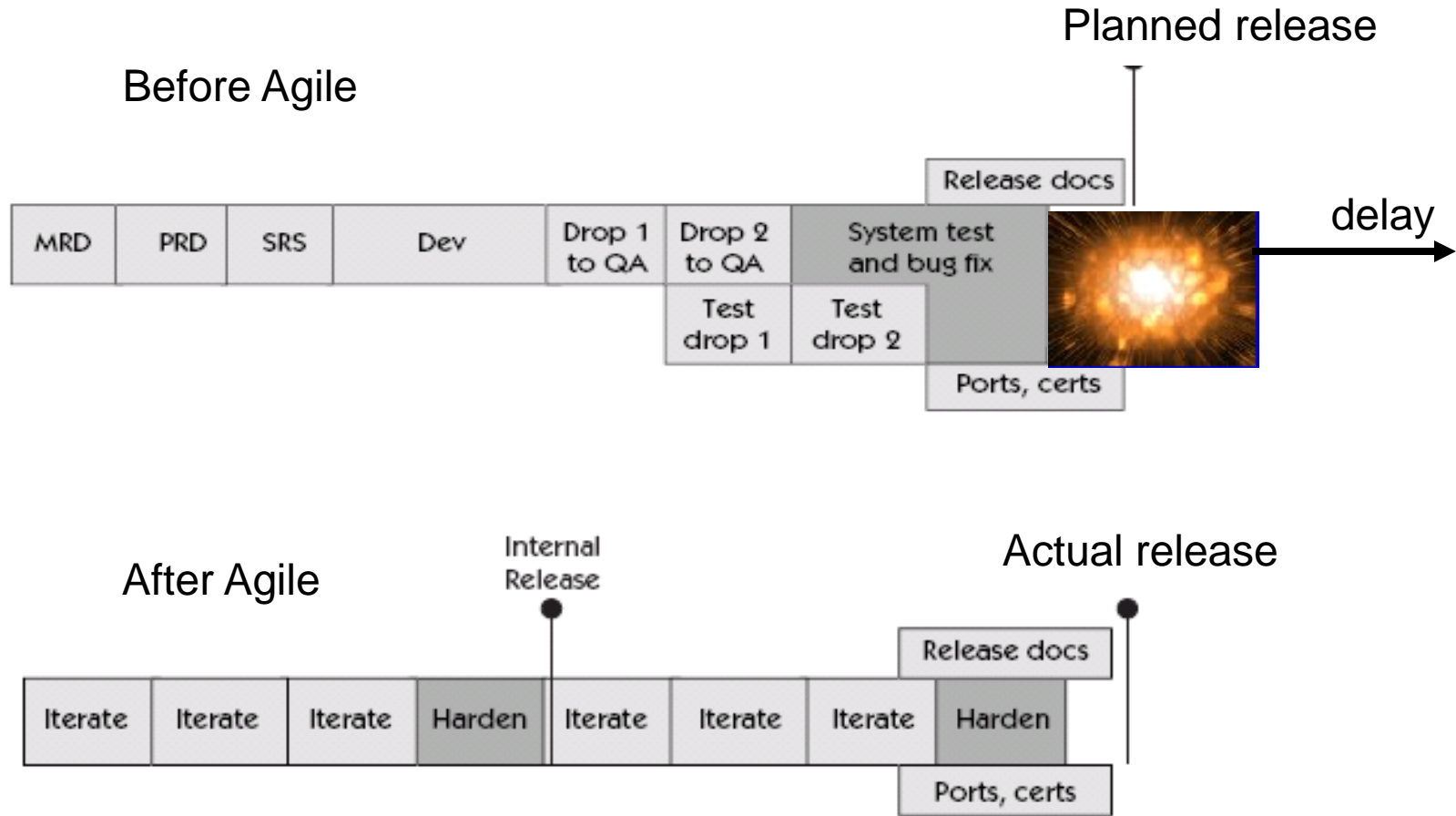
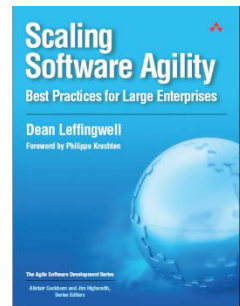
Principles of the Agile Release Train



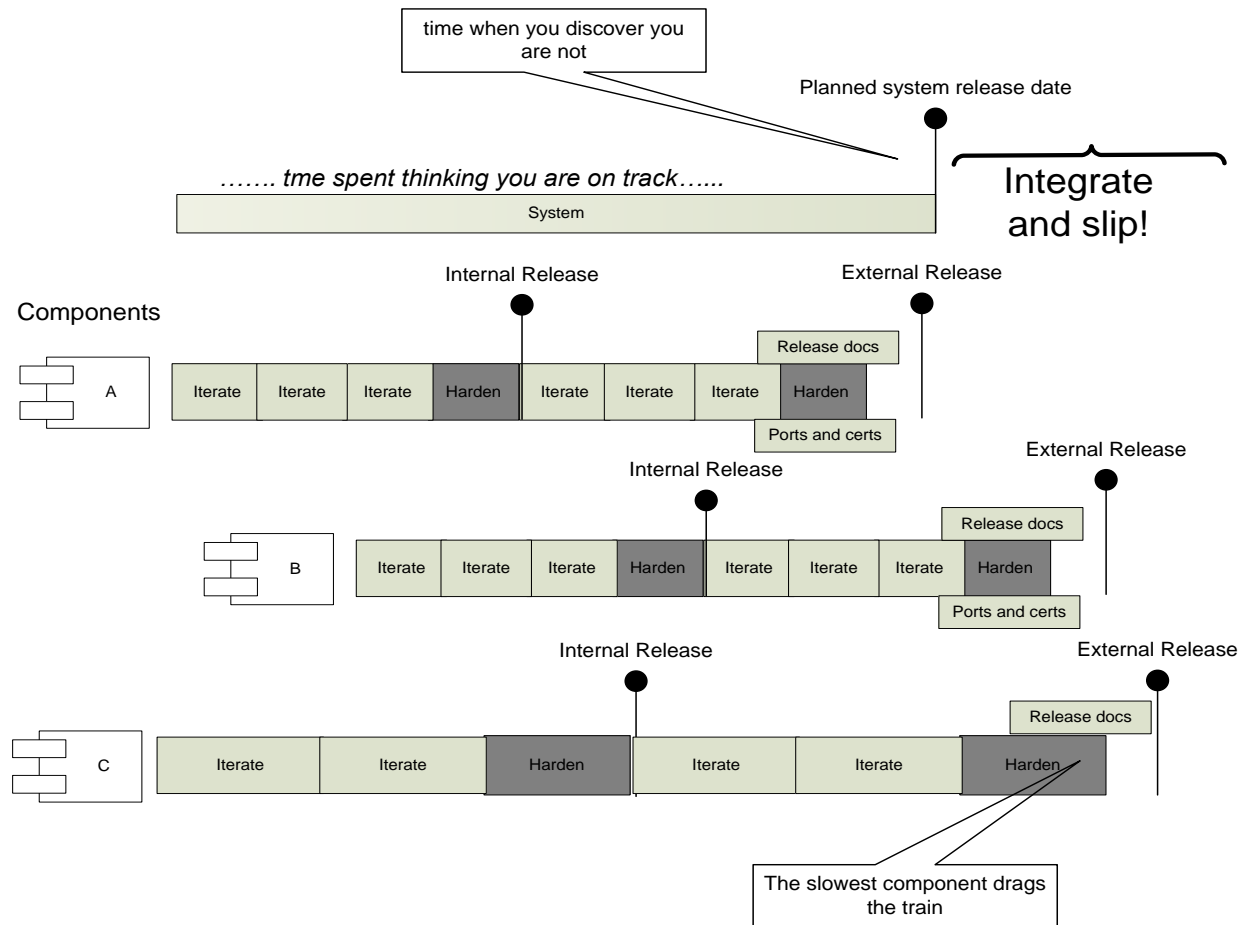
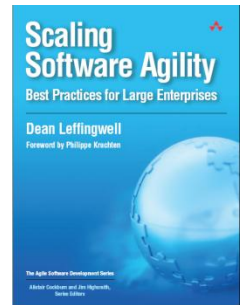
- Release dates for the suite or platform are fixed
- Intermediate, global integration milestones are established and enforced
- Constraining these means that functionality and iteration schedules for the components must flex
- Infrastructure components (interfaces, SDKs, installation) must track ahead
- Component vendors evolve to a flexible model:
 - Design spectrum for new functionality
 - Backup plan to ship the old version if necessary



