Scaling Software Agility: Best Practices for Large Enterprises

Agile 101 – Introduction to Agile Development

By Dean Leffingwell
More from Dean Leffingwell


- Blog and Resources
  - www.scalingsoftwareagility.wordpress.com

- Website
  - www.leffingwell.org

- Reach me at DeanLeffingwell@gmail.com
Agenda

- AGILE 101: Introduction to Agile Development
- AGILE 201: Seven Agile Team Practices that Scale
- AGILE 301: Creating the Agile Enterprise
“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
DILBERT by Scott Adams

We're going to try something called agile programming.

That means no more planning and no more documentation. Just start writing code and complaining.

I'm glad it has a name. That was your training.
Dean Leffingwell’s Background

- Agile Transformation Mentor to BMC & Others
- Founder/CEO of Internet Consumer Company ProQuo

- Methodologist/advisor to Rally Software Development

- SVP of Rational Software - responsible for the Rational Unified Process and promulgation of UML

- Founder/CEO of Requisite, Inc., makers of RequisitePro requirements management tool
- Founder/CEO of RELA, Inc., outsourced product development
Why Enterprise Agile?

“Why, When I was your age, sometimes I’ve believed in six impossible things before breakfast”

The Queen to Alice in Wonderland

“There is probably no job on earth for which an ability to believe six impossible things before breakfast is more of a requirement than Software Project Management”

DeMarco and Lister - Waltzing with Bears: Managing Risks in Software Projects
Why Agile

- Promote rapid delivery of value to customers
- Provide timely and regular solution visibility to all stakeholders
- Increased quality, productivity, morale

*Increased software ROI*
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.
Led by Israel Gat, VP of Infrastructure Development
Initial development team size: 300 people
Executive Coaches: Dean Leffingwell and Ryan Martens
Team coaches and agile tooling: Rally Software Development

Results: Sept 2007 Press Release
Agile Excels at BMC, Programming Technique Provides Customer 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

Source: QSM Associates
BMC Results


- 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

Agile Delivers Higher
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

Agile delivers higher quality than anything I’ve found with the waterfall model

Bill Wood, VP, Ping Identity Corp.

Reduction of 1500+ bugs of debt, potentially release-able product every 30 days

Salesforce.com Case study
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
Development teams are more engaged, empowered and highly supportive of the new development process

Jon Spence, Medtronic
Business Benefits

- Forrester Total Economic Impact Studies
  (5 Companies piloting Agile methods)
  - 3 yr, Risk-adjusted ROI of 23% – 66%

- Agile Methodologies Survey, 131 respondents:
  - 93% stated that productivity was better or significantly better
  - 49% stated that costs were reduced or significantly reduced
  - 88% stated that quality was better or significantly better
  - 83% stated that business satisfaction was better or significantly better

1) Forrester Consulting, 2004
2) Agile Methodologies Survey Results, Shine Technologies Pty Ltd, 2003
The Economics of Agile Success

- **Waterfall**
  - Year +
  - Higher Quality
  - Less Risk

- **Iterative**
  - Entire Project
  - Lower Expense

- **Iterative and Incremental**
  - Entire Project
  - Decrease Investment

- **Parallel**
  - Feature
  - One Iteration

- **Acceptance Test Driven**
  - Cheap (Embraced)

Agile Development

Cycle Time & Visibility

Inventory

Change

Risk

Copyright 2008 Dean Leffingwell
What Is Enterprise 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. Smaller, More Frequent Releases
4. Two-level Planning
5. Concurrent Testing
6. Continuous Integration
7. Regular Reflection and Adaptation
Enterprise Agility

A set of

- organizational best practices
- beliefs
- core values

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. Changing the Organization
6. Impact on Customers and Operations
7. Measuring Business Performance
Before: Predictive Process Approach

Predictive, plan-based Process

Plan – measure – re-plan - repeat

Predictive Processes

Waterfall

Predictive vs. Empirical Process

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

Empirical (Adaptive) Process

**Inputs** → **Process** → **Outputs**

**Controls**

Plan – measure – re-plan - repeat
Agile Process Movement

- Often confused with waterfall
- Continued to be artifact heavy
- A stepping stone to agile...
Agile Process Movement

Agile (Adaptive) Processes
- Scrum
- XP
- Lean
- FDD
- Crystal...

