# AGILE AND LEAN Program Management

Scaling Collaboration Across the Organization



## Agile and Lean Program Management

Scaling Collaboration Across the Organization

Johanna Rothman

ISBN 978-1-943487-04-2



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

Pra	ise Qu	10tes	i	
Acl	knowl	edgments	iv	
Foreword				
Int	roduct	tion	vii	
1.	Defin 1.1	ning Agile and Lean Program Management Review the Twelve Principles of Agile Software	1	
		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.	Cons	sider Your Program Context	12	

	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.	Org	anize 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.	Star	t 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	e Continuous Planning			
	5.1	Differentiate Between Internal and External Re-			
		leases	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		

	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.	Crea	te 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 Pro-	
		gram	80
	6.9	Principles of Create an Environment of Delivery .	80
7.	Enco	ourage 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 Col-	
		laboration	90
	7.8	Beware of Technical Debt	92
	7.9	Invite People to Experiment	93

	7.10	Principles of Encourage Autonomy, Collabora-	
		tion, and Exploration	93
8.	Cond	luct 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.	Estin	nating Program Schedule or Cost	108
	9.1	Does Your Organization Want Resilience or Pre-	
		diction?	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.	Usef	ul 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

	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.	Deve	lop 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.	Shepl	herd the Agile Architecture	151
12.	<b>Shepl</b> 12.1	herd the Agile Architecture	<b>151</b> 152
12.	<b>Shepl</b> 12.1 12.2	herd the Agile Architecture	<b>151</b> 152 155
12.	Shepl 12.1 12.2 12.3	herd the Agile Architecture	<b>151</b> 152 155 155
12.	Shepl 12.1 12.2 12.3 12.4	herd the Agile Architecture	<b>151</b> 152 155 155 157
12.	Shepl 12.1 12.2 12.3 12.4 12.5	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes Daily	<b>151</b> 152 155 155 157 158
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social Activity	<b>151</b> 152 155 155 157 158 160
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityProblems You May Encounter With Architecture	<ul> <li>151</li> <li>152</li> <li>155</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> </ul>
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityProblems You May Encounter With ArchitectureBreak the Architecture with Purpose	<ul> <li>151</li> <li>152</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> <li>163</li> </ul>
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityProblems You May Encounter With ArchitectureBreak the Architecture with PurposePrinciples of Shepherd the Agile Architecture	<ol> <li>151</li> <li>152</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> <li>163</li> <li>164</li> </ol>
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 Solve	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityProblems You May Encounter With ArchitectureBreak the Architecture with PurposePrinciples of Shepherd the Agile Architecture	<ul> <li>151</li> <li>152</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> <li>163</li> <li>164</li> <li>166</li> </ul>
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 Solve 13.1	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityProblems You May Encounter With ArchitectureBreak the Architecture with PurposePrinciples of Shepherd the Agile ArchitectureAsk For the Problems or Impediments First	<ul> <li>151</li> <li>152</li> <li>155</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> <li>163</li> <li>164</li> <li>166</li> <li>166</li> </ul>
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 Solve 13.1 13.2	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityArchitecture is a Social ActivityProblems You May Encounter With ArchitectureBreak the Architecture with PurposePrinciples of Shepherd the Agile ArchitectureAsk For the Problems or Impediments FirstPeople on the Core Team Don't Deliver What	<ul> <li>151</li> <li>152</li> <li>155</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> <li>163</li> <li>164</li> <li>166</li> <li>166</li> </ul>
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 Solve 13.1 13.2	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityProblems You May Encounter With ArchitectureBreak the Architecture with PurposePrinciples of Shepherd the Agile ArchitectureAsk For the Problems or Impediments FirstPeople on the Core Team Don't Deliver WhatThey Promise	<ul> <li>151</li> <li>152</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> <li>163</li> <li>164</li> <li>166</li> <li>168</li> </ul>
12.	Shepl 12.1 12.2 12.3 12.4 12.5 12.6 12.7 12.8 12.9 Solve 13.1 13.2 13.3	herd the Agile ArchitectureArchitects Write CodeMany Developers Become ArchitectsMany Developers Become ArchitectsEncourage Iterative and Incremental ArchitectureArchitects Can Help Expose RisksWhat the Program Architect Accomplishes DailyArchitecture is a Social ActivityArchitecture is a Social ActivityProblems You May Encounter With ArchitectureBreak the Architecture with PurposePrinciples of Shepherd the Agile ArchitectureAsk For the Problems or Impediments FirstPeople on the Core Team Don't Deliver WhatThey PromiseYour Product Owners Have Feature-itis	<ul> <li>151</li> <li>152</li> <li>155</li> <li>157</li> <li>158</li> <li>160</li> <li>161</li> <li>163</li> <li>164</li> <li>166</li> <li>168</li> <li>168</li> <li>168</li> </ul>

