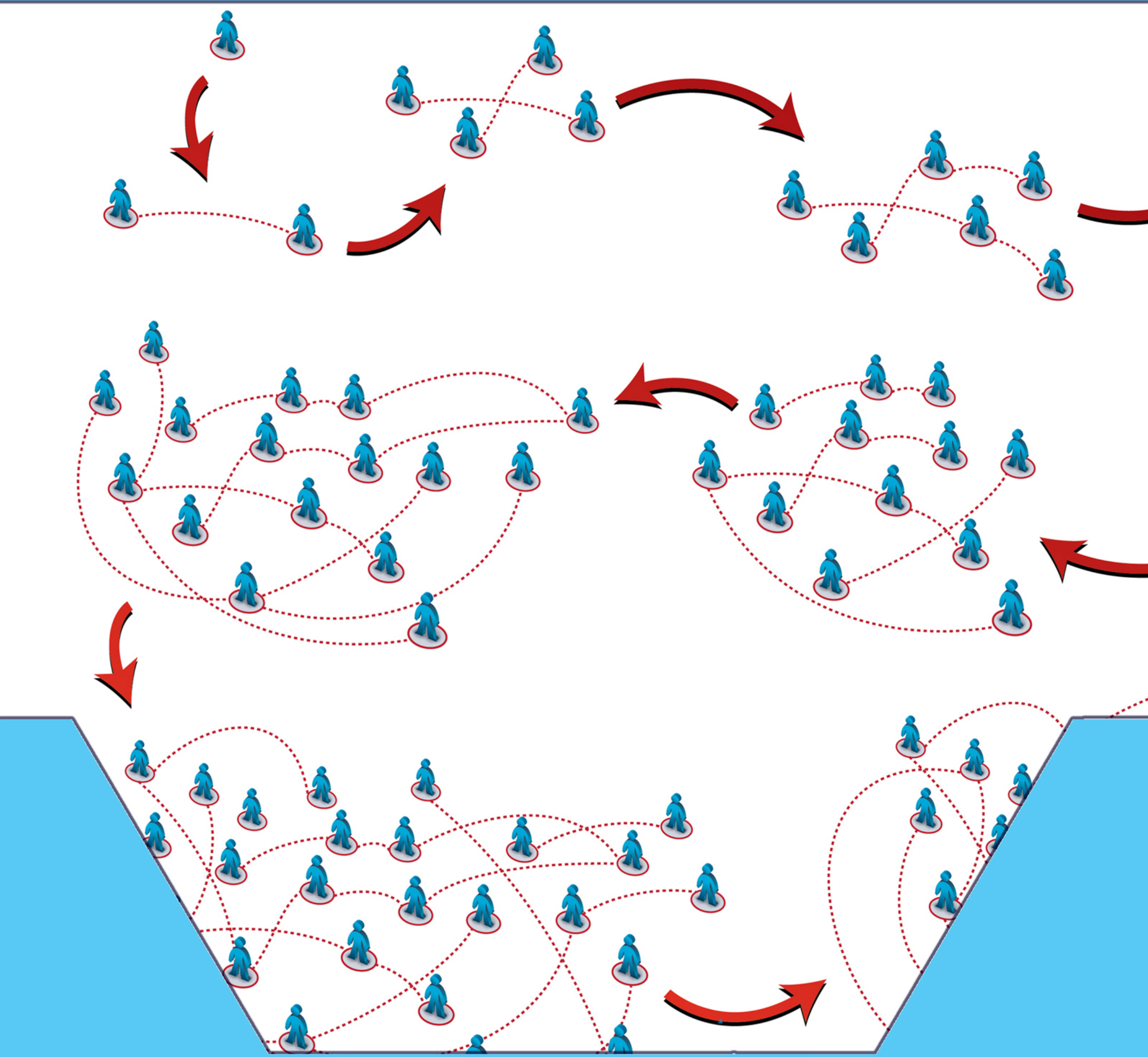
AGILE AND LEAN Program Management

Scaling Collaboration Across the Organization



AUTHOR OF "PREDICTING THE UNPREDICTABLE"

JOHANNA ROTHMAN

Agile and Lean Program Management

Scaling Collaboration Across the Organization

Johanna Rothman

This book is for sale at
http://leanpub.com/agileprogrammanagement

This version was published on 2022-04-26

ISBN 978-1-943487-04-2

Practical ink

For my family. Thank you for your support.