Iterative Processes
- Spiral
- RAD
- RUP...

Predictive Processes


© 2007 Trail Ridge Consulting, LLC

Copyright 2008 Dean Leffingwell
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

- Requirements
- Design
- Coding and Unit Test
- System Integration
- Operation and Maintenance

Advantages
- Logical
- Prescriptive
- Plan-based
- Requirements and design “up front”
Assumptions Behind the Waterfall Model

- There exists a reasonably well defined set of requirements, if we only took the time to discover them
- We will limit change and/or change will be small and manageable
- System integration is a stage in the lifecycle and we can predict how that will go
- 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?

Take Action

- Promote the non-participants
- Scope triage
- Create a new plan
- Reduce testing time
- Cut back on quality
- Extend delivery date
- Repeat failed process

Requirements → Design → Coding and Unit Test → System Integration → Operation and Maintenance

Time in months: 0, 6, 12
Wait, it gets worse-

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

![Time Value of Requirements Diagram](image)
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 ever present
   – 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”

4. We **do not** assume that we can develop new, state-of-the-art, software projects on a fixed function/resource/schedule basis
   **Indeed, we assume cannot.**
   – Instead, we assume that we can deliver the most important features earlier, rather than later, than could otherwise be expected
   – In so doing, we can get immediate feedback whether we are building the right solution
“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 Practices

- A team of five to ten programmers work at one location with customer representation on site
- Development occurs in frequent builds or iterations, each of which is releasable and delivers incremental functionality
- Requirements are specified as user stories, each a chunk of new functionality the user requires
- Programmers work in pairs, follow strict coding standards and do their own unit testing
- Requirements, architecture and design emerge over the course of the project
Extreme Programming (XP)

Values & Practices

- Whole Team
- Collective Ownership
- Test-Driven Development
- Coding Standard
- Pair Programming
- Simple Design
- Refactoring
- Continuous Integration
- Sustainable Pace
- Metaphor
- Small Releases
- Planning Game
- Customer Tests
- Small Releases
XP: Collective Ownership

- **Shared Code Base**
  - Team members collectively assume responsibility for system quality
  - Anyone on the team can improve any part of the system at any time

- **Single Code Base**
  - There is only one code stream
  - Temporary branches never live longer than hours
  - Branches sacrifice micro-expediency for macro-expediency

XP: Refactoring and Coding Standards

- **Agile is dependant on refactoring.**
- **Refactoring is dependant on clean code**
- **Far more time is spend reading code that writing code**

Meaningful **names:**
- intention, distinction,
- pronounceable,
- searchable, domain based

**Error handling!**

**Formatting!**

**Functions:**
- Rule 1: Do one thing
- Rule 2: Keep them small
- Rule 3: Make them smaller than that

Even good **comments** don’t make up for bad code

Leave it cleaner than you found it

---

Code quality comments source: Clean Code: A Handbook of Agile Software Craftsmanship, Robert Martin
XP Drives Test-Driven Development

For each new story:

1. Write the test.
2. Run the test and watch it fail.
   a) Test the test itself and test harnesses that hold the test in place
   b) Illustrates how the system will fail if the code is incorrect.
3. Write the minimum amount of code necessary to pass the test.
4. If the test fails, refactor as necessary until a module is created that passes the test.
An Extremely XP Shop

Courtesy of Menlo Software Factory™ @ Menlo Innovations LLC
Ann Arbor, Michigan
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 code, good design, good tooling, OO, testing and test automation…. 
Popular Agile Methods - Scrum

Meeting: *Iteration planning*

Meeting: *Daily Scrum*

Meeting: *Iteration Demo*

* Everything is time boxed *
Scrum – Guiding Principle

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.

