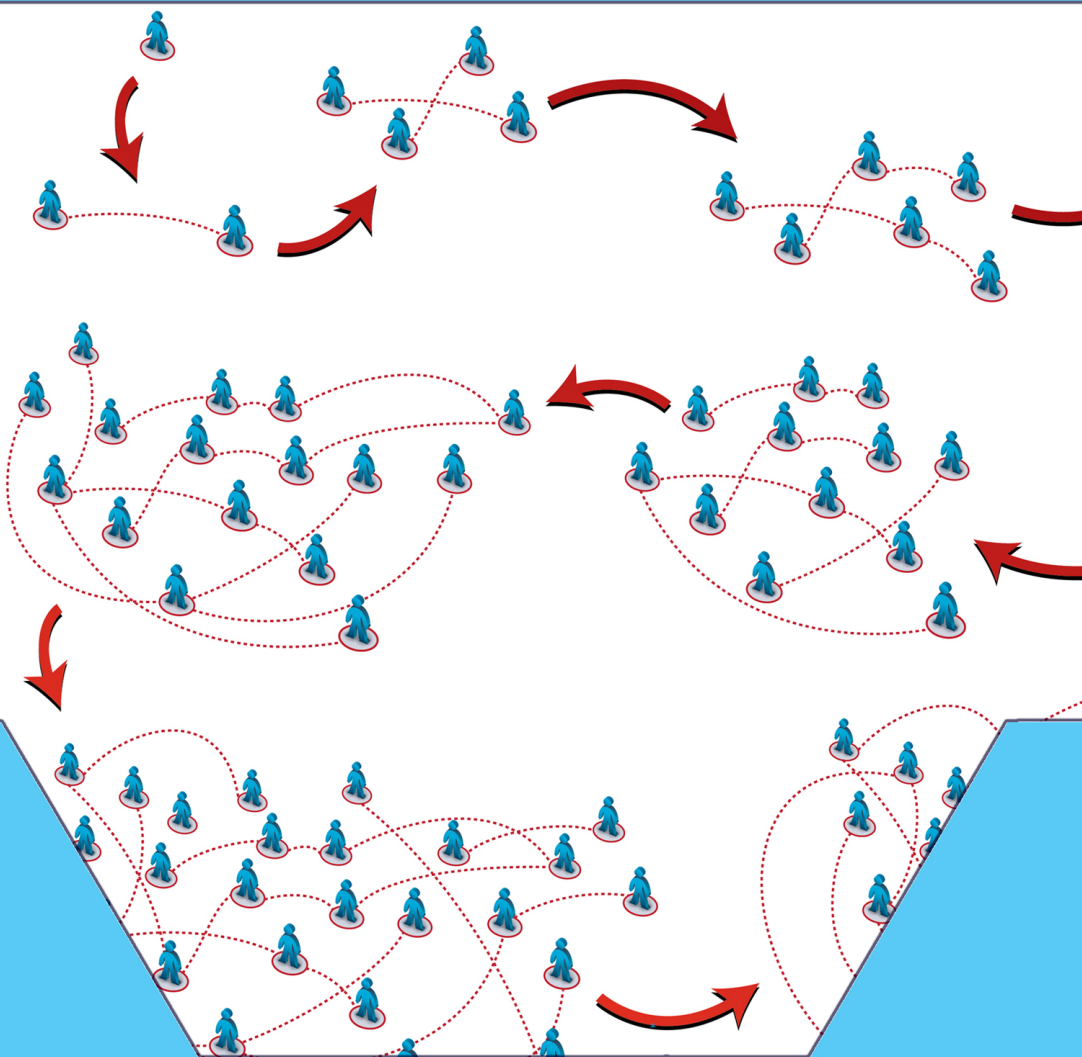


AGILE AND LEAN

Program Management

Scaling Collaboration Across the Organization



AUTHOR OF "PREDICTING THE UNPREDICTABLE"

JOHANNA ROTHMAN

Agile and Lean Program Management

Scaling Collaboration Across the Organization

Johanna Rothman

ISBN 978-1-943487-04-2



Practical ink

No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the author.