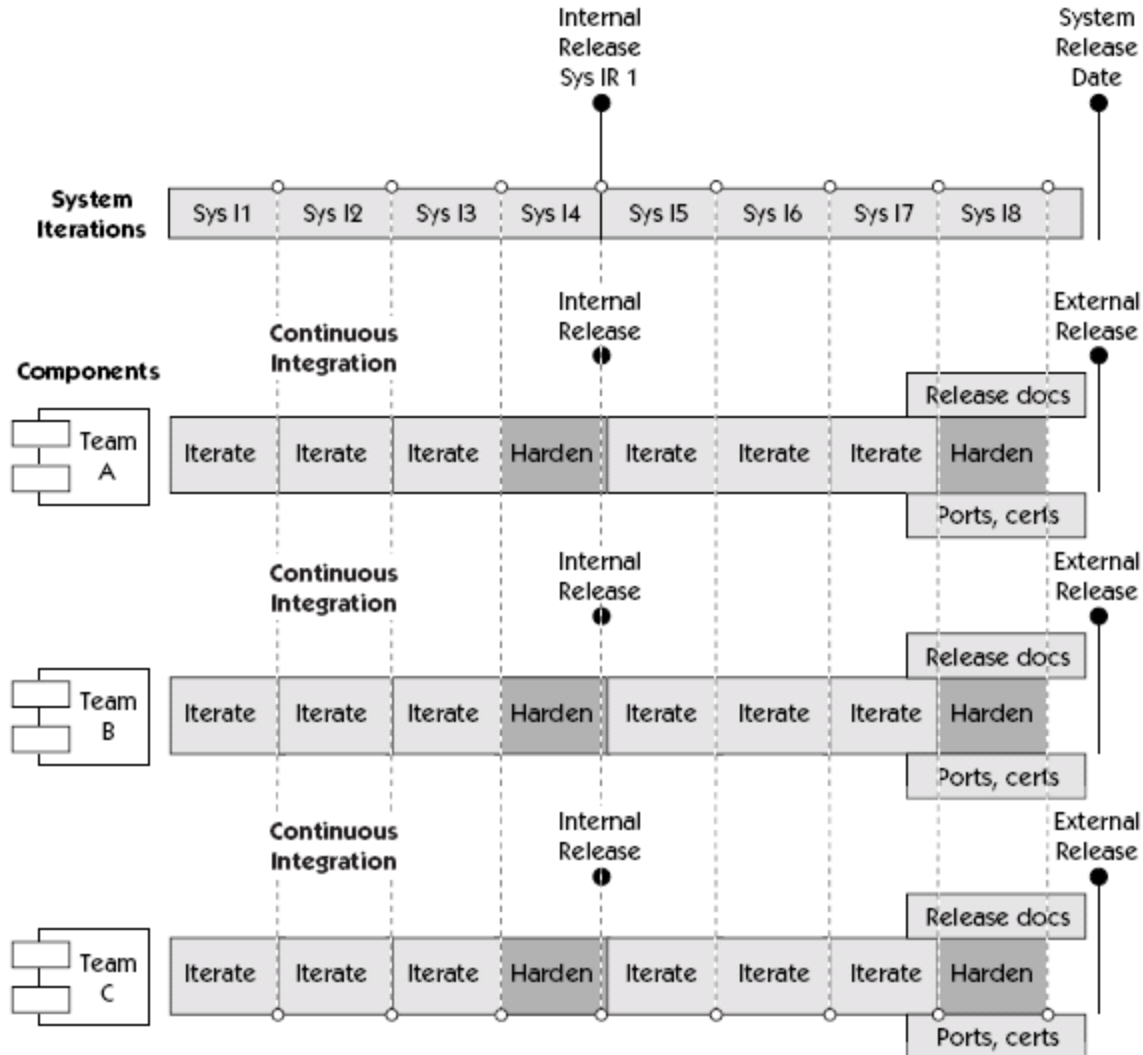
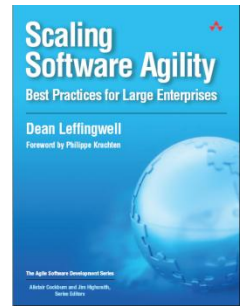
Agile Synchronizes and Better Assures Component Delivery