![Diagram](image-url)
Scrum Philosophy

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

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

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

**Team**
- Create user stories from product backlog
- Commit to iteration plan
- Define/Build/Test/Deliver stories (fully accepted)
The Power of “ba”

The energy that drives a self-organizing team

- Dynamic interaction of individuals and organization creates synthesis in the form of a self-organizing team
- The fuel of ba is its self-organizing nature a shared context in which individuals can interact
- Team members create new points of view and resolve contradictions through dialogue
- New knowledge as a stream of meaning emerges
- This emergent knowledge codifies into working software
- Ba must be energized with its own intentions, vision, interest, or mission to be directed effectively
- Leaders provide autonomy, creative chaos, redundancy, requisite variety, love, care, trust, and commitment
- Creative chaos can be created by implementing demanding performance goals. The team is challenged to question every norm of development
- Time pressures will drive extreme use of simultaneous engineering
- Equal access to information at all levels is critical
Scrum Practices

- Collocated, cross-functional teams of eight or less develop software in 30 day “Sprints”
- Each sprint delivers incremental, tested, user value
- Work within a sprint is fixed
- 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
- Open, collaborative work environment
- The Scrum Master mentors self-organizing and self-managing teams
A Scrummy Shop
Scrum – Lessons Learned

- Measure and communicate every day
- Extreme visibility drives objectivity, honesty, accountability and performance
- Unleash “ba”, the energy of the self-organizing team
- Team work is facilitated by a Scrum Master
- “backlog” is a simple organizing technique for requirements and priorities
- Product Owner is a critical and underserved role
- Scrum drives change – be prepared
RUP – Iterative and Incremental
RUP Characterizations

- A process framework, not a specific process
  - Tailor to specific project context
- Wide - covers software management, software engineering, software deployment and tooling
- Deep – scales to large scale systems and systems of systems
  - Use case-driven and architecture-centric
  - Has been applied to teams of hundreds and thousands
RUP Best Practices

- Develop software iteratively
- Manage requirements
- Use component based architecture
- Visually model software
- Verify software quality
- Control changes to software
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 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
## Lean Software

<table>
<thead>
<tr>
<th>Lean Principle</th>
<th>Applicability to Agile Software Development</th>
</tr>
</thead>
</table>
| Reduced work in process and inventory | - Reduced investment in elaborated requirements. documented designs  
                                         - Reduced process overhead, compliance checks, audits, etc  |
| Reduced cycle times                  | - Build all software in much smaller lots (chunks, stories, use cases)  
                                         - Deliver smaller and more frequent releases to put all inventory (working code) in the hands of customers |
| Cross-training and cell-bases manufacturing | - Increase cross-training with pair programming and shared code assets  
                                               - Have developers write tests as part of their code  
                                               - Move entire team to test and test automation  
                                               - Collocate all team members to extent practical  
                                               - Entire team commits to delivering the iteration |
| Continuous process improvement        | - Continuous reflection and adaption  
                                         - Self-organizing, self-managing software development teams |
Feature-Driven Development

- Domain Object Modeling
- Developing by Feature
- Class (Code Ownership)
- Feature Team
- Inspections
- Regular Build Schedule
- Configuration Management
- Reporting/Visibility of Results
Dynamic System Development Method (DSDM) Principles

- Active user involvement is imperative
- The team must be empowered to make decisions
- The focus is on frequent delivery of products
- Fitness for business purpose is the essential criterion for acceptance of deliverables
- Iterative and incremental development is necessary to converge on an accurate business solution
- All changes during development are reversible
- Requirements are baselined at a high level
- Testing is integrated throughout the life cycle
- Collaboration and cooperation between all stakeholders is essential
DSDM lessons Learned

- The 80-20 rule works in software

Assumption: Nothing is built perfectly the first time, but a usable and useful 80% of the proposed system can be produced in 20% of the time it would take to produce the total solution

- Forced, rank order prioritization is a key to delivering timely customer value
- Modeling is still OK
Question

- What do all these agile techniques have in common?

Answer:
- Deliver working tested increments in a short time box
Iterating for Risk Mitigation

- Confirm or refute project assumptions
- Forces rank-order prioritization of delivery
- Real time feedback on development efficiency
- Real time feedback on business value
- Optimize the benefit of partial deliverables
- More painless cancellation if value not being achieved

Adapted from *Waltzing with Bears*
DeMarco and Lister
Iteration Pattern

**Iteration backlog**

- Story A
- Story B
- Story C
- Story D
- Story E
- Story F
- Story ...