	13.5	How to Start a Program With More People Than	
		You Need	171
	13.6	Principles of Solve Program Problems	173
14.	Integ	rating 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 Pro-	
		gram	189
15.	Trou	bleshooting 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.	Integ	rating Agile and Not-Agile Teams in Your Pro-	
	gram		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 Com-	

	16.4	Principles of Integrating Agile and Not-Agile Teams	
		in Your Program	211
17.	Wha	t 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
Glo	ssary		228
More from Johanna			231

# 5. Use Continuous Planning

You've seen how to create your first roadmap and maybe even the first couple of product backlogs for any given team. Now, consider how you will update the roadmap and backlogs.

As you replan, consider how small you can make the features and minimum viable products. Your program will increase its throughput as the batch size remains small.

### 5.1 Differentiate Between Internal and External Releases

If you have continuous delivery, you can deliver something internally, to your organization, every day or multiple times a day. If you don't have continuous delivery, you might not be able to release every day.

Release something internally to your organization at least once a month. Releasing that often provides the entire program with feedback. It also provides a cadence that others will find dependable. When you release internally, you build trust across the organization. It makes sense to release as often as possible.

Internal releases help the feature teams to obtain feedback about the product. The internal releases will also show your management and sponsors the value of your work. Internal releases show people inside the organization what you have completed.

External releases show your customers what you have done. External releases are a business decision. Maybe your customer can take the updated product now, maybe not. However, the teams still need feedback on their work more often than once a quarter or whenever your customer can take a release. This is why you need internal releases at least as often as once a month.

You can release internally more often than once a month. Make the once-a-month the *minimum* time between internal releases.

# 5.2 What Do You Want to Release This Month?

Teams need small features so they can integrate and release often. Even though you want to release something every month, it will be small. What do you want to release this month?

Let's assume you have two-week iterations. Two two-week iterations fit into one month. If you work in three-week iterations, you could release at the end of each iteration. If you work in flow, maybe you want to release every time you complete a feature, instead of two weeks. Maybe you want to release when you have a minimum viable product (MVP).

I assume you release internally *at least* as often as once a month. More often is great. The more often you release internally, the more everyone—the program participants, your sponsors, anyone interested in your program—can see your progress. Everyone sees feedback.

The less often you release, the more the feature teams have to estimate. With more internal releases, the product owners can change the backlogs. It's a win-win.



Create internal releases so everyone can see program progress. The larger the program, the more you need frequent internal releases.

If you use continuous delivery, you might not need the one-quarter agile roadmap as on Example of an Agile Roadmap for One Quarter.

Your program would release features faster than a product owner could maintain the roadmap.

Consider the lack of frequent-enough delivery an impediment. See if the feature teams can solve this problem, or if it is a program issue.

### 5.3 Create Minimum Releasables

From the big roadmap, you can generate something that allows you to see what your minimum viable products, your MVPs, are for each internal release.

Maybe the product owners for a given feature set say something like this, "We don't have something minimum unless at least 80% of the features exist." They are correct when they consider an *external* release. However, your program needs minimum *internal* releases.

Maybe instead of a minimum viable product, the product owners can consider a *Minimal Indispensable Feature Set*, MIFS (BRO14).

