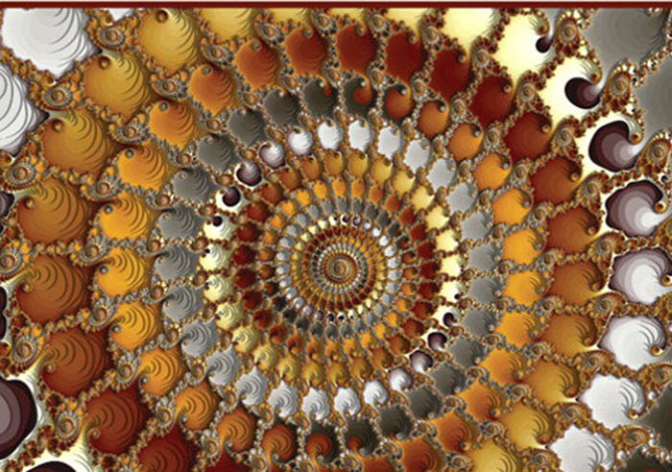


Practices for Scaling Lean & Agile Development

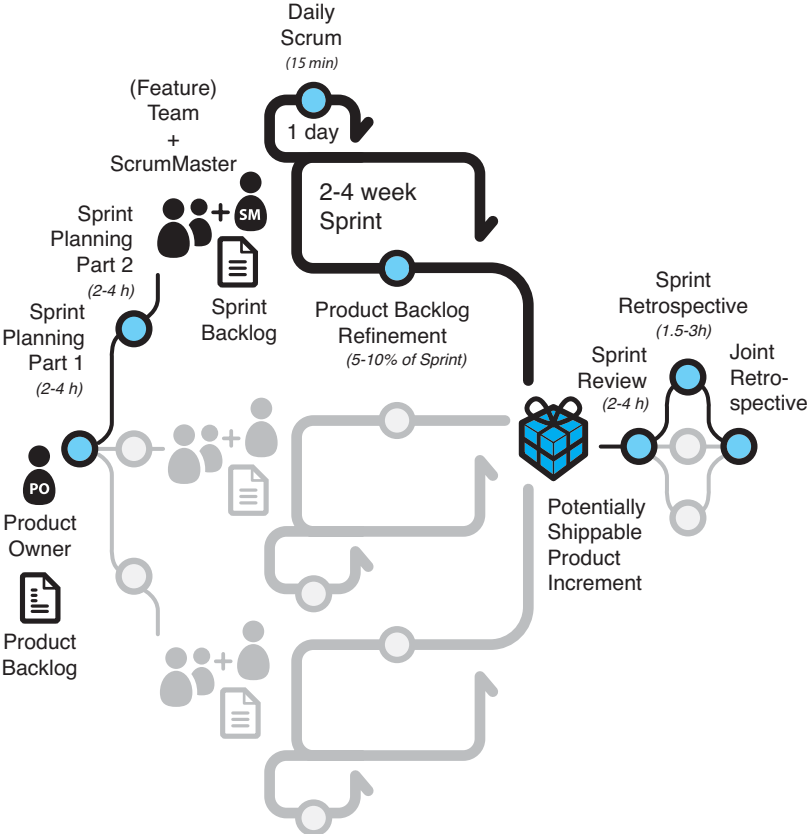


Large, Multisite, and Offshore Product Development
with Large-Scale Scrum

Craig Larman
Bas Vodde

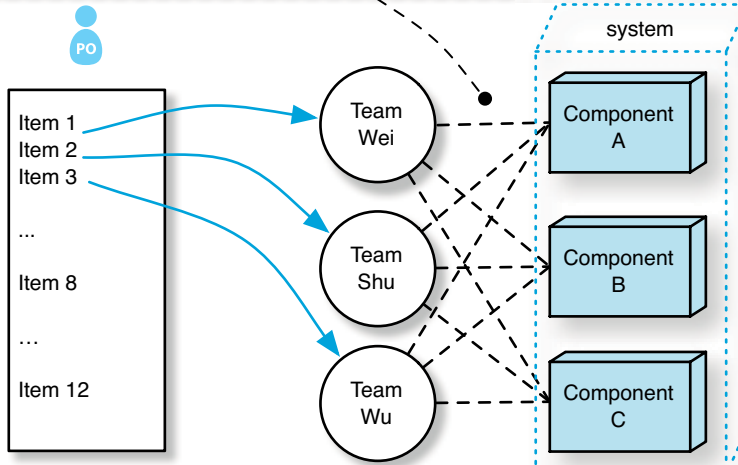


ONE EXAMPLE FRAMEWORK FOR LARGE-SCALE SCRUM



FEATURE TEAMS

With feature teams, teams can always work on the highest-value features, there is less delay for delivering value, and coordination issues shift toward the shared code rather than coordination through upfront planning, delayed work, and handoff. In the 1960s and 70s this code coordination was awkward due to weak tools and practices. Modern open-source tools and practices such as TDD and continuous integration make this coordination relatively simple.

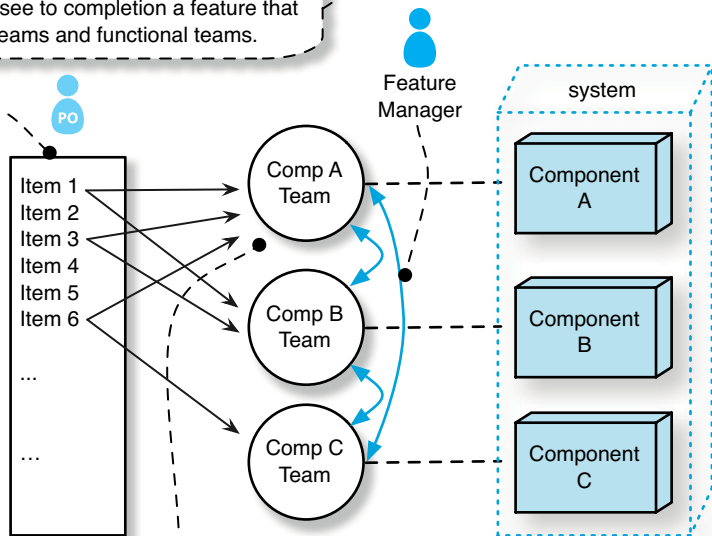


COMPONENT TEAMS

With component teams, a project or feature manager is used to coordinate and see to completion a feature that spans component teams and functional teams.

With component teams, there is a tendency to select goals familiar or 'fast' for teams, not for maximizing customer value. For example, Component B Team does part of Item 3 because it mostly involves Component B work.

This is the "watching the runner rather than following the baton" local optimization.



With component teams, there is increased delay, as one customer feature is split across multiple component teams for programming, and eventually transferred to a separate testing team for verification. There is handoff waste, and multitasking waste—as one component team may work on several features in parallel, in addition to handling defects related to 'their' component.

Practices for Scaling Lean & Agile Development

This page intentionally left blank

Practices for Scaling Lean & Agile Development

*Large, Multisite, and Offshore
Product Development
with Large-Scale Scrum*

Craig Larman
Bas Vodde

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Larman, Craig.

Practices for scaling lean & agile development : large, multisite, and offshore product development with large-scale Scrum / Craig Larman, Bas Vodde.

p. cm.

Includes bibliographical references and index.

ISBN 0-321-63640-6 (pbk. : alk. paper)

1. Agile software development. 2. Scrum (Computer software development)

I. Vodde, Bas. II. Title.

QA76.76.D47L3926 2010
005.1—dc22

2009045495

Copyright © 2010 by Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-63640-9

ISBN-10: 0-321-63640-6

Text printed in the United States on recycled paper at Courier in Westford, Massachusetts.

Second printing, January 2012

To our clients, and my friend and co-author Bas

To Lü Yi, Tero Peltola, and the little one

This page intentionally left blank

CONTENTS

1	Introduction	1
2	Large-Scale Scrum	9
Action Tools		
3	Test	23
4	Product Management	99
5	Planning	155
6	Coordination	189
7	Requirements & PBIs	215
8	Design & Architecture	281
9	Legacy Code	333
10	Continuous Integration	351
11	Inspect & Adapt	373
12	Multisite	413
13	Offshore	445
14	Contracts	499
Miscellany		
15	Feature Team Primer	549
	Recommended Readings	559
	Bibliography	565
	List of Experiments	580
	Index	589

This page intentionally left blank

PREFACE

Thank you for reading this book! We've tried to make it practical. Some related articles and pointers are at www.craiglarman.com and www.odd-e.com. Please contact us for questions.

Typographic Conventions

Basic *point of emphasis* or *Book Title* or *minor new term*. A **noticeable point of emphasis**. A **major new term** in a sentence. [Bob67] is a reference in the bibliography.

About the Authors

Craig Larman has served as chief scientist at Valtech, an outsourcing and consulting group with a division in Bangalore that applies Scrum, where he and colleagues created agile offshore development while living in India and also working in China. Craig was the creator and lead coach for the lean software development initiative at Xerox, in addition to consulting and coaching on large-scale agile and lean adoptions over several years at Nokia Networks, Schlumberger, Siemens, UBS, and other clients. Originally from Canada, he has lived off and on in India since 1978. Craig is the author of *Agile and Iterative Development: A Manager's Guide* and *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design & Iterative Development*.

After a failed career as a wandering street musician, he built systems in APL and 4GLs in the 1970s. Starting in the early 1980s he became interested in artificial intelligence (having little of his own). Craig has a B.S. and M.S. in computer science from beautiful Simon Fraser University in Vancouver, Canada.

Along with Bas Vodde, he is also co-author of the companion book *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*.

Bas Vodde works as a product-development consultant and large-scale Scrum coach for Odd-e, a small coaching company based in Singapore. Originally Bas is from Holland, and before settling in Singapore, he lived and worked in Helsinki (Finland) and Beijing and Hangzhou (China). Much of his recent work is in Asian coun-

tries—especially China, Japan, India, the Philippines, and Singapore—applying agile principles to offshore and multisite development. For several years he led the agile and Scrum enterprise-wide adoption initiative at Nokia Networks. He has been a member of the leadership team of a very large multisite product group adopting Scrum. Bas has worked as developer/architect in multimedia/real-time graphics product development and in embedded telecommunication systems. He is co-author of the CppUTest unit-test framework for C/C++ and still spends some time programming, and coaching agile-development practices such as refactoring and test-driven development.

Bas rushed through his B.S. in computer science so that he could write *real* software. He has been waiting for some university to give him an honorary Ph.D. but is afraid he will actually have to work for it. He is a passionate book collector—especially historical books related to product development and management.

Acknowledgments

Many thanks for the contributions and reviews from...