But Component Agile is not System Agile

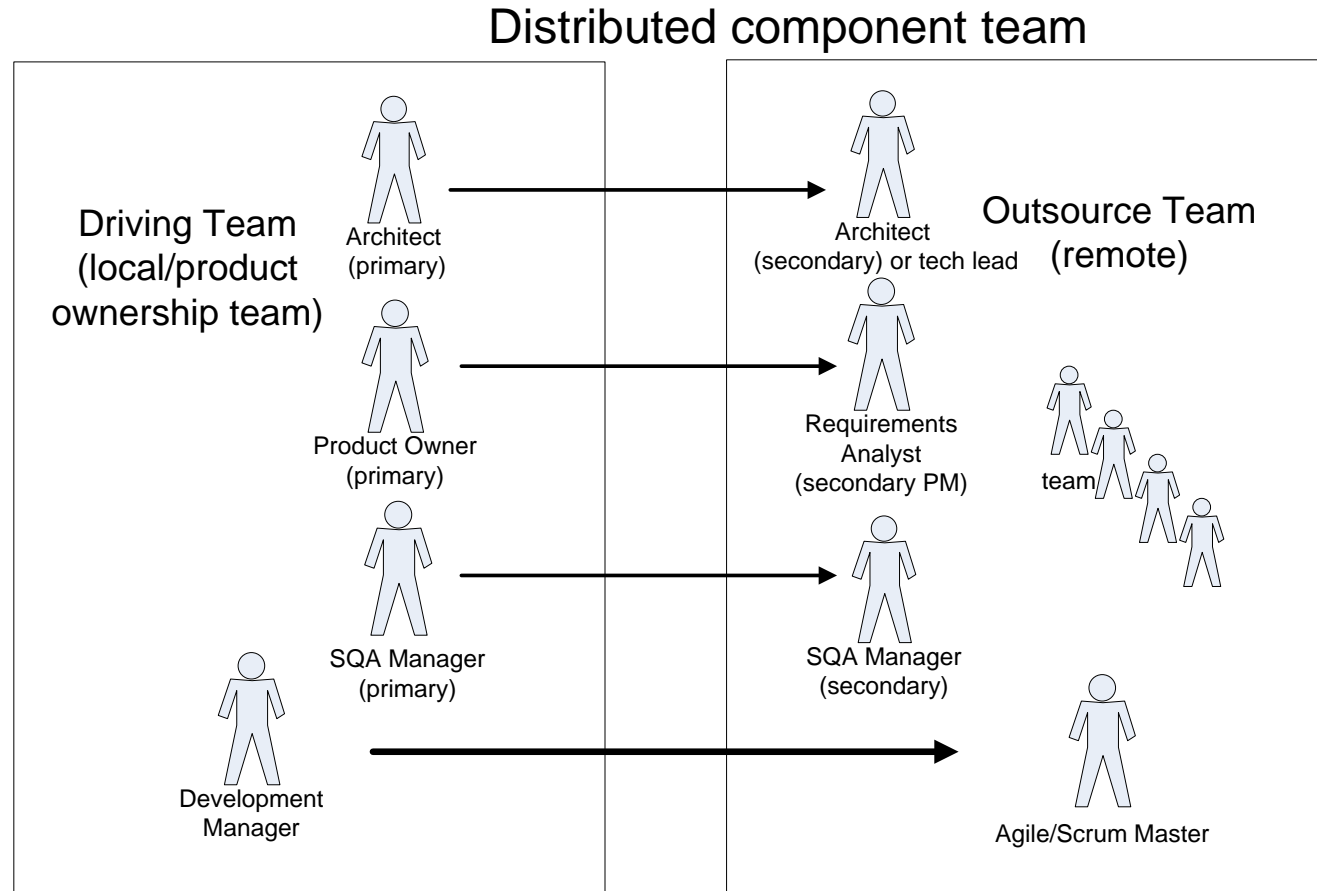
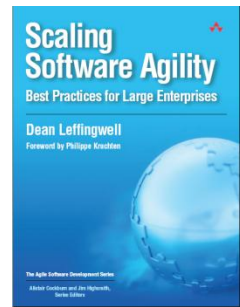


Synchronized Release Train

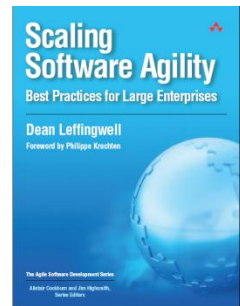


S
H
I
P
!

4. Managing Highly Distributed Development



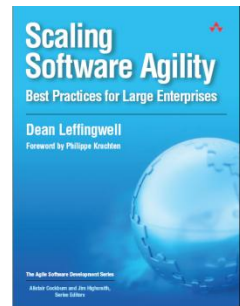
Asynchronous Collaboration



- Shared, program-wide visibility into priorities, status, dependencies & blocks for tight coordination across time zones
 - Communication, communication
 - Skype, wikis, IM, email, Web conferencing, VOIP, international mobiles
 - Shuttle diplomacy
- +A backup for all communication channels



Backlog Management Across Teams



- Centrally capture, organize and decompose by program or project/component
 - Feature lists, user stories, and non-functional requirements
 - Priorities, estimates, status and owners

All	Rank ▲	ID	Name	Release	Iteration	Priority	Status	Story Card Accepted
<input type="checkbox"/>		# <input type="text"/>				All	All	
<input type="checkbox"/>	1.0	FES	Must have shipping functionality			Critical	Completed	(1 of 1) 100%
<input type="checkbox"/>	2.0	FE3	Allow the customer to view their order status			Critical	Planned	(2 of 3) 67%
<input type="checkbox"/>			Component B3					
<input type="checkbox"/>			Component B1					
<input type="checkbox"/>		SC33	Implement UC4: Chi... the Status of Your C...	Olympus Mons (5,6,7)	Iteration 5 (OM)	Useful	B D P C A	
<input type="checkbox"/>			Component B2					

Real-Time Project Reporting



- Quick, daily updates to task estimates, status & remaining effort
- Roll-up by project and by program
- Automated burn down charts

Iteration Status: Iteration 6 (OM): 2005-11-05 - 2005-11-16

Resources: 43.00 Points, 43.00 Points, 25.08 Points
 Estimates: 42.00 Points, 45.00 Points, 28.00 Points
 Available: 1.00 (2.00) (2.92)

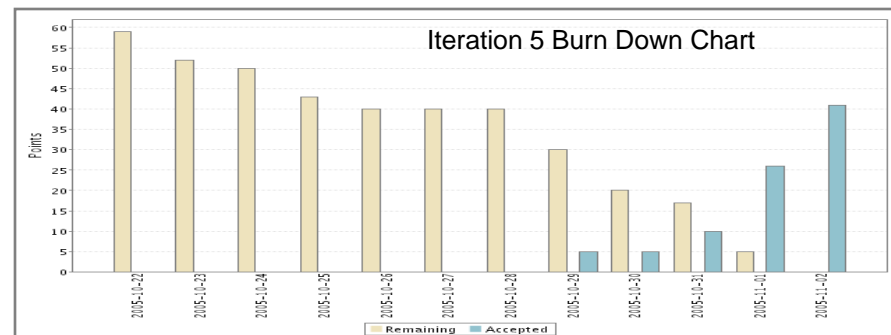
Today: 2005-11-09 Accepted: 2.00 Po
 Start: 2005-11-05 End: 2005-11-16