Every precaution was taken in the preparation of this book. However, the author and publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information contained in this book.

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 Practical Ink was aware of a trademark claim, the designations have been printed in initial

capital letters or in all capitals.

© 2016 Johanna Rothman

For my family. Thank you for your support.

Contents

Praise Quotes	i
Acknowledgments	iv
Foreword	v
Introduction	vii
1. Defining Agile and Lean Program Management . . .	1
1.1 Review the Twelve Principles of Agile Software Development	3
1.2 Review the Seven Lean Principles	4
1.3 Agile and Lean Together Create Adaptive Programs	4
1.4 A Program Is a Strategic Collection of Several Projects	5
1.5 Program Management Facilitates the Program to Release	6
1.6 Program Management Coordinates the Business Value	6
1.7 Agile Program Management Scales Collaboration	7
1.8 Agile and Lean Effect Change at the Program Level	9
1.9 What Program Managers Do	9
1.10 Take a Product Perspective	10
1.11 Principles of Agile and Lean Program Management	11
2. Consider Your Program Context	12

CONTENTS

2.1	Cynefin Helps with Decisions	12
2.2	Understand Your Product's Complexity	16
2.3	Know Which Program Teams You Need	18
2.4	The Core Team Provides Business Leadership and Value	23
2.5	Do You Need a Core Team?	24
2.6	Principles of Consider Your Program Context . . .	25
3.	Organize Your Program Teams	26
3.1	Create Your Core Team	26
3.2	Beware of Forgetting Core Team Members	28
3.3	The Product Owner Role Is Key to the Program's Success	29
3.4	Organize the Software Program Team	31
3.5	Don't Manage More than One Program Team Yourself	33
3.6	Principles of Organizing Your Program Teams . .	34
4.	Start Your Program Right	35
4.1	A Program Charter Sets the Strategy	35
4.2	Develop the Program Charter with the Core Team	36
4.3	We Can't Afford the Travel	37
4.4	Lead the Program Chartering Effort	38
4.5	Create Your Own Program Charter Template . . .	39
4.6	Iterate on the Program Charter and Plans	45
4.7	Create the Agile Roadmap	46
4.8	Create the Big Picture Roadmap	48
4.9	Principles of Start Your Program Right	50
5.	Use Continuous Planning	52
5.1	Differentiate Between Internal and External Releases	52
5.2	What Do You Want to Release This Month? . . .	53
5.3	Create Minimum Releasables	54
5.4	Plan for External Releases	56

CONTENTS

5.5	Deliverable and Rolling Wave Planning Helps . .	57
5.6	Small is Beautiful for Programs	58
5.7	How Often Can You Replan?	59
5.8	Separate the Product Roadmap from the Project Portfolio	61
5.9	Ways to Rank Items in the Roadmap or Backlogs .	62
5.10	Decide How You Will Evaluate Value	67
5.11	Update the Roadmaps Often	68
5.12	Principles of Continuous Planning	68
6.	Create an Environment of Delivery	70
6.1	Visualize Program Team Work	70
6.2	Keep the Program Team Work Small	72
6.3	How Features Flow Through Teams	73
6.4	How Often Can You Release Your Product?	74
6.5	Release Internally, Even with Hardware	75
6.6	Are You Integrating Chunks or Products From Others?	77
6.7	Manage the Risks of Integration from Other Vendors	78
6.8	Create a Culture of Delivery Throughout the Program	80
6.9	Principles of Create an Environment of Delivery .	80
7.	Encourage Autonomy, Collaboration, and Exploration	81
7.1	Software is Learning, Not Construction	81
7.2	Scaling Agile Means Scaling Collaborative Practices	82
7.3	Create Autonomous Feature Teams	84
7.4	Create Small-World Networks to Optimize Learning	85
7.5	Communities of Practice Create Connection and Collaboration	87
7.6	Avoid Hierarchical Titles	88
7.7	Continuous Integration and Testing Supports Collaboration	90
7.8	Beware of Technical Debt	92
7.9	Invite People to Experiment	93