Contents

Acknowledgments

I thank my Managing Product Development blog readers. Your comments made my ideas better. I thank these people who read and reviewed the book and provided me feedback: Matt Barcomb, Arlo Belshee, Ian Brockbank, Clarke Ching, George Dinwiddie, Paul Ellarby, Lior Friedman, Owain Griffiths, Matt Heusser, Gary Pedretti, Catherine Swetel, Michael Vizdos, Rebecca Wirfs-Brock.

I thank my editors, Rebecca Airmet and Nancy Groth. I thank Karen Billipp for the layout and Jean Jesensky for indexing the print book.

Cover art: © Csuzda | Dreamstime.com - Team Growth Photo

Cover: Lucky Bat Books.

Foreword

I wish this book had been published the last time I ran a major project; it is a pragmatic and action based, but in a way that is also consistent with theory—something that is all too rare. The Agile community, as a whole, is riddled with rigid methods imposed with religious zeal to support training and certification programs. In contrast this book understands that scaling Agile is about the assembly of different tools, methods, and practices to achieve a result within a specific context. It has a whole section on what to do when Agile is not a cultural match for the organization. While it is predicated on servant leadership, it recognizes that this does not "mean you are a pushover." Sometimes you have to remove team members.

The early chapters do not mandate a process, and there are few of the engineering-type diagrams that overprescribe and over-structure what should be seen as a service delivery. Instead, they ask a series of questions with a range of suggested responses depending upon the answer. This is not a *one-size-fits-all* handbook, but a *many-things-might-work*, *think-before-you-act* assembly approach. Critically, it does not run away from the idea that management is necessary. One of the early phrases I highlighted was: "Some people call program management scaling agile. You could call it that. The real name is program management." Managing a program is a mixture of strategic and tactical needs, and the two need to co-exist and interact to create resilient and adaptive solutions. By combining Lean and Agile with a basic understanding of complexity (it uses my Cynefin framework), the book sets out a roadmap by which a program can be a unique assembly of appropriate methods and tools derived from multiple sources.

As the world requires shorter cycle delivery against increasingly poorly articulated needs, we need more of this deeply pragmatic thinking. Scaling is not about grand frameworks geared to making people comfortable and securing training revenues. It is about sound advice, good questions, and adaptive and flexible management. This book is a great contribution addressing that need and I am grateful for the opportunity to write this Foreword.

Professor Dave Snowden

Chief Scientific Officer

Cognitive Edge

Introduction

We hear a lot of buzz about “scaling agile.”

Instead of “scaling agile,” consider “scale projects to a program.” Program management is how we move from coordinating one project’s work to coordinating the work of several projects in a program. When your product requires you to collaborate across the organization, you need agile and lean program management.

Program management is not a new idea. What might be new for you is the application of servant leadership to the program manager role. If you want to use agile and lean approaches, you, as a program manager, serve the program. You trust people to do the right thing, and manage by exception.

You use program management anytime you want to scale collaborative teams across the organization. Here are some possibilities:

- You are a project manager, trying to corral a few teams together, to release a product.
- You are a manager who needs several teams to collaborate on one strategic objective.
- You need to have the hardware and software people work together to release a product.
- You need Marketing or Sales or Training or some other function(s) to work with the software people to release a product.