State: Committed

Timeline

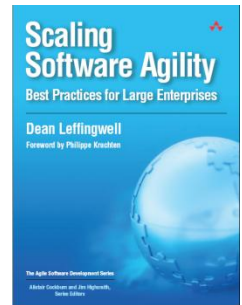
All	Rank ▲	ID	Name	Plan Est	Task Est	To Do	Status	Failed Test Cases
⊕ ⊖	#						All	All
⊕ ⊖	6.5	SC74	Implement UC2: Validate customer (2)	2.0	2.0	0.0	B D P C A	0
⊕ ⊖	7.0	SC20	Implement UC9: Search for Items	6.0	12.0	4.0	B D P C A	1
		TA45	GUI for Search for Items		7.0	0.0	D P C	
		TA46	Implement Business Rules for Searching		5.0	4.0	D P C	
⊕ ⊖	8.0	SC5	Implement UC7:Purchase Your Items Part 2	9.0	10.0	4.0	B D P C A	5
⊕ ⊖	9.0	SC23	Implement FE11: Allow priority shipping options	4.0	3.0	2.0	B D P C A	2

B Backlog D Defined P In-Progress C Completed A Accepted ● Blocked

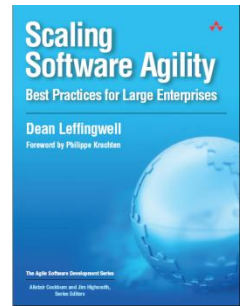


5. Changing the Organization

- Rolling Out Agile
- Organizing for Agile Scale
- Eliminating Impediments

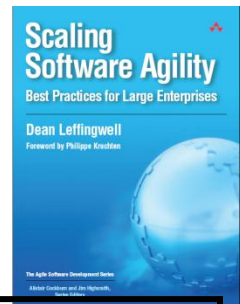


Adopting Agile through Scaling and Maturity



Maturity	Beginning (flow)	Intermediate (pull)	Expert (innovate)
Scale			
Multi-Program Organization			Get Here
Multi-Team Program			
Individual Teams	Start Here		

Successful Adoption Path

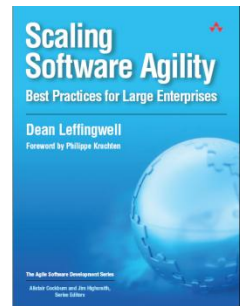


Scale \ Maturity	Flow (Beginning)	Pull (Intermediate)	Innovate (Expert)
Multi-Program Organization		Step 4 ↑	Step 5 ←
Multi-team Program		Step 3 ↑	
Individual Teams	Step 1 →	Step 2	

Each step has a formula of:

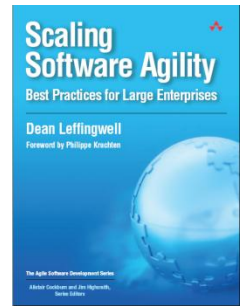
- Disciplines to attain
- Roadblocks to watch for
- Results to pay for continued investment
- Can't build agile systems until teams can continuously integrate
- Tooling for success

Eliminating Impediments



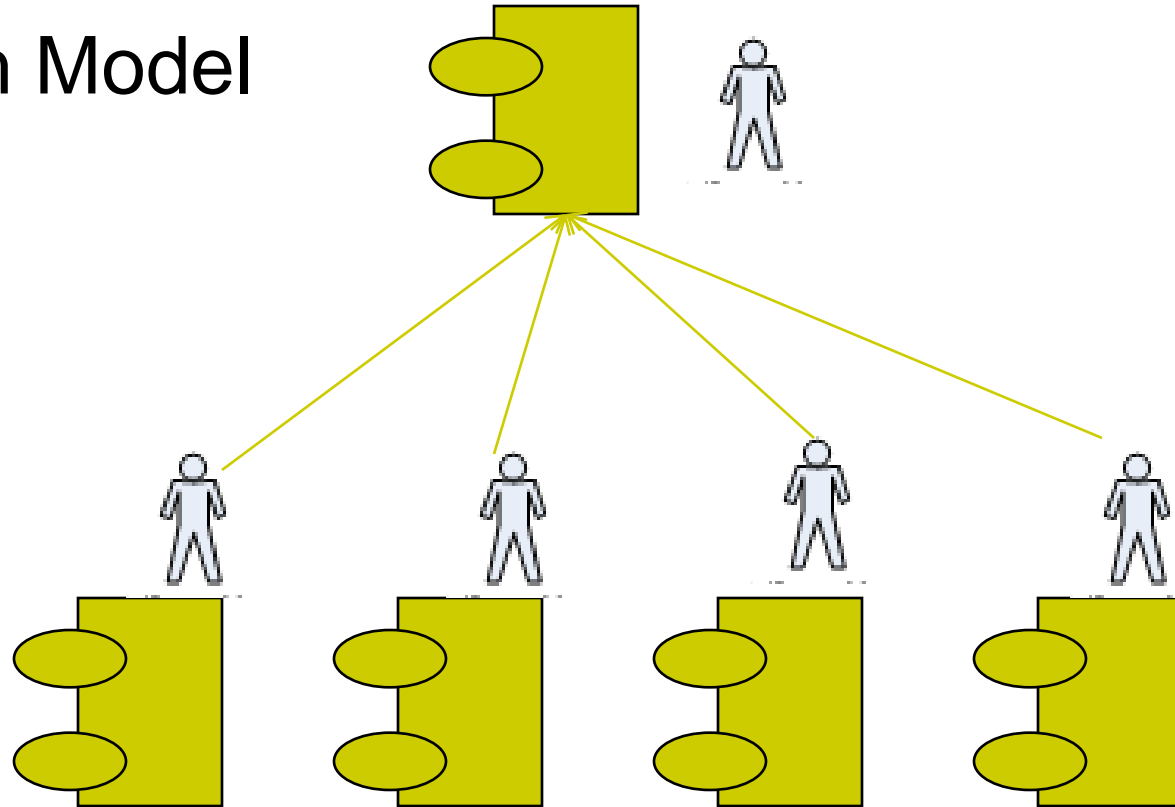
- **The CIO/Sponsoring executive is the “ScrumMaster” for eliminating impediments:**
 - Software process adherence to ineffective processes
 - Existing rules demand adherence to document-driven, waterfall approaches
 - Management assumes fixed-price, fixed-time, fixed-functionality delivery
 - If everything is fixed, it isn't agile
 - Software Test/and or System QA is not integrated with team
 - Organization rewards individual, rather than team behavior
 - Teams not co-located to maximum extent feasible
 - Teams cannot make small decisions needed to do their job
 - organizational, space, expense, technical decisions
 - Other functions, sales, marketing, support not ready for increased pace of product delivery