CONTENTS

7.10	Principles of Encourage Autonomy, Collaboration, and Exploration	93
8.	Conduct Useful Meetings for Your Program	95
8.1	Explaining Status: Do Not Use Standups at the Program Level	96
8.2	Define a Rhythm for Your Program Team	97
8.3	Organize Your Program Team Meetings	101
8.4	Program Team Meetings Solve Problems	103
8.5	Retrospect at the Program Team Level	106
8.6	Principles for Conduct Useful Meetings for Your Program	107
9.	Estimating Program Schedule or Cost	108
9.1	Does Your Organization Want Resilience or Prediction?	109
9.2	Ask These Questions Before Estimating	110
9.3	Targets Beat Estimates	111
9.4	Generate an Estimate with a Percentage Confidence	111
9.5	Present Your Estimate as a Prediction	115
9.6	Spiral in on an Estimate	116
9.7	Supply a Three-Date Estimate	117
9.8	Do You Really Need an Estimate?	118
9.9	Beware of These Program Estimation Traps	118
9.10	Estimation Do's and Don'ts for Program Managers	120
9.11	Principles of Estimating Schedule or Cost	122
10.	Useful Measurements in an Agile and Lean Program	123
10.1	What Measurements Will Mean Something to Your Program?	124
10.2	Never Use Team-Based Measurements for a Program	124
10.3	Measure by Features, Not by Teams	126
10.4	Measure Completed Features	128
10.5	Measure the Product Backlog Burnup	129
10.6	Measure the Time to Your Releasable Deliverable	132

CONTENTS

10.7	Measure Release Frequency	132
10.8	Measure Build Time	133
10.9	Other Potential Measurements	133
10.10	Measure Performance or Reliability Release Criteria	136
10.11	How to Answer the “When Will You Be Done/How Much Will Your Program Cost” Question	137
10.12	Principles	139
11.	Develop Your Servant Leadership	140
11.1	Program Managers No Longer “Drive” the Program	140
11.2	Consider Your Servant Leadership	141
11.3	How Servant Leaders Work	142
11.4	Some People Don’t Want Servant Leadership . . .	143
11.5	Welcome Bad News	145
11.6	Use the Growth Mindset	148
11.7	Ask For the Results You Want	148
11.8	Principles of Develop Your Servant Leadership: . .	150
12.	Shepherd the Agile Architecture	151
12.1	Architects Write Code	152
12.2	Many Developers Become Architects	155
12.3	Encourage Iterative and Incremental Architecture	155
12.4	Architects Can Help Expose Risks	157
12.5	What the Program Architect Accomplishes Daily	158
12.6	Architecture is a Social Activity	160
12.7	Problems You May Encounter With Architecture .	161
12.8	Break the Architecture with Purpose	163
12.9	Principles of Shepherd the Agile Architecture . . .	164
13.	Solve Program Problems	166
13.1	Ask For the Problems or Impediments First	166
13.2	People on the Core Team Don’t Deliver What They Promise	168
13.3	Your Product Owners Have Feature-itis	168
13.4	People on Teams Are Multitasking	170

CONTENTS

13.5	How to Start a Program With More People Than You Need	171
13.6	Principles of Solve Program Problems	173
14.	Integrating Hardware Into Your Program	175
14.1	Hardware Risks Are Different Than Software Risks	175
14.2	Understand Cost and Value for Hardware	176
14.3	Understand Each Part's Value	178
14.4	See the Work	180
14.5	Design Incrementally and Iteratively	183
14.6	Use Continuous Design Review	183
14.7	Integrate Hardware Often	184
14.8	Manage Hardware Risks	185
14.9	Develop the Software Before the Hardware Is Available	186
14.10	Principles of Integrating Hardware Into Your Program	189
15.	Troubleshooting Agile Team Issues	190
15.1	The Teams Are Not Feature Teams	190
15.2	Teams Think They Are Agile, But They Are Not .	194
15.3	The Teams Have Dependencies on Other Teams .	200
15.4	Your Features Span Several Iterations	203
15.5	You Don't Have Frequent-Enough Deliverables . .	203
15.6	Teams Don't Finish When They Say They Are Done	204
15.7	Principles of Troubleshooting Agile Team Issues .	206
16.	Integrating Agile and Not-Agile Teams in Your Program	207
16.1	Waterfall Teams Are Part of Your Program	208
16.2	You Have Teams that Produce Incrementally, But Not in an Agile Way	210
16.3	You Have Teams that Prototype and Don't Complete Features	210