**Define**

**Develop**

**Accept**

- **Plan**
  - **Fixed Resources**

- **Review**

- **Fixed Time (Iteration)**

- **Story A**
  
- **Story B**

- **Story C**
What's Different About Agile?

Conceptually, agile is simple
Most everything is different
Agile Turns Tradition Upside-Down

Predictive Process (Waterfall)

Constraints

Requirements

Plan Driven

Adaptive Process (Agile)

Cost

Schedule

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

![Graph showing value delivery over time for Agile and Waterfall methods](image)
Delivers Better Fit for Purpose

Measure of waterfall customer dissatisfaction

What the customer would, in the end, like to have

What the initial plan estimated the customer would like

Delivery

following agile (adaptive) planning

Time
What Paradigms Are We Breaking?

**Process**
- **Waterfall**
  - Measure of Success: Conformance to plan
  - Mgt Culture: Command and control
  - Reqs/Design: Big and up-front
  - Implementation: Code in parallel/test later
  - Test/QA: Big/planned/test late
  - Planning/Scheduling: PERT/detailed/fixed scope

- **Iterative**
  - Measure of Success: Response to change
  - Mgt Culture: Leadership/collaborative
  - Reqs/Design: Team based, continuous/emergent/just-in-time
  - Implementation: Code and unit test, deliver serially
  - Test/QA: Continuous/concurrent/test early

- **Iterative and Incremental**
  - Measure of Success: Two-level/fixed date

- **Parallel**

- **Test Driven**

---

Copyright 2008 Dean Leffingwell
Measure of Success

Conformance to plan ➞ Response to change

Work breakdown structure
- Single detailed, complete plan
- Serial functions
- Follow the plan
- Procedural stage gates
- Documents, models, reviews

Feature breakdown
- Two-level plan
- Parallel functions
- Adapt to changing facts
- Time boxes, demos
- Working code
## Management Culture

**Command and control**

Management defines dates and scope  
Management dictates implementation  
Culture of sign-offs  
Protect the scope  
Demonstrate at end  
Weekly status meetings

**Leadership/collaborative**

Teams bid stories  
Team selects approach  
Shared learning  
Protect the date  
Demonstrate always  
Daily stand-up meeting
Requirements and Design

Big and up-front

Marketing requirements up front
Software specification up front
Models and plans
Big design up-front
Architecture is planned

Team based, continuous/emergent/just-in-time

Vision and backlog
Just-in-time elaboration
Build in increments
LRM* design decisions
Architecture emerges

* LRM- Last Responsible Moment
**Code and Implementation**

**Implementation**

<table>
<thead>
<tr>
<th>Code in parallel/test later</th>
<th>Code and unit test, deliver serially</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build in parallel</td>
<td>Build serially</td>
</tr>
<tr>
<td>Hand off to test</td>
<td>Partner with test</td>
</tr>
<tr>
<td>Integrate late</td>
<td>Integrate continuously</td>
</tr>
<tr>
<td>Demonstrate at end</td>
<td>Demonstrate always</td>
</tr>
<tr>
<td>Individual code responsibility</td>
<td>Shared code ownership</td>
</tr>
<tr>
<td>Never miss dev. complete date</td>
<td>Never break the build</td>
</tr>
<tr>
<td>Test code later</td>
<td>Code unit test first</td>
</tr>
</tbody>
</table>
Test and QA

Big/planned/test late → Continuous/concurrent/test early

Contract with customer
Big test plan sign off
Testing at the end
QA is responsibility for test
Testers write all the tests
Testing squeezed
Big stand-alone test teams
Automate tests later

Partner with customer
LRM testing decisions
Test from the beginning
Everyone is responsible
Everyone writes tests
Low features squeezed
Integrated with dev
Automate tests now
Planning and Scheduling

- Detailed planning early
- Measures on intermediate deliverables
- Protect the scope
- Demonstrate at end
- Weekly status meetings

- Detail planning JIT
- Measures based on code
- Protect the date
- Demonstrate always
- Daily stand-up meeting

PERT/detailed/fixed scope  Two-level/fixed date

Planning/ Scheduling
END AGILE 101