Peter Alfvín, Bruce Anderson, Brad Appleton, Tom Arbogast, Alan Atlas, James Bach, Sujatha Balakrishnan, Gabrielle Benefield, Bjarte Bogsnes, Mike Bria, Larry Cai, Olivier Cavrel, Pekka Clärk, Mike Cohn, Lisa Crispin, Ward Cunningham, Pete Deemer, Esther Derby, Jutta Eckstein, Janet Gregory, James Grenning, Elisabeth Hendrickson, Kenji Hiranabe, Greg Hutchings, Michael James, Clinton Keith, Joshua Kerievsky, Janne Kohvakka (and team), Venkatesh Krishnamurthy, Shiv Kumar MN, Kuroiwa-san, Diana Larsen, Timo Leppänen, Eric Lindley, Steven Mak, Shiva-kumar Manjunathaswamy, Brian Marick, Bob Martin, Gregory Melnik, Emerson Mills, John Nolan, Roman Pichler, Mary Poppendieck, Tom Poppendieck, Jukka Savela, Ken Schwaber, Annapoorani Shanmugam, James Shore, Maarten Smeets, Jeff Sutherland, Dave Thomas, Ville Valtonen, and Xu Yi.

Current and past Flexible company team members (and reviewers), including Kati Vilki, Petri Haapio, Lasse Koskela, Paul Nagy, Ran Nyman, Joonas Reynders, Gabor Gunyho, Sami Lilja, and Ari Tikka. Current and past IPA LT members (and reviewers), especially Tero Peltola and Lü Yi.

Bas thanks the support of Sun Yuan through another year of writing and traveling. Without her support there would be no book. And thanks Craig for tolerating all the discussion and feedback and... more debugging of Bas's writing. No more "rubber chicken" on this book, what's next?

Craig thanks Albertina Lourenci for the healthy food so that he could write well-nourished, and Tom Gilb for his apartment in London so he could write well-sheltered.

Thanks to Louisa Adair, Raina Chrobak, Chris Guzikowski, Mary Lou Nohr, and Elizabeth Ryan for publication support.

(An Early) Colophon

Layout composed with FrameMaker, diagrams with Omnigraffle.

Main body font is *New Century Schoolbook*, designed by David Berlow in 1979, as a variant of the classic *Century Schoolbook* created by Morris Benton in 1919—familiar to most North Americans as the font they learned to read by, and from the font family required for all briefs submitted to the Supreme Court of the USA.

Chapter

- [Thinking about Adoption & Improvement](#) 374
- [Early Days: Team & Management Changes](#) 391
- [Early Days: Breaking Barriers & Habits](#) 394
- [Early Days: Gatherings](#) 397
- [Coaching & Community](#) 399
- [Continuous Improvement](#) 402

Book

- 1 Introduction 1
- 2 Large-Scale Scrum 9
- [Action Tools](#)
- 3 Test 23
- 4 Product Management 99
- 5 Planning 155
- 6 Coordination 189
- 7 Requirements & PBIs 215
- 8 Design & Architecture 281
- 9 Legacy Code 333
- 10 Continuous Integration 351
- 11 Inspect & Adapt 373
- 12 Multisite 413
- 13 Offshore 445
- 14 Contracts 499

Miscellany

- 15 Feature Team Primer 549
- Recommended Readings 559
- Bibliography 565
- List of Experiments 580
- Index 589

INSPECT & ADAPT

The taxpayers are sending congressmen on expensive trips abroad. It might be worth it—except they keep coming back.
—Will Rogers

We have worked closely in a few enterprise-wide lean or agile adoptions over the years, and have collected experiments. Some, covered later in the *Continuous Improvement* section, focus on scaling and multiteam coordination (such as a Joint Retrospective); many others focus on organizational design and culture. First, a story...

We were coaching in Europe and met with a manager who had been assigned the agile transformation responsibility; he wanted to show us his plan and ask for feedback. He presented a Gantt chart of his planned transformation: many stages of precise duration all in sequence, milestones, specific managers assigned to tasks along the way, cost estimates, and more. According to the plan, in twenty-seven months the group would have transformed to ‘agile.’ The detail was impressive—it was also the wrong approach.

Our colleague had confused *doing* agile and *being* agile. And he was applying command-and-control management thinking combined with predictive planning—in essence, traditional management ‘agile’ adoption. Fortunately, within a few minutes of chatting, the plan was jettisoned and his view shifted to serving the teams, using a backlog, and adaptive planning.

This misunderstanding to agile or lean adoption is common in corporations that (1) *mandate* a top-down ‘transformation,’ (2) think this is another *change project* with an end (“we have now finished chang-

ing to lean—you get the bonus”), or (3) have a centralized group responsible for pushing processes.

Adopt lean and agile principles the same way as applying them: With experiments, adaptation, self-organization, and a focus on the value-add work by applying Go See.

THINKING ABOUT ADOPTION & IMPROVEMENT

Avoid...Adoption with top-down management support

At a time when all of us are struggling to implement lean production and lean management, often with complex programs on an organization-wide basis, it is helpful to learn that the creators of lean had no grand plan and no company-wide program to install it. [SF09]

“Our agile adoption would be so much better if only we had management support.” We have heard that many times, but be careful what you wish for—you might get it! In one enterprise that got official “everyone *do* agile” management support after an informal adoption had been going on for several years, we hear the complaint, “I wish we never had management support; now people are doing things for the wrong reasons.”

Why? In some organizations the culture is

- ❑ management phones in their support but does not deeply learn lean thinking or agile principles¹
- ❑ management ‘drives’ change by setting targets and offering bonuses; in this case, the *agile adoption targets*
- ❑ management directs a centralized process group to “push out the new process”

1. At one of our clients a senior manager asked, “What is the *total cost of ownership* of adopting lean thinking?”

Then, what happens is a superficial cargo-cult agile and lean adoption, with widespread game playing, resentment, hidden resistance, and misunderstanding... another management fad that will pass away if ignored long enough. Perhaps there is a target: “50% of the teams have a ScrumMaster within the year,” and managers get a bonus if that is ‘true.’ Then, existing project or line managers are relabeled as ScrumMasters. Or, “Every product should have a Product Backlog.” The existing work breakdown structure of tasks is copied into a spreadsheet called the backlog. Nothing has really changed, and indeed things may be getting worse because of more disruption and gaming.

Avoid forcing—When coaching we encourage: *volunteering; do not force any agile or lean approach onto people; people should be left the choice to think and experiment.* Create a culture of coaching for those that want to experiment.

Focus—Strive to achieve skill and demonstrate excellence in the adopting groups, with concentrated long-term, high-quality support. The best, most *sticky* adoptions we have seen had this approach.

Try...Adoption with top-down management support

In contrast to the prior case, we have also seen groups with a high-quality management culture that cultivated genuine improvement.

We recall one client (at a bank) where the leadership team quickly dove deep into *reading many books* on agile principles, studied and *applied systems thinking*, all attended a *ScrumMaster training* with their team members, *talked with hands-on experts*, used *agile coaches*, and applied *Go See*. Quickly after starting, this group had made deep changes in organizational design and there was tangible improvement in the flow of value to users.

For ScrumMasters and other coaches the implication is: Only lobby for top-down support when you think the leadership team is seriously interested in learning and in organizational redesign.

Try...Individuals & interactions over processes & tools

One of our colleagues in an agile-coaching group observed, “This company has tried to use processes to compensate for a lack of competence of its employees.”

The first agile value, and the previous story about the effective agile adoption at a bank, reminds us of its veracity—*people, not processes, are the first-order effect* for success [Cockburn99].² A group cannot ‘process’ its way out of a deep hole dug by problems with the individuals in engineering and management—‘agile’ will solve nothing in that case.

So, focus on cultivating and hiring extraordinarily talented people.

But, no false dichotomy... as object-pioneer Grady Booch wrote:

People are more important than any process. ... Good people with good process will outperform good people with no process every time. [Booch96]

Try...Job and personal safety (not role safety)

It is difficult to get a man to understand something when his job depends on not understanding it.—Upton Sinclair

We were in Norway, dining on *lutefisk* with a colleague. He said, “My company has hired consultants for a *lean initiative*. They are identifying redundant employees and firing them.”

This is a perversion of lean thinking. Lean has nothing to do with terminating ‘redundant’ employees, nor with lean-by-consultants. The English name ‘lean’ was *not* chosen to imply removing the *fat* from an organization. Rather, it was chosen³ to contrast *mass* manu-

-
2. An inefficient *process* with large batches, queues, and handoff is itself a major force for failure, but it comes from *people* and their mindsets. Toyota says, “build people, then build products.”
 3. By John Krafcik while working on a graduate degree at MIT; Mr. Krafcik was the first American engineer hired by NUMMI, the Toyota-GM joint venture in California.

facturing with *lean* manufacturing—working in small batches and with less effort to produce goods.

Toyota strives to provide long-term job safety. This is part of the first pillar of lean thinking: Respect for People. And it is also intimately connected to the second pillar: Continuous Improvement. Who is going to strive for continuous improvement in the organization when the likely outcome is job termination? Yet, this does not imply role safety—which inhibits improving the system. For example, project-manager role disappears in Scrum; we have seen people then shift to hands-on engineering or product management.

Personal safety—In Los Angeles one December morning we waited in a room to meet with a team we had been invited (by higher-level managers) to coach for a few weeks. Soon they showed up. We welcomed them and asked, “What are the problems you’d most like to work on? Maybe we can help a little.” There was a long silence—people were uncomfortable to openly discuss problems. So, below the extreme case of job loss, there is the issue of personal safety and improvement. In the *Crystal Clear* agile method, it is identified as one of seven key properties set up by the best teams:

***Personal Safety** is important because with it, the team can discover and repair their weaknesses. Without it, they won’t speak up, and the weaknesses will continue to damage the team.*
[Cockburn04]

A book we sometimes suggest to ScrumMasters (and others) is *The Five Dysfunctions of a Team* [Lencioni02]. The first two of these dysfunctions are *absence of trust* and *fear of conflict*. An improving Scrum team must resolve this. See the recommended readings for material that might help.

Offshoring is another context that we regularly see personal safety problems; a company terminates employees in higher-cost regions and shifts more work offshore. This impacts motivation and collaboration between people in different regions.

In a new large-scale Scrum adoption initiative, ScrumMasters and others need to be mindful of these dynamics: Is Scrum or lean development going to be viewed as a mechanism to ‘streamline’ and terminate overhead? And whereas in little companies active opposition

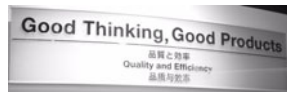
to the system is common, in large product companies there is often a sense of disempowerment and reduced personal safety to challenge the existing organization. Then, for instance, people complain that Scrum Retrospectives are ritualistic, useless, or dead. Or perhaps even worse, people develop a passive-aggressive attitude in response to this ‘streamlining,’ with subtle negative consequences.

It takes active ongoing encouragement from the leadership to keep kaizen mindset alive. As Toyota CEO Katsuaki Watanabe said:

The root of the Toyota Way is to be dissatisfied with the status quo; you have to ask constantly, “Why are we doing this?”
[SR07]

Try...Patience

Toyota has taken *decades* to cultivate a lean culture; similar patience is needed elsewhere. Further, Toyota rapidly expanded in the 1990s and then experienced more difficulty in spreading and sustaining a lean-thinking culture, especially in their satellite plants. It is easy to start losing that culture without ongoing constancy of purpose by lean-thinking manager-teachers [Womack09].