CONTENTS

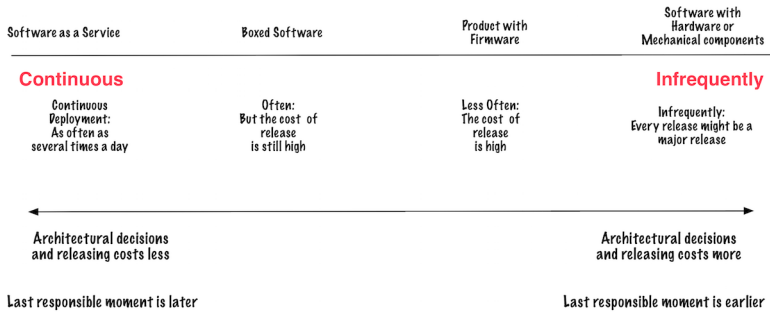
- 16.4 Principles of Integrating Agile and Not-Agile Teams
in Your Program 211
- 17. What to Do If Agile and Lean Are Not Right for You 212
 - 17.1 Try an Incremental Life Cycle 213
 - 17.2 Organize by Feature Team 216
 - 17.3 Learn to Release Interim Deliverables 217
 - 17.4 Learn How to Reduce Batch Size With a Large
Program 217
 - 17.5 Try Release Trains 218
 - 17.6 Principles for What to Do if Agile and Lean Are
Not Right for You 221
- Annotated Bibliography 223
- Glossary 228
- More from Johanna 231

12. Shepherd the Agile Architecture

One of the big problems in agile and lean program management is how to manage the product's architecture. If you don't shepherd the architecture, you end up with a mess. If you create frameworks before you have features, you will be wrong. You might have significant rework (not refactoring) late in the program. Architecture throughout the program is the way we manage that risk. You might need to [Encourage Iterative and Incremental Architecture](#), [Architects Can Help Expose Risks](#), and decide [When Should You Consider Architectural Stories](#).

The risks of deciding on the frameworks up front are considerable in a program. On the other hand, no architectural guidance might be a disaster on your program. Consider how your program can create an iterative and incremental approach to architecture. Also consider when is the most responsible moment to decide on the product's architecture and the frameworks.

Back in [How Often Can You Release Your Product?](#), you saw the potential for release frequency, based on the kind of product you have. Now, consider when to make architectural decisions.



Release Frequency and the Cost of Architectural Decisions

The closer your product is to SaaS, the longer you can wait to make many architectural decisions. You might have to make product-guidance architectural decisions, but you often don't have to make many large up-front design decisions. The closer your product is to the right side of the continuum, with hardware, the more you might have to use set-based design approaches, or provide more architectural guidance earlier.

The program architect should not decide alone. The program architect works with feature teams and other architects across the program to collaborate and decide when to select which frameworks.

That makes your program architect's job one of shepherding the business value of the architecture, which is a social and collaborative role requiring communications. The program architect helps facilitate the autonomy, collaboration, and exploration for the feature teams.

12.1 Architects Write Code

If we start with the premise that all architects on our program write code, we start well.

In software programs, we are accustomed to having enterprise, solution, or application architects. Often, those people do *not* sit

with the project teams. Instead, they proclaim the architecture from afar, early in the program.

That doesn't work in agile or lean programs. It doesn't work in other programs either, but we can discuss that later, over a beverage of your choice.