Organizing for Scale



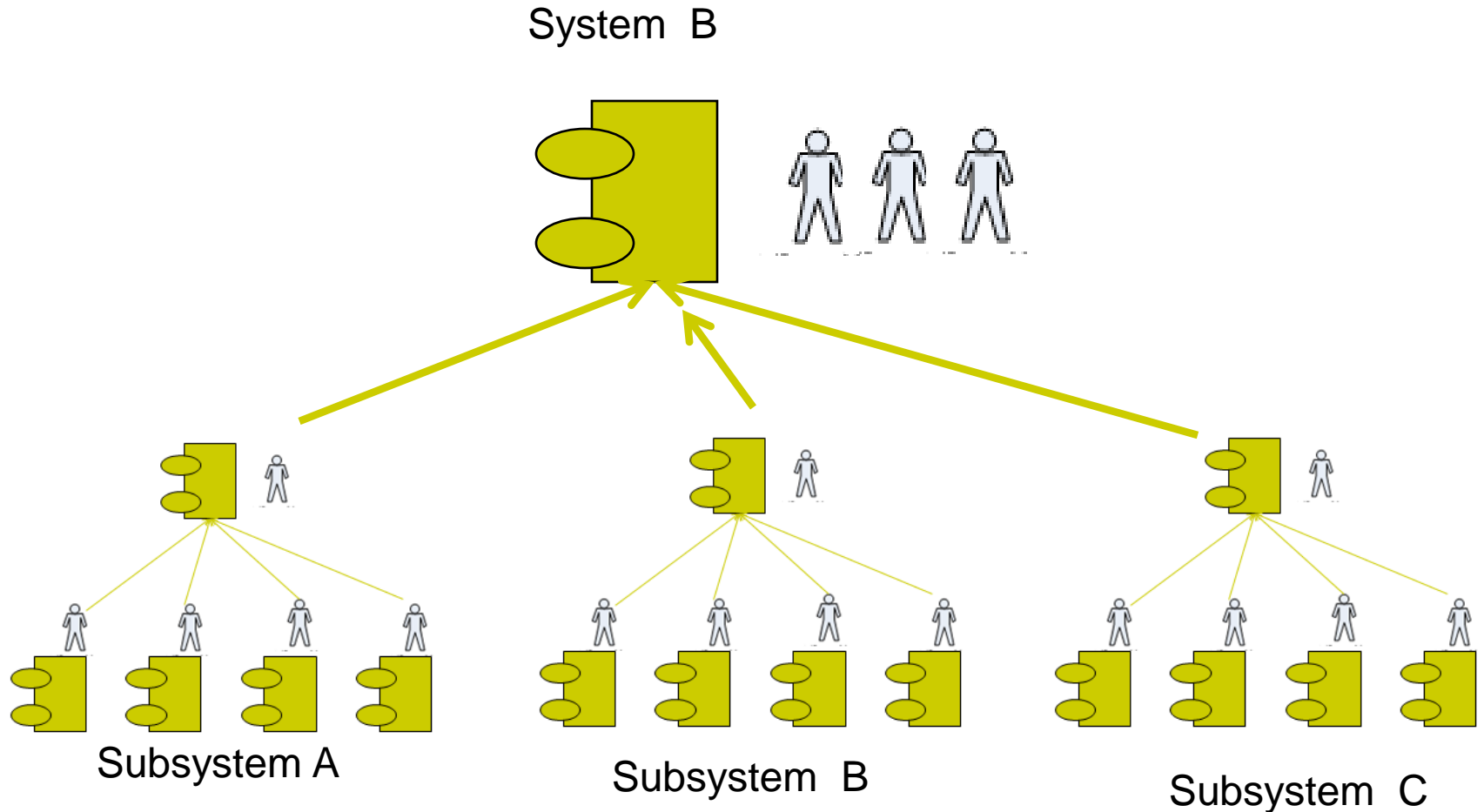
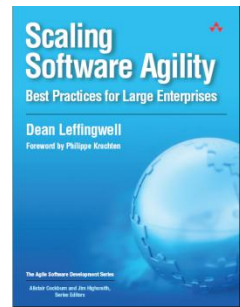
Subsystem Team

Subsystem Model



Component or Feature Teams

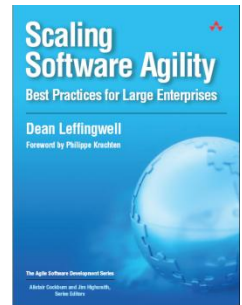
Systems of Systems-Teams of Teams



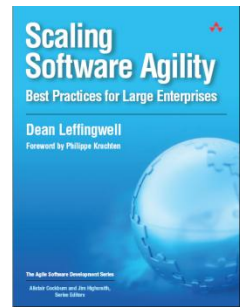
6. Impact on Customers and Operations

More frequent releases challenge:

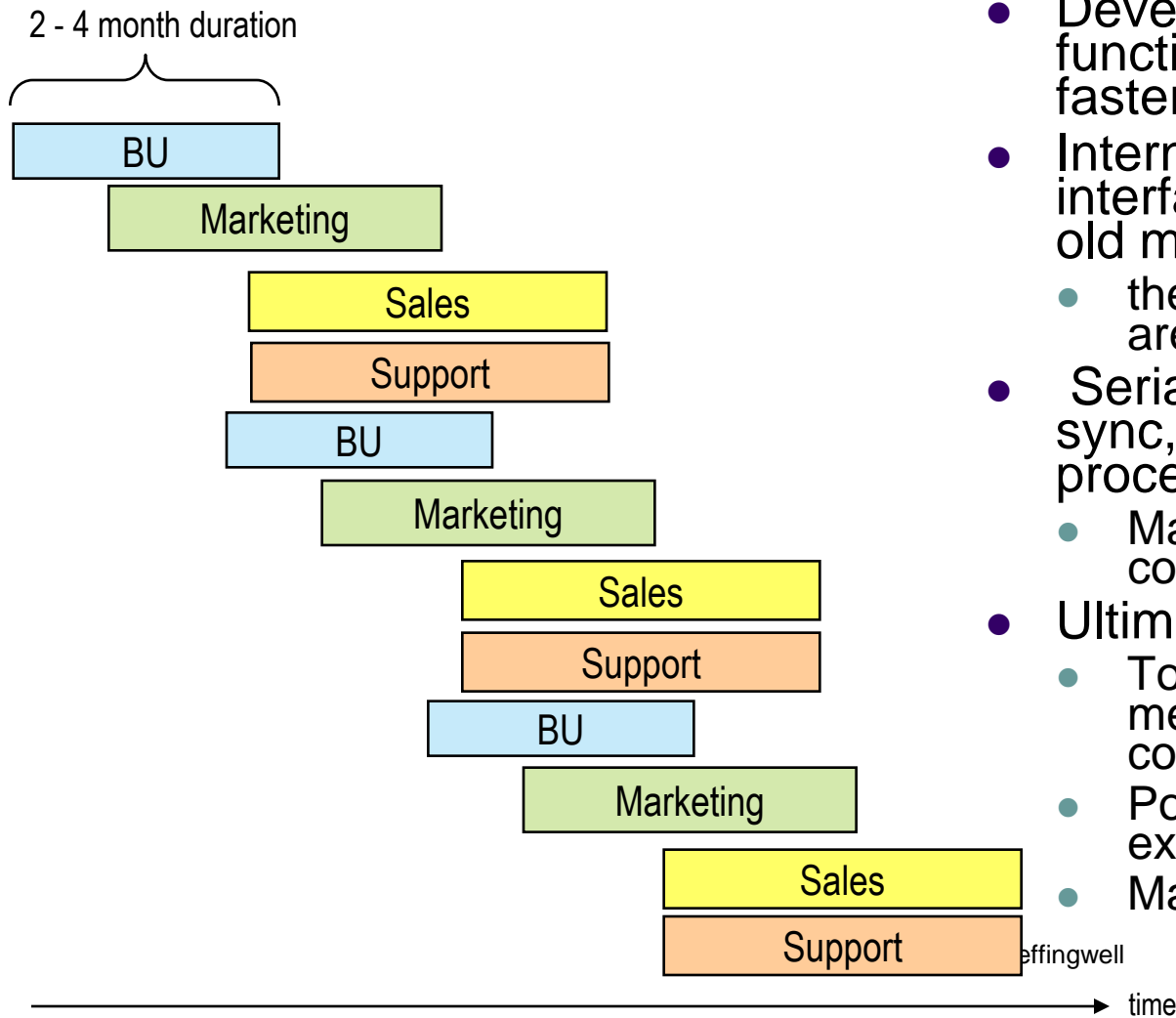
- Our customers
- Our sales and marketing teams
- Our customer care and support teams



Agile Market Release Planning – Pitfall #1

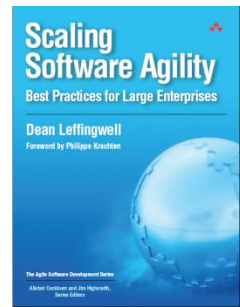


Option #1 – The “Hyper-Waterfall” Effect



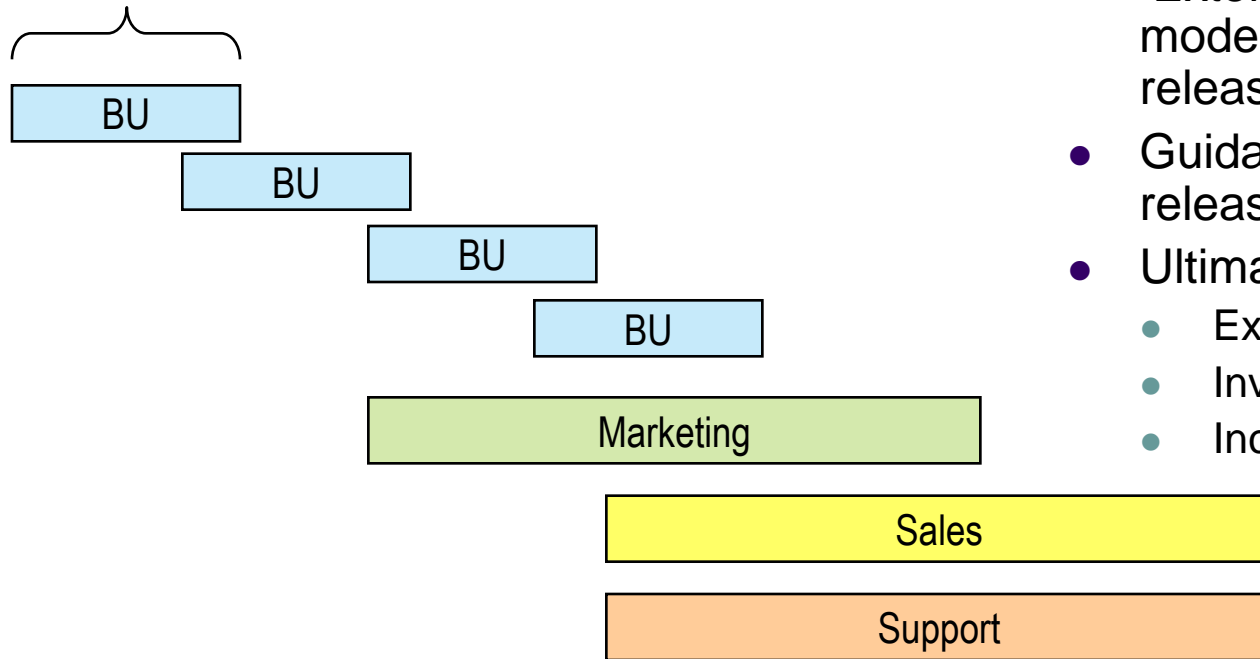
- Development ships more function, in smaller increments, faster
- Internal/External Processes, interfaces & organizations follow old model
 - they try to keep up and quickly are overloaded
- Serialization leads to, loss of sync, relationships, feedback processes
 - Machinery can break down completely
- Ultimate outcome:
 - Too much noise in system, messaging and feedback are compromised
 - Poor execution & increase execution risk
 - Market confusion