Daily stand-ups and visual management can be installed in days. But it takes years to develop an enterprise of people that know, teach, and apply agile and lean thinking. It is worth it—there lies the great leverage for sustained improvement. Hence the Toyota message, *build people, then build products*.

Avoid...Adopting “do agile/lean”

Be agile rather than do agile was the theme of the *Agile* chapter in the companion book. Agile is not a practice; it is a set of values and principles. Some of the clients we work with misunderstand this and

establish a large-scale transformation project that is measured in terms of *observed practices*, such as,

having a Product Backlog	doing a daily stand-up	working in time-boxed iterations
having information radiators on walls	doing planning poker	writing user stories

To be clear, we recommend trying these practices—indeed, the next suggestion emphasizes that *doing* concrete agile or lean practices is very important. But there is a big difference between a genuine jelled self-managing team that wants to hold a daily stand-up so that they can coordinate, and a group that has been told to have a Daily Scrum—especially if that is on someone’s checklist of “practices in place that prove we are doing agile.”

It is common to find groups where all these practices are *observed*, but where there is only superficial adoption or understanding and little or no *agility*.

Similarly, we recently visited a large outsourcing client in India that was “doing lean.” We asked what that meant. Answer: Using a software tool to measure their WIP levels, and trying to reduce it.

Avoid...Being agile/lean without agile/lean practices/tools

“We understand the Agile Manifesto and lean thinking, and focus on the big ideas—we understand that all practices are just context dependent. And the standard tools don’t work in our context, because we’re different. We have very lean analyst teams, component teams, and test teams, each focusing on their flow.”

In addition to seeing shallow practice adoption, we have seen the opposite: A claim to follow agile or lean thinking but no (or little) application of *any* concrete tools and practices. This is associated with relabeling existing ways of working as agile/lean, when in fact very little has changed or improved.

What happens if there is genuine effort to adopt *many* agile or lean practices or tools? For example, test-driven development, visual management of WIP (perhaps combined with a limited-WIP policy),

reduction in handoff, and more? This *doing* creates a *concrete* framework for learning and kaizen, and a force for deep transformation. Without that concreteness, it is easy to (1) miss subtle insights and context-dependent lessons, (2) miss discovering benefits of these tools, and (3) avoid really improving.

Walk before running

In *Agile Software Development*, Alistair Cockburn [Cockburn07] discussed the *shu, ha, ri* model of stages of skill development in Aikido and its applicability to practices-versus-principles in agile adoption. This parallels the *apprentice, journeyman, master* model. People need to walk before they can run—they cannot become masters without first spending time with tools, mastering them by the book, and experiencing different contexts.

The kaizen cycle starts with learning and applying a standard practice⁴ for similar reasons and because improvement should be against a baseline of insight gained by practice. And there is similar advice in Scrum...

Rule changes should only be entertained if and only if the ScrumMaster is convinced that the Team and everyone involved understands how Scrum works in enough depth that they will be skillful and mindful in changing the rules. [Schwaber04]

No false dichotomy—Principles without practices lead nowhere; practices without principles, theory, and context lead to misapplication and waste. Adopt principles and practices together: thinking tools and action tools are complementary.

Avoid...Agile/lean transformations or change projects

Framing the adoption of lean thinking or agile principles as a transformation or change project leads to the notion

4. Discussed in the *Lean Thinking* chapter of the companion book.

- it is a project, with an end
 - rather than lifelong continuous improvement based on experimentation and growing problem-solving skills
- it is something that people do
 - rather than a change in mindset, culture, and paradigm
- it is something to define and direct by managers

So, rather than framing this as “the agile change project,” experiment with framing it as...

Try...Agile/lean adoption *forever*

One of the pillars of lean thinking is continuous improvement; lean adoption is not a project with an end. Similarly, a group has never finished adopting Scrum; the framework implies inspect-and-adapt every iteration, without stop. Therefore, do not establish an agile change project; rather, build a permanent system for improvement.⁵ And rather than framing what managers do as managing “the agile change project,” experiment with framing what they do as...

Try...Impediments service rather than change management

Sometimes, phrases are influential. Consider the difference between *manage the agile transformation* and *impediments service*.

In the latter case, in the lifelong agile or lean journey (it is not a project), the team members and Product Owner create an *impediments backlog* of their impediments—policies, structures, environment, tools, and more. The role of managers—in the context of agile adoption—is to help the teams and Product Owner by never-ending *impediments service*—working to remove impediments forever.

This change in behavior—and phrasing—is a shift from top-down or command-and-control to bottom-up service.

5. There is an analogy here to the transition from project-mindset to *continuous product development* discussed in the *Organization* chapter in the companion book.

It leads to more Go See behavior by managers and the chance to serve as teachers rather than controllers or planners. For example, many team members will not even realize something is an *organizational design* impediment; lean-thinking manager-teachers have an opportunity to help them learn to see this.

Iterative and adaptive; pull from the backlog—This is also a shift from predictive to adaptive planning. In this model, agile adoption is based on (1) a prioritized impediments backlog, (2) short impediment-service cycles⁶ executed by managers, and (3) *adaptive iterative planning* based on a re-prioritized backlog each cycle. Who knows what will be done in the next impediments-service cycle?—As with Scrum, the impediments backlog is emergent and continually re-prioritized.

There is no predictive planning, schedule, milestones, targets, or Gantt chart with the “agile adoption schedule.” Rather, Scrum and agile adoption is *iterative and adaptive*, just as regular agile development.

Prioritization and impediments backlog owner?—An official backlog owner is probably not needed. Instead, experiment with this: Every team, when they add an impediment to the backlog, give it a priority. Then, prioritize based on (1) number of teams that raise the same impediment, and (2) average priority of the impediment.

Avoid ‘impediments’ added from quality and management areas—Some years ago, in China, we were coaching a Scrum-adopting product group that had an impediments backlog. All the original impediments came from hands-on workers. But after some time, *quality managers* and department managers started to add their own ‘impediments.’ These were not impediments of flow of value to customers, nor impediments from the value-worker viewpoint; rather, they were ‘impediments’ such as “not conforming to centralized pro-

-
6. As in Scrum-for-development, some management groups use *time-boxed* cycles to improve cadence, to address the Student Syndrome problem, and to motivate splitting large impediments into smaller ones—with smaller incremental solutions. But do not assume all the practices of Scrum (such as timeboxing) will successfully apply in non-development contexts, such as this.

cess practice <X>.” Avoid that; the important work is the value stream of the teams and Product Owner, and removing their impediments. All that said, ...

Avoid ‘impediments’ added from hands-on workers—If you ask a typical existing team of testers or a component team, “What is the best team structure?” They will say, “Our current structure, of course!” It is common that people—arguably even more so in non-management positions—have *not* developed systems-thinking or lean-thinking skills, nor have they studied organizational design, team, or product-development research. Toyota (and management thought leaders) have emphasized the vital role of managers who have this kind of knowledge, educate others, and improve the system with insight. Suppose there was a recent shift to feature teams and early testing, and then ex-test-team members added an ‘impediment’ to the backlog: “the testers should be in a separate group, and avoid testing early so that it can be done efficiently at the end.” ScrumMasters and manager-teachers have a responsibility to debug these local-optimization thinking mistakes, and clarify problems that genuinely impede the flow of value. It is easy to fall into the trap of local suboptimization thinking—*watching the runner rather than following the baton*, forgetting gemba and Go See. We make this mistake too. Testing our ideas against people educated in systems thinking can help.

Managers add system impediments—Building on this last point, there are system weaknesses to the value stream (usually in policies and organizational structure) that team members are unlikely to grasp or consider candidates for change. Managers have a pivotal role in identifying and removing these. The *Organization* chapter in the companion book centered on these weaknesses.

Add impediments from the Product Owner and product management—The value stream is within the teams *and* in the work of the Product Owner and product management. Invite product management to impediments backlog workshops.

Accept the priority given by the hands-on workers—At one of our clients in Greece, we facilitated an initial impediments backlog creation workshop with team members. After all the voting, what was their number one impediment?—A slow network. For years that had been the dominant issue (it inhibited integration, for instance), but no one in management had done anything about it—the priority of

this and other impediments had never been *this* clear. Now, with a list of 50 prioritized impediments, the number one issue was unambiguous. To their credit, the management team—that had agreed to move to the model of impediments service—accepted its priority and within a few months, problem solved. This also built trust and cooperation because the teams saw managers genuinely helping solve their key problems.

Create the initial impediments backlog in a workshop—We have helped set up many impediment services for management teams, and have found the following approach useful to start it off:

1. Convene a workshop with hands-on people from teams, the Product Owner, and other product managers. In



other words, focus on *gemba*—where the value work is. Start with *brain-writing* impediments on cards, in pairs.

2. Next, form larger groups from four or five pairs. The groups discuss, merge, and refine the impediments into a new set of cards. Use the floor.



3. Combine the refined cards from all groups into a central floor area. Do *affinity clustering* to group them. Remove duplicates. Then, do *dot voting* by all participants. Finally, lay out all the cards in (dot voted) priority order. Discuss and refine—final tweaking.





4. Use visual management. Set up the backlog on a wall outside the office of a senior manager. (This photo shows a day-one backlog with no ‘service’ yet). For example, in Greece it was set up near the office of the head of the development center. During impediments-service Sprint Planning, or at other times, managers volunteer for an impediment, write their name on the card, and move it to the middle WIP column.

Rather than “manage agile transformation,” help agile-adopting teams and product management with *impediments service*.

Try...Human infection

Thinking and acting outside the box is possible but hard when everyone is inside it. Lean thinking, agile principles, self-organizing teams, test-driven development, feature teams, manager-teachers... these are mindset, culture, and behavior changes—and to be sticky or meaningful, these kinds of changes require *human infection* from experts through long-term face-to-face coaching.

In the most successful adoptions we have seen, the organization established internal coaches supplemented with external coaches (both were important), and emphasized lots of hands-on mentoring from these agents-of-change during the real work.

Avoid...Agile/lean adoption targets or rewards

Rewards work. An economist wrote in his blog a story to prove this: His son still wore diapers to bed each night. The economist told his son, “If you don’t wet your diapers tonight, tomorrow I’ll buy you the toy you want.” The next morning, the father went into his son’s room. His son had successfully fulfilled the goal and was looking forward to the reward. He had removed his diaper the previous night. The *bed* was all wet, but his *diaper* was dry.

The *Organization* chapter of the companion book summarized the hard facts that performance-based incentives lead to gaming, opacity, and a weakening of the system. We have seen their deleterious effect in promoting “fake agile” adoptions in several groups. Avoid that—and avoid “agile adoption” target setting. The quality guru W. Edward Deming, in his *14 points for management* [Deming82], summarized this in number...