In an agile or lean program, the architect is responsible for the business value of the architecture, not for telling people what to do. The program architect does this in many ways:

- Balances the short-term goals with the overall system integrity, risk, expediency, technical debt, anything else that you would trade off short term goals against.
- Sustains development against technical debt. For test systems, this is the age-old problem of testing versus automating the tests and how you automate the tests. I'm a huge fan of automate enough and refactor your way into what you need, because you may not know what you need until you see how the system under development evolves.
- Writes acceptance criteria for system qualities and quality scenarios for the product.
- Leads the definition of how a complex system is structured, organized, and implemented. Landing zones can help guide this effort.
- Works with a feature team in a hands-on way. No seagull architects. No PowerPoint architects, (See *Practices of an Agile Developer* for an excellent description of this, [SH06](#)). No prophets. No police. Agile architects develop code and develop tests.
- Works with users (or with the program product owner on behalf of the users) to understand what the users do, how the users work, what the users understand and don't understand about the system. What is the product vision? (See [Develop the Program Vision](#) for more information.)

Architects work with the entire project team, not alone. Architects work on all parts of the product, not just the challenging or interesting parts. In fact, if there are rote parts or boring parts, maybe that's where the architect is needed most to automate something so humans don't have to do it.

In my workshops and in my executive briefings, I tell managers they should put their most talented people, aka architects, on the things that are agile or lean impediments. For complex programs, those are most often the build system and test automation. I suggest they use the architects for several iterations to make significant progress on those problems, and get to some version of done.

You may have different roles for your architect, especially if you are integrating Commercial Off the Shelf (COTS) software or vendor-supplied products:

- Act as editor-in-chief for architecture decisions on the team.
- Guide the individual feature team architects who do the actual work.
- Help establish new products that are based on the architecture. This means understanding re-use, and establishing a vision for how the architecture slowly evolves as new products come and go. Can we harvest frameworks and products from what we have now?
- Help the business people understand and take advantage of the architecture for new system features, third-party integrations, and new product lines. The architect might use the product vision to discuss the relative value of features with product owners and the value of frameworks with feature teams.

You can see that people who have architecture responsibility shepherd the *business value* of the architecture. This is not the traditional "I'll tell you how to build it because I know everything" position that way too many architects take.

12.2 Many Developers Become Architects

If architects write code, and if everyone owns the code, and we get to the final product by refactoring—which is how agile works—some substantial number of developers will work as architects at any given time. If your teams also pair or swarm over the code, no one will be able to tell who is the architect and who is not. That works quite well.

You still may need a program architect who can discuss risks with the business people, especially on the core team.



A program architect acts as a risk manager. She is experienced and able to talk with business and management with ease. She shepherds the business value of the architecture.

In [Avoid Hierarchical Titles](#), I suggested you don't call a program architect a "chief" architect. You want the architect to identify with the program, not with the organization's hierarchy. When you use words such as *chief*, *master*, or *über*, you create or reinforce a hierarchy.

Architects may need to coach other developers, especially in how to create iterative and incremental designs—if they know how.

12.3 Encourage Iterative and Incremental Architecture

Many developers and architects see the big picture of the architecture, before they write any code. They know where they want to go and they want to implement the entire feature, *now*. That's not helpful in an agile and lean program.

Instead, request that the architects collaborate on evolving the picture of the architecture over time. If they see that they have “curlicue” features, request that the architects collaborate with the teams to simplify the teams and the features.

Sometimes, this might mean that the teams realize they don’t have a cross-functional team that can deliver value. When the teams realize this, they will agitate for change. Program managers and program architects are technical leaders who can help the teams reorganize themselves, if necessary. See [The Teams Have Dependencies on Other Teams](#) for an explanation of straight and curlicue features.

I have yet to see an architecture last from initial design through the end of a program unchanged. Maybe you have.

The risks of totally designing an architecture are too high for an agile and lean program. Help the architects learn how to create features iteratively and incrementally.

Here’s one way to think about iterative and incremental architecture and design. Assume you have a product with a three-tier architecture. People have a picture of the architecture and while security pervades the product, the base security component is in the Platform layer.