Agile Market Release Planning – Pitfall #2



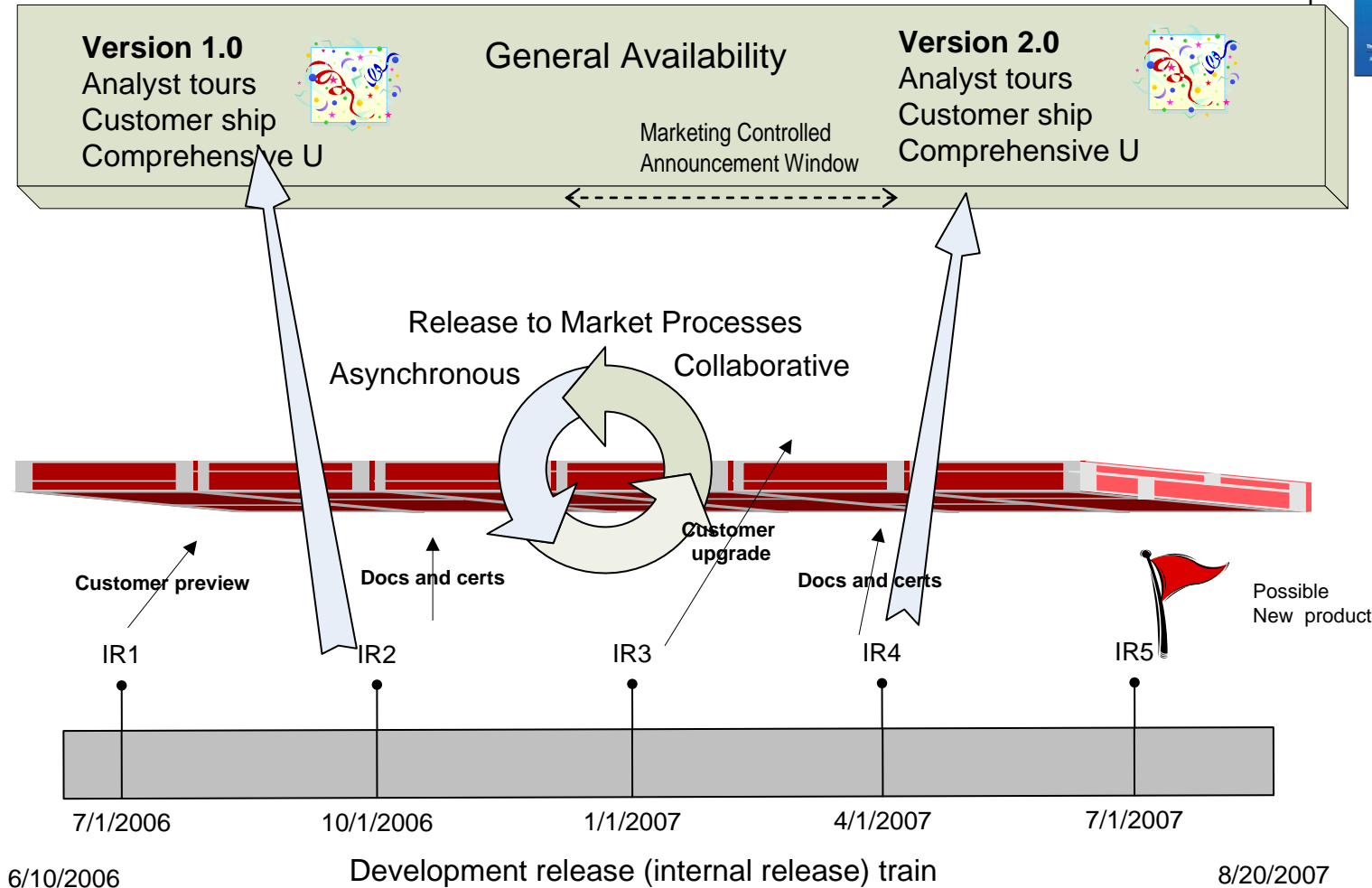
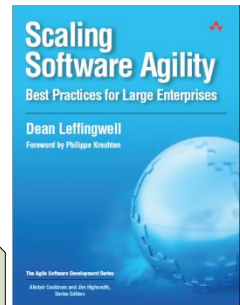
Option #2 – The “Ignore Agile” Approach

2 - 4 month duration

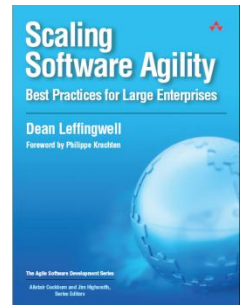


- Development ships faster, smaller increments
- External processes follows old model, ignoring incremental releases
- Guidance & feedback into releases compromised
- Ultimate outcome:
 - Extremely limited feedback
 - Inventory of undelivered value
 - Increased product trajectory risk

The Answer: Separation of Concerns



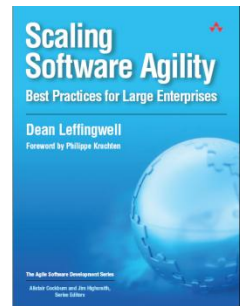
7. Measuring Business Performance



*“The primary metric for agile software development is whether or not working software actually exists, and is demonstrably suitable for its intended purpose. In Agile, this key indicator is determined empirically, by demonstration, **at the end of every single iteration.**”*

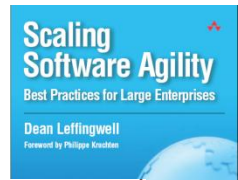
All other measures are secondary (but not useless)