11. Eliminate management by objective. Eliminate management by numbers, numerical goals. Substitute leadership.

Avoid...Competitive ‘improvement’

At some clients we have worked with, the introduction of kaizen gets mixed up with their prior management culture, such as competitive incentives. Then, teams or individuals are offered rewards if they improve more than others. This leads to a competitive rather than cooperative culture, in which parties are less willing to share or help others since they might ‘lose’ individually.

Avoid...Try...‘Easy’ agile or lean adoption

‘Easy’ agile adoption is an existing weak organizational design not meaningfully changed, and a thin veneer of practices painted on: managers relabeled as ScrumMasters, existing component/analyst/testing teams get their own “Product Backlog” and hold a daily stand-up meeting *every week*, and more. There is no significant improvement, and people do not take continuous improvement seriously—or worse, they think, “the agile adoption is finished.”

On the other hand, Scrum emphasizes the *art of the possible*. It may be that minor modifications are the current limits of change because of limits in mindset.⁷ These will not meaningfully enhance the value flow to customers, but perhaps (1) adding prioritized backlogs, (2) working in short timeboxes, (3) lowering WIP, (4) holding standups,

7. Sometimes, people have invested years in sequential life cycle processes and the existing team structures; they will not easily consider the possibility it was not ideal for flow of value.

and (5) reducing multi-tasking will help *fractionally*. It is a first step before deeper change and improvement. Then, we suggest...

If you're going through hell, keep going.—Winston Churchill

Try...Experiment rather than improve

The mandate to *improve* is a lofty goal, and can scare off people from experimenting. What if the *improvement*...doesn't? Kaneyoshi Kusunoki, a student of Taiichi Ohno and executive vice-president at Toyota, said about kaizen and management support:

A defining characteristic of the corporate culture at Toyota is that managers don't scold you for taking initiative, for taking a chance and screwing up. Rather, they'll scold you for not trying something new, for not taking a chance. Leaders aren't there to judge. They're there to encourage people. That's what I've always tried to do. Trial and error is what it's all about! [SF09]

Developing problem-solving skills through many experiments is central to lean thinking. The only bad experiment is the one not tried!

The real measure of success is the number of experiments that can be crowded into 24 hours.—Thomas Edison

In this light, the *Try...* and *Avoid...* ideas in this and the companion book are just experiments—and also because systems are too complex and variable to assume prescriptive advice will work.

Try...Encourage experiments; offer coaching

Avoid...Forcing adoption of practices

The mandate “adopt agile development” is daunting and large. The mandate “do continuous integration” reflects command-and-control, forcing practices. An alternative to both these approaches is to foster the kaizen mindset encouraged in lean thinking: People are encouraged to experiment and are supported with coaching and education. For example, a ScrumMaster can explore with teams the problems associated with delayed integration, describe continuous integration as an alternative, and arrange coaching if the teams want to try it.

Avoid...Adopting <X> because “agile didn’t work here”

Survey decades of management and product-development trends, and some patterns emerge. Possibly the dominant one is

1. difficulties exist due to system weaknesses in organizational design, poor engineering skill, and ineffective management
2. try new ‘thing’ to address a problem (insert: MDD, PMI certification, Kanban, CMMI, Scrum, SOA, agile, next-generation lean, ...)
3. do not address the systemic issues; try ‘thing’ superficially
4. after two years, abandon ‘thing’ because “it doesn’t work here”
5. go to step 2

We see this in some groups trying Scrum. Scrum is a simple framework that acts as a mirror: Rather than fixing problems, it increases visibility of systemic weaknesses, inviting inspect-and-adapt with experiments. In some groups, rather than fixing the system, it is easier to try the next thing... “Let’s call in new consultants specializing in Scrum failure, and then adopt...next-generation lean.”

Avoid...IBM/Accenture/... agile adoption

This is not about IBM or Accenture per se; it is about

- ❑ the misconception that agile is a process or practice
- ❑ shifting responsibility for agile/lean success to an external consulting group

From this stems the notion it can be bought and installed—and there are companies happy to take your money claiming so. Plus, it is related to the misunderstandings summarized in the *False Dichotomies* chapter of the companion book: *agile means iterative development*, *Scrum means daily stand-ups*, and so forth.

Avoid...Adopting agile with “agile management” tools

“We’re starting to do agile. What tool should we buy for agile project management?” This is a question we hear often; our suggestion is always the same—and we mean this even for the very large-scale cases: “Avoid any special agile tools until several years after starting the adoption. Keep it simple. Use the wall or, in the most complex solution, a simple spreadsheet and wiki.” Why?

Problems from system weakness cannot be solved with processes or tools. Worse, attempting to *quick fix* systemic problems with tools creates an illusion of control or change but no real improvement... Executive: “What is the agile transformation progress?” Agile-change manager: “We have installed <AgileToolX> and three of the projects are using it. Come take a look at the burndown charts...”

Avoid the lure of “tools to do agile management” for at least several years after starting to adopt agile or lean development, so that *people’s focus* can be where it belongs: on the *system*. By removing all crutches and quick-fix illusory solutions, people may—just possibly—be prompted to squarely face the important but hard issues: competent individuals, interactions, organizational design, the illusion of command-and-control, and so on.

If you automate a mess, you will get an automated mess.—anon

We are not suggesting agile-management tools are poor—or good. This is about focusing on important things first and avoiding the dysfunctions that accompany management-reporting tools.

After <N> years? Prefer free tools so that the cost of experimenting is low and there are fewer barriers to discarding tools. We have heard the following several times: “We can’t stop using tool (or process) X because we have invested so much in it.”

We have seen thousand-person multisite development groups successfully apply large-scale Scrum with some Excel spreadsheets for their Product Backlog and Release Burndown chart. Indeed, they are almost certainly better off for doing so; it keeps their attention more on fixing the system.

Also, there is a more subtle, pernicious danger with agile-management tools. These are the fifth and eleventh agile principles:

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

A theme in Scrum (and other agile methods) is self-managing teams, as covered in the *Teams* chapter in the companion book. And the fifth principle emphasizes *trust* and *support*, which is quite different from monitoring people's work. So what?

The agile-management tools we have seen emphasize tracking and displaying individual and team tasks and Sprint Backlogs to managers—almost the antithesis of these principles. In Scrum, the team's tasks (the Sprint Backlog) are created by the team to help them self-manage, not to report their status to others. As the well-known team researcher, Richard Hackman, explains, “In self-managing teams, the responsibility of tracking the progress is delegated towards the team” [Hackman02]. Since the team is self-managing, they are not to be tracked or monitored; such tools are a slippery slope that may reinforce a traditional command-and-control culture rather than a culture of self-management.

We know a coach who works for an ‘agile’ tool vendor. He told us that they had been joking about adding a “*real Scrum*” button to their tool. This button would turn off all the non-Scrum and unnecessary features that were requested by their traditional-management clients...and there would be almost nothing left in the tool.

There is a well-known case of a company where project managers inspected daily the Sprint Burndown charts of teams, and “solved the problem” when the charts did not go down. Ken Schwaber—the Scrum co-creator—was visiting and noticed that all the burndown charts had almost no deviation between the burndown and ideal lines. Eventually he discovered that a team kept two charts: a fake one for the managers so that they would stop interfering, and a real one to support self-management.

Computerized management-reporting tools can also take people away from *gemba* and the practice of *Go See*. Lean thinking emphasizes—to understand what is really happening—go with your feet and see with your eyes at the real place of work, help solve problems there, and build relationships with the workers there.

Finally, these tools are optimized for *reporting*—not for success, improvement, or a better flow of value. What *meaningful* problem do they solve?

EARLY DAYS: TEAM & MANAGEMENT CHANGES

Try...Transition from component to feature teams gradually

Over the years that we have been involved in the transition to feature teams from component teams (in large groups involving hundreds of people), we have seen several strategies—and not always smooth. In *Feature Teams* in the companion book we shared two:

- big-bang reorganization
- gradual expansion of component teams' responsibility

The first strategy can work better than one might expect, but not many organizations want to take that plunge because the change is big and they consider it risky. Plus, it *is* a challenge in a 20-year-old multisite product group with 100 long-established component teams. The second strategy does not work that well, because it creates both the drawbacks of feature and component teams.

Another strategy we have experimented with (not described in the companion book) is the gradual introduction of feature teams, *applied only to the most important new customer features*.

For instance, take the most important new feature, *item-1*. Form one new cross-component and cross-functional feature team, *Team Red* (Figure 11.1), by extracting only a few members out of existing component and single-function teams (such as analysis and testing). The old teams remain, slightly smaller, and Team Red is born: starting life by working on *item-1*. In this way, new high-value work benefits

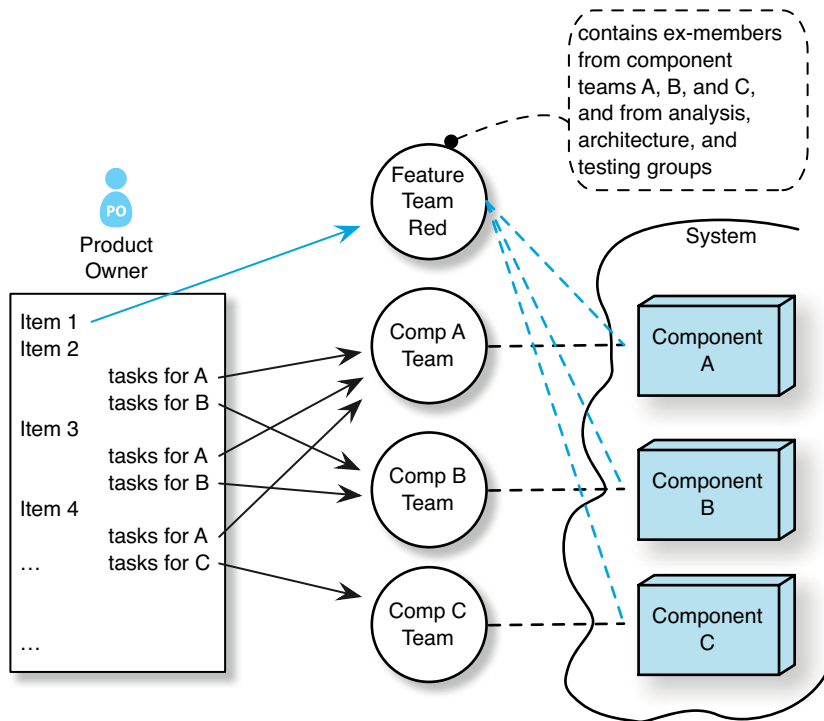
see the Feature Teams Primer chapter

“Try...Component guardians for architectural integrity when shared code ownership” section on page 314

“Try...Component mailing lists” section on page 314

from the speed and simplicity of feature teams, while change impact is softened.

Figure 11.1 a gradual transition from component to feature teams, focusing on the most important features



stable teams: see the Feature Teams and Teams chapters in the companion book

Note!—Team Red is not a temporary project group formed only for the purpose of feature-M. We are not suggesting the traditional practice of resource management with resource pools for short-term work groups. Rather, Team Red is a new *stable team* that will stay together for years; feature-M is but the first of many features they will eventually do.