MVPs or MIFS will vary in size. Each feature set might need something different for an MVP.

### "Our Product Grew Differently Over Time"

We had an email system as part of our product. We had an MVP of basic get-and-send emails in our first MVP. But, we didn't do forwarding or attachments until our second internal release. We didn't do group emails until our third internal release. We took other features from other feature sets, even though we were the "email" team.

I was surprised that the team didn't have such a difficult time with that. I had a harder time because I was the product owner.

I wanted to finish the email system, already! But, the team saw where the product roadmap was going, and it made sense to them. They were okay with doing different features, and they had fun with it.

They called themselves the "Email and... Team," because they did email and lots of other features. They said that knowing their MVPs made a difference for them.

-A feature team product owner

Do not try to plan specifics of the feature sets/themes for more than one quarter at a time. Even one-quarter is a ton of planning. Note that you need to consider your MVPs for release.

If you restrict your planning to the MVPs for the internal releases for a quarter: what has to be in your MVPs for each internal release each quarter and then work towards that, you will do enough planning for most projects.

If you release something every month, you never have to do big release planning. If you update the agile roadmap every iteration, or after every few features when teams work in flow, you can direct the product development without big release planning. It's all about MVPs, minimum viable product. As long as you select your MVP for the feature set, or for the entire product, and create small stories, the teams will work towards that.

# Continuous Delivery and Quarterly Planning

If you use continuous delivery, do you still need quarterly planning? You might.

If you need to commit across the organization or to customers,

use a roadmap. The roadmap will show people the small items for product direction now, and the larger items later. Everyone can see the product direction.

The fact that you do continuous delivery makes it much easier to deliver as needed and to commit to those predicted deliverables.

The roadmap is a wish-list. The deliverables are the reality.

### 5.4 Plan for External Releases

If the product owners always define MVPs, and the teams always deliver MVPs, and the MVPs move the product towards the release criteria, no one has to worry about what goes into external releases.

If you have continuous delivery, you don't have to worry about external releases. You release all the time.

You have to worry about external releases when:

- The program doesn't release all the time.
- The feature teams don't do continuous integration and release what they have into the mainline.
- Teams work on architecture as opposed to features (when the feature teams don't create features).

If you get caught in these traps, the program has problems. Either the teams have problems at the team level, or the entire program has problems. The product owners can start addressing these problems by creating MVPs and making sure the teams deliver value, not architectural stories.

### 5.5 Deliverable and Rolling Wave Planning Helps

Internal releases are deliverable-based planning. The product owners specify the deliverable chunks they want to see. As the teams finish the chunks, they can take more.

Rolling wave scheduling is this:

- Schedule your next deliverable. Make sure that deliverable is no longer than two to four weeks away.
- At the end of your first week, schedule the next deliverable.
- Repeat, after each week.

Now you always have a two-to-four week schedule with deliverables.

The teams can use iterations or flow. It doesn't matter. Each team has this responsibility: provide a constant flow of value without incurring technical debt. See Continuous Integration and Testing Supports Collaboration for more information about ways to remove technical debt.

You or the program product owner might decide that the program can take some technical debt to meet a specific deliverable. (I don't recommend this.) As part of your deliverable-based planning, add the resolution of that debt to the product roadmap or a future backlog.

Using rolling wave budgeting and incremental budgeting is especially helpful if you have people who want to know how much the project will cost. You can update the spend and plan numbers with every release.

### 5.6 Small is Beautiful for Programs

Some people think as you create an agile and lean program, it's difficult to have short iterations. They tell me that because more people and teams are on the program, you need to make the iterations longer.

The problem is this: the more you want the benefits of agile or lean, the more you need feedback. The larger the program, the more frequently you need feedback. Why? You do not want to drive the company under while it is waiting for you to complete the program. The longer it takes to get feedback on any feature or set of features, the more difficult it is for the company to know whether the program is succeeding.

The larger the program, the more the organization spends on your work. You need to deliver—at every level—often. The value of making progress every day is that everyone gets feedback. People learn early if anyone is going down the wrong path. You don't have the opportunity to bankrupt your organization because you are not delivering.