Agile Project Metrics



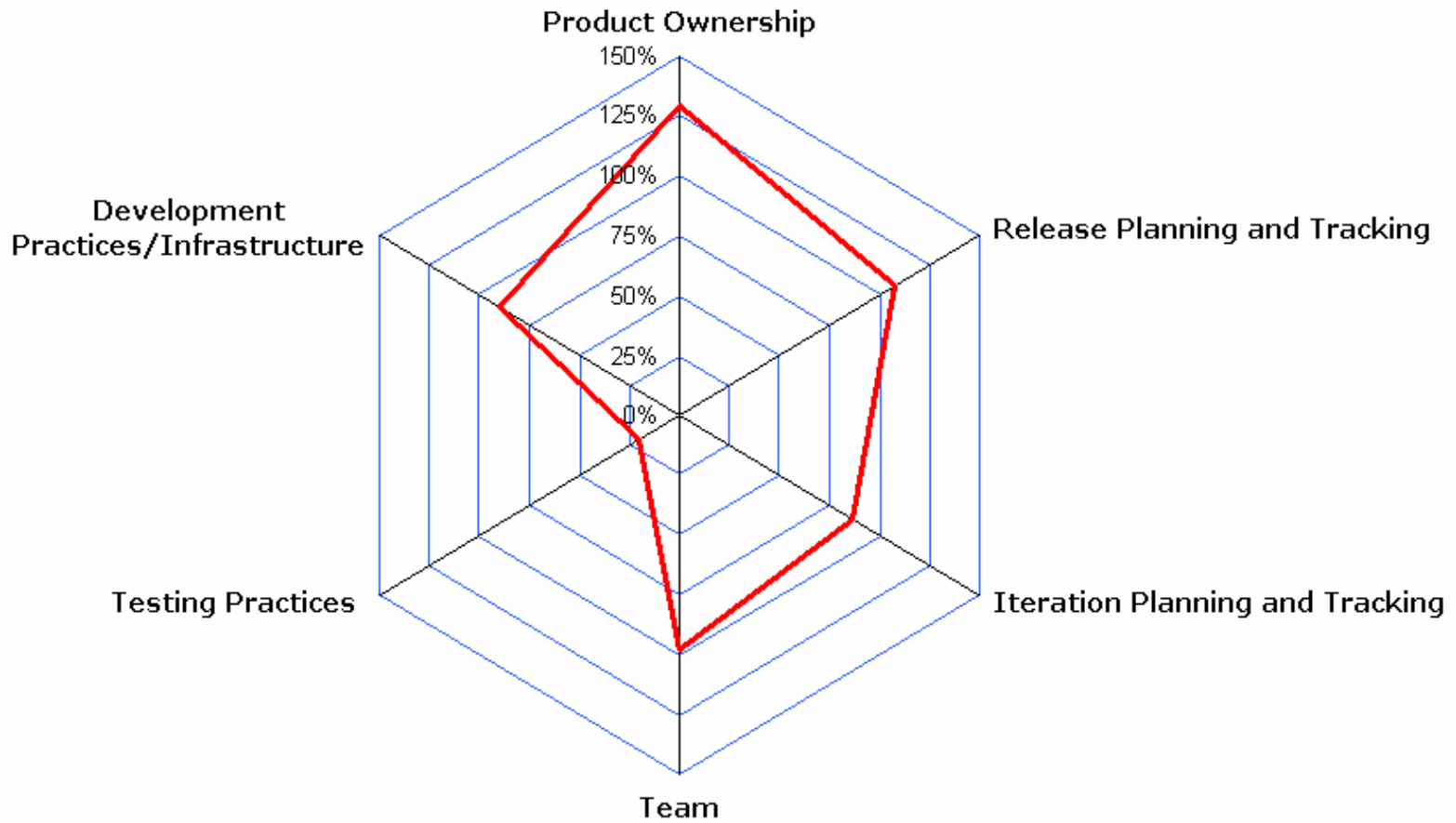
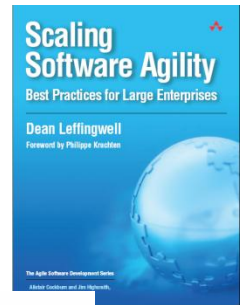
Functionality	Iteration 1	Iteration 2											
# stories (loaded at beginning of iteration)													
# accepted (defined/built/tested and accepted)													
% accepted													
# not accepted (not achieved within iteration)	<table border="1"> <thead> <tr> <th>Quality and test automation</th> </tr> </thead> <tbody> <tr> <td>% SC with test available/tests automated</td> </tr> <tr> <td>Defect count at start iteration</td> </tr> <tr> <td>Defect count at end of iteration</td> </tr> <tr> <td># new test cases</td> </tr> <tr> <td># new test cases automated</td> </tr> <tr> <td># new manual test cases</td> </tr> <tr> <td>Total automated tests</td> </tr> <tr> <td>Total manual tests</td> </tr> <tr> <td>% tests automated</td> </tr> <tr> <td>Unit test coverage percentage</td> </tr> </tbody> </table>		Quality and test automation	% SC with test available/tests automated	Defect count at start iteration	Defect count at end of iteration	# new test cases	# new test cases automated	# new manual test cases	Total automated tests	Total manual tests	% tests automated	Unit test coverage percentage
Quality and test automation													
% SC with test available/tests automated													
Defect count at start iteration													
Defect count at end of iteration													
# new test cases													
# new test cases automated													
# new manual test cases													
Total automated tests													
Total manual tests													
% tests automated													
Unit test coverage percentage													
# pushed to next (rescheduled in next iteration)													
# not accepted: deferred to later date													
# not accepted: deleted from backlog													
# added (during iteration, should typically be 0)													

Process Self-Assessment Metrics

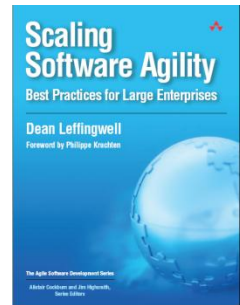


Software Agility Team Self-Assessment	Iteration progress tracked by task to do (burn-down chart) and card acceptance (velocity)
Backlog prioritized and ranked by business value	Work is not added by the product owner during the iteration
Backlog estimated at gross level	Team completes and product owner accepts the iteration
Product owner defines acceptance criteria for stories	Iterations are of a consistent fixed length
Product owner and stakeholders participate at iteration planning	Iterations are no more than 4 weeks in length
Product owner and stakeholders participate at iteration review	Iteration review meeting attended and effective
Product owner collaboration with team is continuous	Team inspects and adapts (continuous improvement) the iteration plan
Stories sufficiently elaborated prior to planning meeting	Total Iteration Planning and Tracking Score
Total Product Ownership Score	Unit tests are written before development
release date	Acceptance tests are written before development
Release review meeting attended and effective	100% automated unit test coverage
Team inspects and adapts (continuous improvement) the plan	Automated acceptance tests
Team meets its commitments to release	Total "Testing" Practices Score
Total Release Planning and Tracking Score	

Process Metrics Radar Chart



Balanced Scorecard Approach



<p><u>Efficiency</u></p> <p>Product ROI</p> <p>Employee turnover</p> <p>R&D cost as % revenue</p>	<p><u>Value Delivery</u></p> <p># releases in last 12 months</p> <p># features in last 12 mos</p> <p># story cards in last 12 mos</p> <p>%sc achieve in last 12</p> <p>Release date % achieve last 12</p> <p># arch re-factors in the last 12</p>
<p><u>Quality</u></p> <p>Normalized defects</p> <p>Normalized support calls</p> <p>Support satisfaction</p> <p>Product satisfaction</p> <p>Escalation rate %</p>	<p><u>Agility</u></p> <p>Product management</p> <p>Release planning and tracking</p> <p>Iteration planning and tracking</p> <p>Agile process</p> <p>Teamwork</p> <p>Development practices</p>

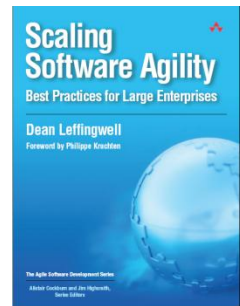
Summary

Stage I Agile Teams

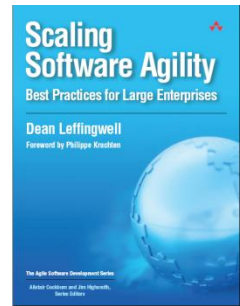
1. Iteration Foundation
2. The Define/Build/Test Component Team
3. Smaller, More Frequent Releases
4. Two-level Planning
5. Concurrent Testing
6. Continuous Integration
7. Regular Reflection and Adaptation

Stage II - 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 Distribution
7. Measuring Business Performance



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 Dean@Leffingwell.org