As everyone creates features, it appears that the base security component is violating the Principle of Least Surprise and the Single Responsibility Principle. (The Principle of Least Surprise says that the product should act as the users expect it to. The Single Responsibility Principle says that one component should do one, and only one thing. Otherwise, you have coupling.)

This is a great time to refactor the code. Refactoring the code might not be sufficient when security violates two principles. And, you might refactor and discover performance problems. It’s time for iterating on the architecture.

You have several options. Consider the options in [Architects Can](#)

Help Expose Risks and Break the Architecture with Purpose.

No one could tell at the beginning of the program that security—as you planned it—would be a problem. The more the teams create features and refactor to patterns, the less likely the product will have a brittle architecture. With features first, everyone can contribute to the architecture.

I recommend as part of the release criteria, the feature teams define any performance or reliability criteria for the product or a piece of the product.

12.4 Architects Can Help Expose Risks

Aside from iterative and incremental development, the program architect can help expose risks. Maybe it's worth the time for an architectural spike to learn about some area of the product? Back in [Software is Learning, Not Construction](#), I said that we can learn about risks early to manage them.

Some product features are quite difficult to refactor in. These include scalability, some performance issues, and reliability to name just three. Don't proceed with just features when these quality attributes are critical to your product's success.

One way to manage these risks is to verify your roadmap has a walking skeleton (also known as the tracer bullet) approach to developing features. When you show feature teams and product owners the walking skeleton, they will ask, "How fast is this part?" or "How will this part scale from 300 to 30,000 users?" You now understand their system qualities for performance or scalability. You can adjust the system qualities as you proceed.

What if you need to know about some parts of the architecture first, because they will drive other program tradeoffs? You might. For example, in a smartphone, you might need to know the screen size

because that will drive the common GUI decisions and the heat dissipation risks.

There are several options for this kind of a potential product problem. The solution you select might depend on the kind of product you have, based on your product's complexity. See [Understand Your Product's Complexity](#). Here are some options that might work for you:

- Do a pre-program research project. Bring together enough people or teams to prototype the architecture that those people believe will support the product you need. Once you have enough information, start the program.
- Develop an architectural roadmap integrated with the program roadmap, so you create features and manage architectural risks as you proceed.
- Integrate architecture spikes with feature development. Maybe your program can still develop the operating system (the platform), and can answer other questions as you proceed.

The larger the program, the more you want to see architectural problems early. You can't do that if you can't show the product working. What would it take for your program to show a walking skeleton of working product? That is the question your product owners and architects can answer.

12.5 What the Program Architect Accomplishes Daily

Architects lead by doing. Sometimes they do the hard work to pay down technical debt that's been accumulating for years. Sometimes they do the hard work of seeing how the features are evolving into an eventual framework, or two or three. And, when you have 200

or 300 or 400 people on a program, all over the world, working in 2-week iterations, you may well need people who explore just ahead of feature teams, so that the feature teams are free to develop features.

There is a difference between agile on a small program of about three teams and agile on programs of more than 10 teams. Part of it is the communication paths. No matter how much you try to communicate, the larger program will have more communication issues, just because there are more people.

Coordinating the design and architecture among very large programs is a non-trivial task. It's partly managerial and partly technical. It's also social and communication work. See [Architecture is a Social Activity](#).

Evolving the architecture is not a problem that a program can solve with hierarchy and maintain agility. And it is a difficult problem to solve. Communities of Practice can help.

Consider these options for an architect's daily work:

1. Use an architectural kanban based on the agile roadmap. Decide what risks the architecture wants to address now and how.
2. Perform architectural spikes with a feature team. This helps a team learn how the architect thinks about problems and solutions. In addition, working with a team spreads the architecture knowledge so everyone can work better.
3. Lead (and don't direct) an Architecture Community of Practice. What do you want people to know, to evolve the architecture in a coherent way? What architecture problems do you want to raise? What do other people want to raise and address?
4. Provide direct coaching to people who want it.