If you Review the Twelve Principles of Agile Software Development, and Review the Seven Lean Principles, you can see that the principles are about delivering working software, as fast as possible. Shorter iterations allow you to do that.

What if the people on your teams think that short iterations encompass overhead for planning and estimation and, even retrospectives? There are several reasons for that.

• When you hear the word "overhead," you are hearing someone who has not yet fully transitioned to agile. Overhead is code for "we have impediments, and we don't yet realize what they are, so we call them overhead." These impediments might be large stories, and the lack of understanding that they can spike a large story to break it into smaller chunks; or it could be a misunderstanding of what a minimum viable product could be.

- Those folks might not realize how little planning they need to do, to complete small deliverables and achieve an internal release each month.
- If your organization has not yet started to manage the project portfolio, people are multitasking among several projects or features. Under those conditions, you will have trouble building and maintaining a program of small features.
- You have a complex product, so the teams extend their iterations to more than two weeks to achieve some form of an MVP. I'll talk more about this in Shepherd the Architecture.

What if *you* think the iterations need to be longer? If you think planning is overhead, I bet you don't have small stories, or that you are trying to use estimation to manage the product roadmap or the project portfolio.

Start thinking about value. Start thinking about the smallest feature that will show everyone the progress of a feature or feature set.

### 5.7 How Often Can You Replan?

Continuous planning works in much the same way as continuous integration. When the feature teams integrate all the time, code integration is easier. When you replan all the time, the planning takes less time and is easier.

When you use continuous planning, you don't have to have big plans. You can plan for the next iteration (or two). You can plan for the next deliverable (or two). You never have to have everyone in the same room for release planning.

As the product owners see and accept the features that the teams complete in their backlogs, they can update the roadmap as a product value team. Continuous planning avoids the need for a large "let's get everyone in the same room" to plan a quarter's worth of work.

Very few teams can plan for a quarter at a time and meet that plan. Your program might have interruptions from operations/support, the rank of some features might change, and teams encounter problems every day. If you plan for a quarter, you are not likely to accomplish everything you plan.

With continuous planning, you update the backlogs just in time and keep your program open to change. The smaller your planning, the more likely the teams are to be able to achieve the vision and release criteria.



Keep planning small. With small stories, small planning, and small teams, your program is more likely to have faster throughput and faster feedback. Small and frequent planning helps your program be more resilient.

The more you can move to continuous planning, the more agile and lean your program will be. The point of the roadmaps is to show the team the big picture of the product, and how that vision changes over time. The backlogs are the specifics for each team.

The more risk you have in your program, the more feedback you need. The more you want to keep the sponsors engaged, the more often you might have to change the roadmap—and by extension—the backlogs.

If you want more feedback, release more often. Can you release every day? If not every day, what impediments do you have for creating an internal release at least once a month? As a program manager, remove those impediments. Then you can ask the program product owner to update the roadmap at least as often as once a month.

### "It's not the Plan; It's About Planning"

I used to use a more waterfall approach to my programs. I tried to plan once and have it be "the plan of record" for the entire program.

It didn't work so well. I was always replanning. Then I discovered rolling wave planning, and I learned about the value of *planning*, where we discussed what we could do when, and where the risks were, versus the actual plan, which was always out of date the next day.

Now, I use our planning as a way to understand problems and risks. I use the planning to help make decisions over the next few weeks. I never expect the plan to last past a couple of weeks. But I'm in better shape because of the risk discussions we had.

-A senior program manager

Good planning, in the sense of providing a roadmap for the teams and reflecting the current reality depends on more feedback, not less. When you plan less often, you don't see your current reality. The plans become targets, instead of plans the teams can use to guide their work.

### 5.8 Separate the Product Roadmap from the Project Portfolio

The larger your program, the more you might have projects in the form of feature sets to sequence. I like to think of this ranking as a form of feature portfolio management. The sequencing occurs when you say something like this, "We need to work on enough