Disadvantages—This approach also has drawbacks. The first, broadly, is conflicts caused by having two ‘competing’ organizations in place at the same time...

- ❑ feature teams change code that component teams own
- ❑ the analysis and architecture groups lose ‘control’ over deciding how to implement a feature, and the test group over the testing

The second drawback is that this approach is slow—not a major problem for big product groups that are around for a long time!

Avoid...Waiting for the organization chart

Official agreement on changes to the organization chart for a reorganization to cross-component and cross-functional teams can take a *long* time—especially in long-established large groups. In the groups we work with, the successful strategy is to not wait for that, but to immediately and informally create new cross-functional Scrum teams by dispersing the old teams. The existing line managers (say, a test manager) then have people ‘reporting’ to them from multiple teams. Usually, after some months, the organization chart catches up.

What about the prior line managers, such as the test-group line manager? They may become line managers of several new cross-functional cross-component Scrum teams.⁸

Avoid...In-line ‘ScrumMaster’ line- or project managers

Before adopting agile development, most groups had project managers or line managers. In some, during early days of agile adoption, rather than supporting the emergence of self-managing teams (the 11th agile principle) with a real ScrumMaster, the managers relabel themselves ScrumMasters of their in-line teams—often to meet a top-down target to do Scrum. Avoid that, since a ScrumMaster is not the team’s line or project manager and has no authority over the team they serve; there would be a conflict between having authority and no authority.

see “Avoid...Fake ScrumMasters” in the companion

“Avoid...ScrumMaster coordinates” section on page 197

On the other hand, some line managers can serve as excellent real ScrumMasters—they may have the right skills and servant-oriented character, they may have some influence in the organization, and this role increases their focus on improving the system. How to

Try...Line manager as ScrumMaster of out-of-line team

8. This assumes that the new teams report to a line manager, which is not required by law nor in a self-managing organization; see the recommended readings in the *Organization* chapter of the companion book for companies that do not organize in a hierarchy.

resolve? In some groups at Xerox, for example, a line manager of team-A *offers* to serve as a ScrumMaster for *out-of-line* team-B; team-B decides on the offer. The two points are (1) it is an out-of-line team, and (2) ScrumMasters are chosen by the team, not imposed.

EARLY DAYS: BREAKING BARRIERS & HABITS

Try...Break the walls—team areas with whiteboards

ScrumMasters remove barriers for teams. At Valtech India, when we saw the cube farm on the left, we arranged to gut the interior of the building, and create team areas with plenty of whiteboards.

before



after



Try...Two-week iterations to break waterfall habits

Although Scrum allows iterations of up to four weeks, this is seldom advised or practiced. The *Scrum Guide* suggests:

Tip: When a Team begins Scrum, two-week Sprints allow it to learn without wallowing in uncertainty. [Schwaber09a]

When we started coaching large-scale groups in Scrum, we assumed that four-week iterations would be useful to gradually “lower the waters in the lake.” What we discovered, however, was that four weeks is *just* long enough to maintain old habits: sequential life cycle practices, the existing single-function teams, and handoff between groups. Consequently, there was no strong force for out-of-the-box thinking or transformation to a profoundly different organizational design with concurrent engineering, continuous integration, feature teams, and so on.

But, two-week iterations—with the goal of getting items really done according to *done*—do not readily allow for old habits. Things have got to change—dramatically.

A similar suggestion, for other good reasons, is found in the first book on scaling agile development:

*Although you may have heard otherwise, the larger the team is, the more important **short** cycles are. The reason is simple—if a large team takes a completely wrong course from the entirety of its three-month development cycle, the cost of correcting the course will be enormous. And even if the team took the correct course, it wouldn't benefit from the frequent feedback that is possible with short development cycles. [Eckstein04]*

Try...One flip chart for tasks of one Product Backlog item

Figure 11.2 shows a common-style Sprint Backlog, with one row of task cards for each Product Backlog item, and three columns: *to do*, *underway* (meaning, WIP), and *complete* (meaning, done).



Figure 11.2 Sprint Backlog—rows for each item, columns for *to do*, *underway* (meaning, WIP), and *complete*.

In the early days of a big-group adoption, a coach will notice—by looking at this display and in the behavior of the team—two symptoms of old habits:

- Many tasks cards at the same time are in the *underway* column—there is high WIP.
- Key point—task cards for *multiple backlog items* are in the WIP column *because people are thinking “I only do **my** special tasks.”*

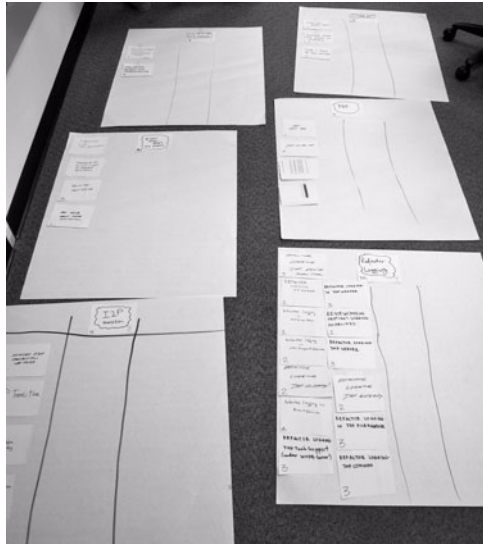
For example, “I am an interaction designer. I have finished *my* interaction design tasks for item-1. Therefore, no more tasks for me in item-1, so I will start on *my* interaction design tasks for item-2.”

Team members have *primary specialities*, and will do tasks in those areas, but when those are finished, the idea is for team members to take on other tasks *of the current item in progress*, in less familiar areas—perhaps in an area of *secondary speciality*. This both reduces WIP and increases multi-area learning.

A visual management technique to encourage this is illustrated in Figure 11.3. Now, the Sprint Backlog is spread across a set of flip chart posters. Each Product Backlog item has task cards on a separate poster—and each poster has the three common columns: *to do*, *WIP*, *done*. Now—key point—the team displays only one or two posters on the wall at a time;⁹ the other posters (items) are out of sight. Then, the *whole team* focuses on getting one item at a time *done*, increasing learning and reducing WIP.

9. Two items may be in progress either because each is so unusually small that the entire team cannot realistically work on one item together or because something is blocked.

Figure 11.3 one flip chart for each item



EARLY DAYS: GATHERINGS

Try...Repeating large-audience introductions

When there are tens of thousands of people in a company, it is useful to convey a consistent introductory message to everyone. One technique is written material, but that is low-impact—few read it, and the nuance of “bringing Scrum to life” is lost.

Frequent one-day large-audience seminar introductions (say, 200+ people at a time) make a bigger impact—due to immediacy, Q&A, and especially the many ‘discussions’ that take place during coffee and lunch breaks. These seminars *break the ice* and *add some steam*.

Try...Open-Space Technology for early-days adoption



From India to Hungary to the USA, we have seen the positive impact of using Open Space Technology (OST) [Owen97] during the early days of large-scale Scrum adoption within groups. We usually serve as facilitator, starting by announcing the

theme of “agile adoption at companyX,” explaining the time-space board, and briefly sharing the OST principles and laws.

OST is a meeting technique that encourages emergence and self-organization; it is highly complementary to agile principles and Scrum, and we encourage groups to experiment with it in multiple contexts: early days, Scrum-of-Scrum meetings, and more.

Figure 11.4 OST early-days agile adoption events: Budapest and Bangalore



Try...Big gatherings to share stories & experiments



During the first few years of Scrum adoption at one of our clients, we helped organize an annual internal Scrum Gathering in which

hundreds of people from around the globe came together to share stories and tips, listen to expert speakers, and so forth. This sustained and added momentum to the adoption.

COACHING & COMMUNITY

Try...Central coaching group

In some of the enterprise-wide adoptions that we have seen, an internal agile or lean coaching group was established, consisting of hands-on agile experts who go and work with directly with teams. Try that.

Form a cross-functional coaching group to learn the diversity of perspectives and issues and to build support for change in more diverse areas. For example, include product management, software development, hardware development, field service, sales, manufacturing, marketing, and more. That said, in the early days of adoption, the focus is typically within R&D and product management, so the original scope of coaches is usually limited to these areas.

Avoid...Central coaching group with formal authority

Caution—Avoid a group that has formal authority to mandate practices, policies, and processes. Rather, create a group that focuses on coaching people interested in adopting agile or lean development.

Try...Concentrate the coaching on a few products

Genuine learning and change of behavior within a product group takes a lot of coaching and time. Plus, misunderstandings are easily created without sufficient coaching. We have seen product groups flounder because they received only a smattering of occasional education. It is better to concentrate the attention of the internal coaching group—supplemented with external coaches—on a few products. Only move on to new groups after solid mastery in old groups.

Try...External agile coaches

Good external agile or lean coaches are worthwhile because they bring fresh perspectives and ideas, sometimes have more credibility than internal coaches (even if not justified) and can therefore make a quicker change-impact, and they can “speak the unspeakable.” Also, ...

Try...Pair external agile coaches with internal ones

When external coaches visit, pair them with internal coaches. There are several advantages, including

- ❑ *learning from each other*—for example, the external coach will learn things about the enterprise—policies, politics, and so forth—that would otherwise be difficult or slow to grasp
- ❑ *increased learning in the broader coaching network*—the two coaches connect each other to broader networks (internal and external) which share and learn from one another

Avoid...Advisors/consultants who are not hands-on coaches

Big companies often have a centralized process or improvement group. The people working in this area sometimes drift away from doing hands-on development and become *PowerPoint process consultants*. Avoid people like that in an agile or lean adoption initiative. Similarly, watch out for consultants or coaches who may not have read the foreword to the four agile values:

*We are uncovering better ways of developing software **by doing it and helping others do it.** (emphasis added)*

Some ‘agile’ consultants do not directly develop software with the teams—coaching agility and lean thinking at *gemba*. Rather than *doing it* with hands-on developers and practicing Go See, they sit in rooms presenting or reviewing process diagrams that may have little to do with what is really happening, or they write emails speculating about problems and their solutions. Managers and consultants may be pleased with the *agile PowerPoint process*, but the reality on the ground is different.

Instead, develop a cadre of internal and external agile/lean coaches who apply Go See and who are masters of the real value work (programming, testing, ...). These coaches and consultants spend most time with engineers while coaching, and only occasionally leave *gemba* to meet with senior management—bringing their insight of what is *really* happening at *gemba*.

Try...Structured intensive curriculum for all teams

For example: At one of our clients the focus is on lean development plus agile engineering practices. In collaboration with management, we set up (and coached) the following curriculum for development people (organized by team). There are intervals of several weeks to several months between each step:

1. Short warm-up e-learning (web-based) courses that focus on basic concepts and terminology related to lean thinking.
2. Lean development-1 (LD-1): Five days in classroom with class projects, with an emphasis on hands-on doing.
3. LD-2: Five days in a structured workshop with teams, applying the skills from LD-1 to their real products, and learning some new skills. A coach mentors. The workshop is in a separate location from their normal work environment.
4. LD-3: For five days, a coach visits the team at their normal work area, reinforcing LD-1 and LD-2 skills in the context of their day-to-day work, doing pair work, and facilitating workshops (such as Sprint Planning).
5. LD-4: Same as LD-3.

Thousands of people are involved in this multiyear coaching endeavor, and the leadership's commitment to in-depth meaningful lean and agile coaching is an illustration of the foundation of the Toyota Way: manager-teachers who have long-term constancy of purpose with lean thinking.

Avoid...Internal agile/lean cookbooks

“Let's write an internal agile cookbook so that all the people can better adopt agile development in our company.” It sounds like a good idea: more efficient, more harmonized, ... But we have seen—through Go See with the teams—the subtler dynamics at play...

- ❑ It reduces critical thinking—people assume that if something is written in a corporate-sanctioned guide, then it is good.

- ❑ It reduces challenging the status quo—people assume that what is written in corporate guides should be accepted or followed, rather than challenged.
- ❑ It reduces learning, especially *good* agile/lean learning—high-quality agile, lean, and Scrum teachings have been written in *books* by founding thought leaders; but rather than study these *original sources* for good learning, people assume that secondary corporate guides contain reliable insight.
- ❑ (Related to prior point) it increases misrepresentation—in the interest of ‘harmonization,’ internal process writers *revise* these systems... “let’s remove *self-organizing teams* from our agile description—people won’t like that.”
- ❑ It reinforces the corporate illusion that system problems can be solved with processes and process documentation.
- ❑ If there is an internal group that only writes documentation, and the people in this group do not do hands-on agile coaching, then (1) what is written is undesirable because it is not based on experience, and (2) it perpetuates more overhead work away from gemba.

A group at Toyota described their early documentation effort, and what Taiichi Ohno thought of that:

So we went to work on preparing a systematic description of our [Toyota] production methodology. ... Ohno, of course, hated that kind of deskwork. If he saw people poring over written work like that, he’d tell them to get out onto the plant floor. So the team couldn’t do its work within his sight... [SF09]

CONTINUOUS IMPROVEMENT

This section has two categories:

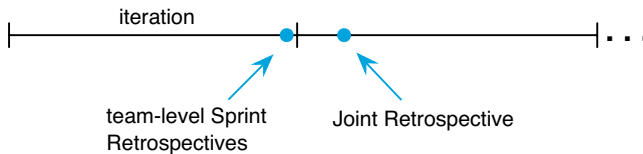
- ❑ multiteam coordination, such as a Joint Retrospective
- ❑ other general experiments

Multiteam Coordination Experiments...

Try...Joint Sprint Retrospectives

An iteration ends with an individual team Sprint Retrospective, where the focus is team-level improvement actions. In large-scale Scrum there is the bigger system to inspect and adapt. For this, experiment with **Joint Retrospectives** each iteration.

When?—Since the iteration ends with a team retrospective, most of our clients hold this early in the first week of the subsequent iteration—when the issues of the previous iteration and recent team-level retrospectives are still fresh in mind.



Who?—In general, one or two representatives from each team. Since ScrumMasters are closely involved in understanding and helping improve the system, they are good candidates. However, avoid ScrumMaster-only meetings; this gives the wrong impression that ScrumMasters are solely responsible for improvement (rather than other team members too), and it increases bias during the workshop.

Scope of teams?—This depends on the scale: If there is only one small 10- or 20-team group at one site, one Joint Retrospective with representatives from all teams suffices. If it is larger and there are *requirement areas*, then each area is a good scope for a retrospective. Because many issues are site specific, a site-level retrospective is also useful: one in Curitiba, one in Chengdu, and so on. Finally, for larger groups, experiment with a top-level Joint Retrospective (above the site and requirement areas); in this case, it is most often a multisite retrospective.



Where?—Use a *big* room, with lots of whiteboards since there may be dozens of people in a Joint Retrospective. See the *Multisite* chapter for tips in that case.

“Try...Requirements workshops for Product Backlog refinement” section on page 243

How?—As with any retrospective, *variety* of workshop activities over time is a guiding principle. Broad suggestions:

- ❑ Try Open Space Technology [Owen97], World Café [BI05], and Future Search [WJ00] for Joint Retrospectives.
- ❑ Apply the *diverge-merge* pattern—useful in any large workshop.

“Try...Coordination working agreements” section on page 212

What?—Too often, a retrospective focuses only on problems. Experiment with sharing what is going *well* for a site or team, that others may try. This is the *yokoten*—spread practices laterally—approach used at Toyota. A joint retrospective is also a time to review and change existing *coordination working agreements*.

Try...Joint Retrospective big improvements in Product Backlog

Major (expensive) improvement ideas are added to the Product Backlog so that they are visible to—and prioritized by—the Product Owner. This is even more important when there are intermediate Joint Retrospectives below the overall product level. For example, suppose there are 20 teams in Curitiba (Brazil) and 20 teams in Chengdu (China). Each sub-group holds its own site-level retrospective and identifies the same major improvement goal. These need to flow into a common list, the backlog, to prevent duplication and so that the Product Owner sees cross-site problems.

And who takes on this work? An existing feature team.

Note—This relates to other suggestions in this and the companion book. If the improvement goal involves common software, this leads to a feature team working on shared infrastructure (see *Feature Teams* in the companion). If it involves creating common test-auto-

mation testware, this leads to a feature team doing test automation (see the *Test* chapter).

Try...Cross-team working agreements

External-to-team **working agreements** usually define how teams agree to work together; for instance, holding a *joint design workshop*. They may or may not be product-wide; a subset of teams that work together frequently can have their own agreement. They are defined or evolved in Joint Retrospectives.

“Try...Coordination working agreements” section on page 212

Try...Joint Sprint Reviews

A Joint Retrospective is vital to inspect and adapt the system-level ways of working. Similarly, a **Joint Review** is pivotal to focus on inspect-and-adapt for the overall product. At one of our large-group clients, the last day of the iteration runs as follows:

1. Product-level Joint Review—The overall Product Owner (PO) and supporting PO representatives are in meeting rooms around the world, all linked together with video conferencing and shared desktop technology. There are also representatives from various teams.¹⁰ What is presented? A subset of items that are of special or overall interest to the entire product group. What is discussed? Issues relevant to the overall product.
2. Single-team Sprint Review or multiteam Joint Reviews—When a supporting PO representative is served by only one team, a standard Sprint Review occurs. When the PO representative is served by several teams or the Area PO is involved, we have seen clients either (1) stagger the Sprint Reviews so that the PO representative or Area PO meets separately with each, and (2) a Joint Review with several teams together.
3. Single-team Sprint Retrospectives.

10. With the exception of Joint Retrospectives, we discourage Scrum-Masters from acting as representatives, to avoid giving the wrong impression that they are the team representative or manager.

A review bazaar—A Sprint Review involves conversation, not only a demonstration of the product; nevertheless, showing the running system is important. One technique applicable to a Joint Sprint Review is a *bazaar* [Schatz05], analogous to a *science fair*: A large room has multiple areas, each staffed by team representatives, where the features developed by a team are shown and discussed. Members of the Product Owner Team and Scrum teams visit areas of interest.

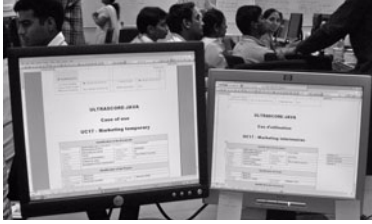
Avoid...Try...Individual team-level Sprint Review

If an individual team has its own separate Sprint Review, there is a danger—one that we have seen in action—that the team focuses on ‘their’ result instead of the overall product created by all teams together. This leads to a loss of systems focus and an increase in local sub-optimization. Avoid that. However, a Joint Review does not review all items developed during the iteration (since there are so many), and the team that developed a feature might need detailed feedback from their Product Owner. If separate reviews are held, people need to watch out for a loss of product-level focus.

Other Experiments...

Try...Spend money on improving, instead of “adding capacity”

Very large product groups become large because their default response to delivery-speed problems is to hire more people. Avoid that, and in contrast, apply the lean-thinking strategy of removing waste to improve the flow of value—reducing handoffs, WIP, and so forth. *Note that the approach is more subtractive than additive.* Often, this waste removal does not even incur additional capital investment or operating expense.



And yet, spending more money (“increasing cost”) can contribute to improving—without using it to hire more people. For example, when I (Craig here) started working at Valtech India, I noticed that people had only one small monitor. Research suggests improvements if people have more than one [Atwood08], so we bought a second monitor for everyone.

Other common—and valuable—examples include hiring expert coaches who mentor people, and classroom education with great teachers.

Try...Lower the waters in the lake



One metaphor for continual improvement—sometimes used in lean thinking—is the *lake and rocks*.¹¹

How to work toward flow of value to customers and continually improve? Do this by gradually lowering the waters in the lake. The water level symbolizes the amount of *inventory*, *WIP*, *batch size*, *handoff*, or *cycle time*.¹² That is, gradually decrease their size. As they grow smaller—as the water level lowers—new rocks hidden below the surface of the water are revealed. These represent the weaknesses and impediments in the system.

For example, perhaps a group first moves from a long two-year sequential life cycle to a four-week timeboxed iterative cycle. Some outstanding weaknesses in the system—the biggest rocks—will become painfully obvious; for instance, lack of automated tests and efficient integration. The group works on these big visible rocks; eventually they shrink in size. Then, as discussed in the “Try...Two-

11. This metaphor was also presented in *Queuing Theory* and *Lean Thinking* in the companion book.

12. These are interrelated; for example, a big batch means more WIP.

week iterations to break waterfall habits” section on page 394, the cycle time is lowered to two weeks to confront deeper problems.

Especially in large traditional groups there is a *massive* pile of rocks. The scale of improvement work can seem overwhelming. The strategy behind this metaphor makes the work tractable, while also signifying that kaizen is never finished.

Avoid...Rotating the ScrumMaster role quickly

It takes study and practice to become an effective ScrumMaster—at the very least a year. And a ScrumMaster ought to focus on organizational change—and that requires long-term constancy of purpose.

If the role is rotated quickly within a team, that necessary period of practice is missing and the organizational-improvement focus is missing or diminished. Therefore, do not rotate the position quickly.

On the other hand, a learning self-managing team should not be forever reliant on one person for this skill, and different team members should eventually have the opportunity or challenge to grow as ScrumMaster. Rotate the role—very slowly.

Try...Reduce harm of policies that cannot yet be removed

“We know that performance appraisals and performance-based incentives weaken the system, but we can’t do anything about them—they’re mandated by HR.” We hear variations of this from some people who then want to give up trying to improve the system. But Scrum encourages *the art of the possible*. With creativity, the harm from various policies can often be reduced. And possibly sometime in the future, eliminated.

For example, Bas used to work in an organization that mandated performance reviews, targets, and bonuses. When he met with people that reported to him, instead of focusing on performance in their ‘normal’ work, they set targets related to learning, such as reading books and giving presentations. During the next review, they talked about the learning and how it applied at work. One person told Bas

that nobody believed it when he told friends that he got a bonus for reading books.

Similarly, if performance-based rewards are mandated, perhaps they can be shifted to team-based goals so that there is a reduction in competition and an increase in cooperation.

CONCLUSION

Gandhi (at least as reported by his grandson Arun) once said, “We need to be the change we wish to see in the world.” This is equally applicable to the world of work—*an agile adoption needs agile adoptees*. Scrum and lean development cannot be successfully adopted with command-and-control management, predictive planning, or process recipes or “best practices” coming from ivory towers.

Even when those involved in an agile adoption have a conducive mindset, a repeating problem we have seen is a lack of Go See behavior, and therefore, a lack of insight into the real problems and useful solutions. How many product leaders or process engineers spend time regularly sitting with developers while doing the real hands-on work? Without that experience, initiatives have little useful impact; they can also focus in the wrong area—on management-level ‘improvements’ rather than at *gemba*.

Scrum, lean, agile development: these are never finished being adopted. *Agile is not a change project*. Rather, continuous improvement is a pillar of lean thinking, coupled to the idea that the people best suited to create improvement experiments are the workers.

Naturally, hands-on workers at *gemba* also have limitations. All people—including us—get stuck in inside-the-box behaviors and beliefs that inhibit challenging the status quo. So, in a lean enterprise, manager-teachers who deeply understand lean thinking, who have constancy of purpose, and who inspire kaizen mindset in others are a key positive force to promote and sustain a culture of agility.

But meaningful change and improvement cannot rely on manager-teachers; it relies on...us.

RECOMMENDED READINGS

- ❑ *The Birth of Lean*, edited by Shimokawa and Fujimoto, offers a glimpse into the evolution and adoption of lean production and thinking at Toyota. For example: “At a time when all of us are struggling to implement lean production and lean management, often with complex programs on an organization-wide basis, it is helpful to learn that the creators of lean had no grand plan and no company-wide program to install it.”
- ❑ *Fearless Change: Patterns for Introducing New Ideas* by Mary Lynn Manns and Linda Rising comes from authors with experience in change initiatives and knowledge of agile development; they emphasize a bottom-up approach to change.
- ❑ The site www.solonline.org, from the *Society for Organizational Learning*, contains many learning resources and recommended readings related to organizational improvement.
- ❑ Taiichi Ohno, in his *Workplace Management*, conveys a sense of the importance—for creating a lean culture—of leaders who truly grasp lean thinking, and relentlessly coach others in this.
- ❑ There are several good (and more bad) books on team building; some of the better ones are recommended in the *Teams* chapter of the companion book. Two mentioned in this chapter include *The Five Dysfunctions of a Team* and *Overcoming the Five Dysfunctions of a Team* by Patrick Lencioni.
- ❑ *Teamwork Is an Individual Skill: Getting Your Work Done When Sharing Responsibility* by Chris Avery emphasizes taking personal responsibility for creating an effective team, and shares tips for how to do so.
- ❑ *The Fifth Discipline: The Art & Practice of The Learning Organization* by Peter Senge, is a classic in systems thinking, learning, and the qualities needed by effective leaders for sustainable, high-impact organizational improvement.
- ❑ *Agile Retrospectives: Making Good Teams Great* by Esther Derby and Diana Larsen covers core retrospective skills. And *Project Retrospectives* by Norm Kerth explores how to do retrospectives with larger groups.

Continuous Improvement

- *Agile Coaching* by Rachel Davies and Liz Sedley captures many practical tips for ScrumMasters and other agile coaches, from two experienced coaches.

INDEX

- A**
- acceptance test-driven development
 - coach 56
 - compared to test-driven development 47
 - definition 42
 - for requirements 271
 - for UAT 463
 - in iteration 48
 - is not testing 47
 - offshore 462
 - overview 44
 - recommended reading 96
- adapters 327
- adoption
 - agile curriculum 401
 - avoid cookbooks 401
 - large-group introductions 397
 - Open Space 398
 - overview 373
 - project 380
 - targets 385
- Adzic, Gojko 49
- agile modeling 268, 292, 303
 - in design workshops 289
- ambassador
 - activities in coordination 194
 - multisite 432
 - offshore 455
- analysis
 - see requirements
- analysis group 234
- Ancona, Deborah 193
- andon 359
- appraisals
 - CMMI 480
- appraisers
 - CMMI 494
- Arbogast, Tom 499
- architect
 - active master programmers 288, 302
 - astronauts 302
 - avoid handing off to programmers 308
 - avoid separate review of work 312
 - coaches during design workshops 299
 - impact if not programming 286
 - PowerPoint 285, 302
 - program spikes 308
 - programmer in tiger team 308
 - teaches during code reviews 312
- architecture
 - analysis 301
 - and customer-centric features 307
 - build vertical slices 305
 - Community of Practice 313
 - design 301
 - documentation 310
 - see SAD workshops
 - group 234
 - integrity 293
 - outdated 302
 - question finality 301
 - see also design
 - spikes 308
 - versus growing, gardening 282
- Area Backlog 15, 133, 215, 221, 555
- Area Product Owner 15, 133, 135, 136, 215, 423, 555
- artifacts
 - see documentation
- A-TDD
 - see acceptance test-driven development
- attrition 468, 469
- B**
- backlog grooming
 - see Product Backlog refinement
- best practices 4, 492
- branching 358
- browser wars 334
- bug-free code 39
- build speed 361
- business advantages 100
- business analyst
 - not the Product Owner 124
- business manager
 - as Product Owner 121
- business rules 52

C

- C++ unit testing 73
- career paths 342
- cargo cult 2
- Carmel, Erran 413
- certifications
 - agile 493
 - CMMI 480
- change management
 - contracts 521
- change project 380
- changes
 - large ones 369
- chief engineer 128, 191
- chief Product Owner 135
- clean up your neighborhood 346
- ClearCase
 - avoid 362, 441
- CMMI
 - appraisers 494
 - overview 480
- coaches
 - avoid coaches who aren't hands-on 400
 - external 399
 - external and internal 400
 - offshore 469
- coaching
 - internal group 399
- code
 - HTLM-ize it 317
 - is the design 282
 - multisite 438
 - reviews 312
- coffee 86
- Cohn, Mike 195
- collaboration 116
- co-located team 413
- commitments 190, 335
- committer role 314
- communicate in code 211
- communication barriers 209
- Communities of Practice
 - design/architecture 313
 - for communication 208
 - general 207
 - multisite 433
- Community of Practice
 - testing 35
- competition between teams 198
- component guardians 314
- component teams
 - drawbacks 553
 - to feature teams 391
- Concordion 57
- continuous integration
 - developer practice 352
 - how frequently? 356
 - misconceptions 351
 - multisite 424
 - overview 351
- continuous integration system
 - multi-stage 364
 - overview 65, 359
 - scaling 361
 - scaling example 366
- continuous product development 157
- contract game 106
- contract negotiation 106
- contracts
 - acceptance 522
 - agile 518
 - appreciate lawyer point of view 502
 - change management 521
 - collaborate with lawyers 516
 - collaboration 116
 - common misunderstandings 504
 - contract game 106
 - deliverables 525
 - delivery 519
 - fixed price 527
 - fixed-price fixed-scope 531
 - hybrid pricing 530
 - incentives, rewards, penalties 514
 - internal 190
 - key agile insights 500
 - lawyer education 501, 509, 511, 513
 - liability 524
 - multi-phase models 539
 - multi-phase variable-model 543
 - offshore 494

- overview 499
- payment timing 526
- pay-per-use pricing 529
- progressive 536
- release contract 106
- scope 519
- silo mentality 505
- target-cost 520, 540
- termination 522
- thinking about 500
- time and materials 527
- traditional assumptions 504
- value-based pricing 528
- variable-price variable-scope 536
- warranty 525
- cookbooks 401
- coordination
 - centralized 200
 - cross-department 190
 - decentralized 206
 - meetings 200
 - responsibility for 196
 - ScrumMaster's responsibilities 197
 - team is responsible for 194
 - thinking about 189
 - travelers 207
- coordinator 190
- coordinator, ambassador, and scout 193
- copy-paste 336
- CppUTest 73
- craftsmanship 337, 339
- cross-department coordinator 190
- cross-functional teams 191
- cubicles 209
- Cucumber 57
- culture
 - multisite 437
 - overview 468
- Cunningham, Ward 57
- customer documentation 192
- customer-facing test 42
- customers 145

D

- Daily Scrum 14, 124
- defect tracking 39
- defects
 - (to fix) in Product Backlog 225
 - zero tolerance 39
- Definition of Done 15, 170, 178
- demo preparation 59
- department interfaces 190
- dependency injection 318, 319, 320
- design
 - multisite 435
 - overview 281
 - see also architecture
 - sending offshore 316
 - thinking about 282
 - walking skeleton 305
- design patterns 316
- design workshops
 - at the start 296
 - each iteration 295
 - in team room 297
 - joint for multiple teams 298
 - overview 289
- developer testing 72
- development skills 335, 339
- discuss-develop-deliver cycle 44
- dispersed team 413, 416, 419, 420, 472
- distributed teams 413, 416
- documentation
 - architectural 310
 - offshore 461, 462
 - requirements
- done
 - see Definition of Done
- dual targeting 76
- duplication
 - between requirements and tests 56
 - between tests 66
 - code 82

E

- education
 - for all teams 401

- embedded software
 - learning tests for new hardware 80
 - testing 77
- environment mapping 211
- epic
 - see splitting
 - terminology 222
- estimation
 - Monte Carlo simulation 184
 - multisite 429
 - overview 181
 - value 139
- examples for requirements 50, 245
- experiments 2
- exploratory testing 62
- external coordinator 195
- Extreme Programming
 - see XP

F

- false dichotomies 2
- Feathers, Michael 73
- feature 222
- feature screening 216
- feature teams
 - as automation team 38
 - choosing 554
 - dispersed 420
 - from component teams 391
 - in large-scale Scrum 12
 - multisite 418
 - overview 549
 - transition 555
 - vs component teams 551
 - vs project groups 552
- Fit 57
- FitNesse 57
- fixed-price contracts
 - see contracts
- flexibility and specialization 551
- Fowler, Martin 351
- FPGA 322
- function-to-function-pointer refactoring 78
- FURPS+ 231

G

- Git 358
- Grenning, James 97
- grooming
 - see Product Backlog refinement
- growing vs building 355

H

- Hackman, Richard 198
- hardware 317
- hardware abstraction layer 320, 321
- hardware design 322
- hardware simulators 71
- Hetzel, Bill 29
- Hohmann, Luke 152

I

- impediments
 - backlog 381
 - service 381
- improvement 373
- incentives 514
- incremental handoff 179
- infrastructure work 128, 168
- inspect-adapt
 - overview 373
 - product management 148
- interaction design
 - see UI design
- interaction design group 234
- interface API design 323, 324, 326
- INVEST test 247
- ISTQB 32
- iteration planning
 - see Sprint Planning

J

- jidoka 353
- JIT modeling 295
- joint design workshop 298
- joint requirement workshops 246
- joint Scrum meetings 205
- joint Sprint Retrospective 15, 17, 403

joint Sprint Review 17, 405

K

Klärck, Pekka 57

L

lake and rocks metaphor 407

language 456

large-scale Scrum

artifacts 13

definition 9

framework-1 10

framework-2 15

overview 9, 10

roles 12

law of communication paths 199

law of the inverse relationship between size and skill 339

lead Product Owner

overall 135

learning debt 336

learning tests 79

Lecht, Charles 334

legacy code

awareness 342

lethal 347

overview 333

solution 343

line manager 393

M

maintainable tests 65

Marick, Brian 27

Martin, Bob 57

matrix organization 31

MDA 291

MDD 291

meetings

multisite 428, 431, 435

Meszaros, Gerard 36

milestone 108

mocks 318, 321

modeling

agile 292

avoid extremists 303

just-in-time 295

requirements

tools 291

Monte Carlo simulation 184

moving skeletons 368

multisite

ambassador 432

avoid ClearCase 441

centrifugal forces 413

coding style 438

Communities of Practice 433

continuous integration 424

culture 437

design 435

dispersed vs. distributed 416

estimation 429

feature team 418

is non-trivial 415

matchmakers 435

meetings 425, 431, 435

one iteration per product, not site 417

Open Space 430

overview 413

partner sites 423

planning poker 429

Scrum of Scrums 430

shared space 209

site organization 417

teams 420

thinking about 414

tools 438, 439

transition to feature teams 421

video culture 425, 428

visits 432

myriad coordination methods 199

N

Netscape 334

non-functional requirements

in Product Backlog 225

see FURPS+

O

offshore

- acceptance TDD 462
 - ambassador 455
 - certification 480
 - CMMI 480
 - CMMI appraisers 494
 - coaches 469
 - contracts 494
 - culture 468
 - design problems 316
 - documentation 461, 462
 - domain and vision workshop 460
 - educate customers 446
 - educate Sales 448
 - feature teams 470
 - kickoff workshop 448
 - language 456
 - matchmakers 450
 - onshore partnership 469
 - onshore Product Owner 457
 - overview 445
 - partnership 470
 - planning 470
 - remove barriers 450
 - requirements 462, 465
 - requirements workshop 458
 - ScrumMaster 468
 - Sprint Retrospective 456
 - Sprint Review 454
 - team visits onshore 457
 - teams 466
 - titles 467
 - tools 495
 - translator on team 455
 - UAT 463, 464
 - UI design 461
 - video sessions 451
 - visits both ways 454
- ### Open Space
- for agile adoption 398
 - multisite 430
 - overview 204
- ### outsourcer
- avoid factories and factory mindset 478

- choose good programmers 479
- choosing 475
- four-year programmers 477
- improve together 480
- poor environment 477
- top-heavy management 476

outsourcing

- and legacy code 341
- choosing a partner 475
- overview 445

overall product focus 193, 198

overall Product Owner 135

overburden 337

P

PBI 215

PDMA 152

penalties 514

personal safety 376

Pichler, Roman 152

planning

- infrastructure 168

- iteration 163

- overview 155

- research and learning 166

- Sprint 163

planning poker

- multisite 429

platform departments 191

platform development 128, 168

Poppendieck, Mary 4

potentially shippable 26

potentially shippable product increment 14, 170

PowerPoint architects 302

practices

- context dependent 4

pricing

- contracts and outsourcing 527

prioritization

- avoid categories 143

- of Product Backlog 139

- of value 139, 141

Product Backlog

- Area Backlog 215

- avoid tasks 237
- avoid team-level backlogs 238
- creation 155
- items 215
- major improvement goals 404
- one per product 132
- only one per product 13
- PBI 215
- prioritization 139, 141
- refinement 166
- themes 216
- visual management 229
- Product Backlog refinement
 - for A-TDD 49
 - initial 155, 158
 - joint or asynchronous 166
 - overview 15
 - workshop 243
- product management
 - avoid short-term focus 123
 - changes when adopting Scrum 104
 - collaboration with R&D 116
 - contract negotiation 106
 - inspect-adapt 148
 - overview 99
 - traditional assumptions 117
- product manager 120, 126, 128
- Product Owner 120, 122, 135
 - Area 15, 133, 135, 136, 215
 - avoid too inward 124
 - chief 135
 - fake 123
 - has business authority 121
 - help from Team 147
 - interaction with other POs 131
 - lead 135
 - looks outward 124
 - not just an analyst 124
 - offshore development 457
 - overall 135
 - overview 12, 120
 - PO Team 17, 136, 137, 236
 - proxy 135
 - representative 135, 138
 - supporting PO 134, 135
 - us-them versus Team 125
- Product Owner representative 135, 138
- Product Owner Team 17, 136, 137, 236
- profit 141
- program manager 190
- project managers 196, 393
- projects
 - prefer product view 127, 157
- prototypes 304
- proxies 327
- proxy Product Owner 135
- punching holes 210

R

- refactoring
 - large ones 369
- Reinertsen, Donald 4
- relative value points 139
- release contract 106
- release planning 155, 158
- Release Sprint 175, 177
- release train 180
- requirement areas 133, 215
 - for non-functionals 70
 - overview 555
 - vs development areas 556
- requirements
 - acceptance TDD 271
 - artifacts 229
 - by example 245
 - clarifying by writing tests 49
 - customer-centric 236
 - meta-models 232, 233
 - multiple descriptions 56
 - non-functional 225
 - offshore 462
 - offshore to onshore 465
 - offshore workshop 458
 - overview 215
 - splitting 217, 247
 - tables 245
 - tool 462
- requirements workshop
 - A-TDD workshop agenda 54

- for Product Backlog refinement 243
- overview 240
- so-called optimizing 51
- research
 - fake 228
 - in Product Backlog 227
 - planning it 166
- rewards 385, 514
- risk 141
- Robot Framework
 - architecture 87
 - calling C code 90
 - example using 83
 - introduction 57
 - test library 86
 - types of tables 86
- room
 - see team room
- rubber chicken 354

S

- SAD workshop 310
- safety (personal) 376
- SAGE 338
- salary 342
- scenario 249
- Schwaber, Ken 9
- Scientific Management, critique 4
- scout activities 194
- Scrum
 - see large-scale Scrum
- Scrum 2.0 9
- Scrum of Scrums
 - alternatives
 - Open Space 204
 - Town Hall 205
 - format 201
 - multisite 430
 - overview 200
 - rotate representatives 203
 - rotate representatives too frequently 203
 - ScrumMaster's role 203
 - status to management 202
 - two parts 202

- ScrumMaster
 - avoid representing team 434
 - in large-scale Scrum 13
 - not project manager 393
 - offshore 468
 - slow rotation 408
- Second Life 209
- secret toolbox 336
- shared space 208
- simulation layers 321
- Slim 57
- small changes 355
- specialization 550
- spikes 308
- splitting requirements 217, 247
- splitting user stories
 - see splitting requirements
- Sprint Backlog 13
- Sprint Planning 163
 - in large-scale Scrum 14, 17
 - multisite issues 165
 - part one 163
 - part two 166
- Sprint Retrospective
 - in large-scale Scrum 15
 - joint 17, 403, 433
 - multisite 433
 - offshore 456
- Sprint Review
 - bazaar 206
 - in large-scale Scrum 15, 17
 - joint 17, 405
 - multisite 454
 - offshore 454
 - show tests 59
 - team level 406
- stakeholders 141
- stop and fix 38
- stories
 - see user stories
- story points 181
- strategic alignment 141
- stubs 318, 321
- supporting Product Owner 134, 135

T

- tables for requirements 245
- target-cost contracts 520
- targets 385
- task-coordinator activities 194
- tasks
 - in Product Backlog 237
- Taylor, Frederick 4
- TDD
 - see test-driven development
- Team (in Scrum) 12
- team room 297, 394
- team size 192
- teams
 - cross-functional 234
- technical debt 336
- technical writing 34
- test 23
- test automation team 37
- test department 30
- test education 34
- test independence 29
- test sessions 64
- test smells 36
- test tools
 - commercial 40
 - conventional 57
 - wrap conventional tools 58
- test-driven development
 - better architecture 319
 - coach 74
 - internal 75
 - overview 74
- tester certification 32
- testing
 - and Product Owner 51
 - assumptions 24, 26
 - before release 42
 - classifications 27
 - customer-facing 28, 42
 - developer 28, 72
 - in Sprint Planning 41
 - in Sprint Review 42
 - keyword-driven 83
 - legacy code 346
 - manual 60, 61
 - meaning of 24
 - on the hardware 317
 - overview 40
 - skills 96
 - specialization 33
 - terminology 26
 - thinking about 24
 - through the UI 67
 - traditional 46
 - UAT 463, 464
 - using walls 52
- testing community 35
- tests
 - automated 60
 - deleting 66
 - distill 55
 - expensive 71
 - long-running 70
 - non-automatable 62
 - non-functional 69
 - on development environment 76
 - on real hardware 77
 - performance 70
 - refactoring 81
 - reliability 70
 - table format 53
 - user-acceptance 59
 - workflow 54
- testware 37
- themes 216
- tiger team 308
- TMM 32
- tools
 - agile management 389
 - for modeling 291
 - for requirements 273
 - multisite 438, 439
 - offshore 495
 - requirements offshore 462
 - testing 40, 56, 57
- Town Hall meeting 205
- TPI 32
- traceability 67, 68, 229
- tracer code 305

- transformation project 380
- transition
 - component teams to feature teams 391
 - overview 373
 - to feature teams 421
- travelers 207
- trivializing programming 341
- TTCN 57

U

- UAT 463
 - pre-UAT 464
 - with A-TDD 463
- UI design
 - importance of 300
 - offshore 461
- UML 291, 295
- Undone Unit 31, 177
- Undone Work 173, 177, 179, 225, 226
- unit testing
 - overview 72
 - rules for 73
- unit tests
 - slow 83
- use case 249
- user stories
 - formats 271
 - history 223, 271
 - overview 266
 - question their use
 - splitting
 - term 222
- user-acceptance test 59
 - see UAT

V

- value 139, 141
- velocity 184
- video sessions 451
- video technology 425
- virtual shared space 209
- virtualization of hardware 71
- visual management 229, 367

W

- walking skeleton 305
- weakly-typed interfaces 324
- whiteboards 290
- wikis 275, 440, 462
- wishful thinking 337
- workflow test 54
- working agreements
 - cross-team 405
 - for coordination 212
- workshops
 - design 289, 295, 296, 297
 - initial Product Backlog refinement 158
 - joint 246
 - joint design 298
 - multisite 428
 - requirements 54, 240, 243
 - SAD 310

X

- XP 303
- xUnit 76