You might have a difference circumstance for your program. All programs have one thing in common—the people collaborate across the organization to deliver the product. Whatever your product is, you or your team alone can’t ensure that your product releases, no matter how agile or lean you are, when your team says, “Done!”

Programs are strategic collections of projects with one business objective. Program managers coordinate that one business objective across the organization.

When you coordinate across the organization, you recognize the need for the other teams—regardless of their function—to maintain their autonomy in how they create their deliverables. For programs, everyone comes together to serve the program's needs. Everyone optimizes for the program, not for their team.

Each program is unique. Some of you will have software-only programs. Some of you will want to use this book for products that include software, hardware, firmware, and mechanical components. That's why this book is based on principles, not mandates.

Principle-based agile and lean might also be new for you, too. Remember, that if you duplicate what works in small projects to larger programs, all you get is bloat. Bloat doesn't deliver—at least, not easily. Take the principles of agile and lean, and think, "How can I apply these principles to my context?"

Whether you are a team member on a feature team, a core team member, or the program manager, this book has something for you. Why? Because the agile and lean program is a complex adaptive system. Everyone has his or her own role to play. And, everyone in the agile and lean program has to be aware of the entire rest of the program. No one succeeds without everyone else succeeding.

This book will help you see how to use agile and lean approaches to manage your program. Here's to your success. Now, let's start.

1. Defining Agile and Lean Program Management

Imagine this scenario:

> You're the program manager for an entire product. You arrive at work and check your email. You discover that one of the feature teams found a Big Hairy problem, but they fixed it with the help of another team. Did you need to intervene? No. Neither did the software program manager. Yes, your program is large enough—18 feature teams—that you need a software program manager also.
>
> You're meeting with the core team today. Ellie, the Marketing Communications rep to the core team has been working on her deliverables for a couple of weeks. The feature teams know they have to provide performance information so MarComm can finish their glossies. MarComm knows their deliverables are key to a successful product launch.
>
> Once you explained how to set up a kanban in MarComm, they all got "kanban fever." Well, it seems that way. They love watching those stickies move across the board. The core team understands how their deliverables intersect with everyone else's deliverables now, and why it's so critical that their parts are complete and done when they commit to dates. Everyone on the core

> team is talking about "done." "We sound just like the software teams," they say.
>
> The program architect was concerned about the architecture evolution just two months ago. He'd never seen an architecture evolve. He'd always planned the architecture in advance. Then the architecture evolved anyway. You and the program product owner and the software program manager all felt as if you talked him "off the cliff." He conceded, and was willing to try to evolve the architecture.
>
> Surprisingly enough, the product is simpler than he thought—right now. He's coding, for the first time in years. He's happy. So are the feature teams. They feel as if they are part of the design thinking, not just taking orders from some guy with his head in the clouds.
>
> Senior management is happy with you, because every month you demonstrate something real, even if it's small. It's only been three months and you have a release candidate. R&D has never been able to produce something that fast. Three months into the program and you have a working product that the company can sell. Well, once MarComm finishes their deliverables.

Is this a fantasy? No. This is how agile and lean program management works. In fact, with the exception of the kanban board, that is how I worked in 1988 on a real product, in a real organization. Many successful programs repeat these principles: build trust among the teams on the program; deliver often to see feedback; build trust across the organization.

Let's review the agile and lean principles so you can consider how to apply them to your program.

1.1 Review the Twelve Principles of Agile Software Development

The list below paraphrases the twelve primary principles of agile software development. See the source for the original principles at the Agile Manifesto Principles.

1. Deliver early and often to satisfy the customer.
2. Welcome changing requirements.
3. Deliver working software frequently.
4. Business people and developers must work together.
5. Trust motivated people to do their jobs.
6. Face-to-face conversation is the most efficient and effective method of conveying information.
7. Working software is the primary measure of progress.
8. Maintain a sustainable pace.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. Reflect and adjust at regular intervals.

The point of the agile principles is that you collaborate across the organization, seeing working product as a way to work with the customer and make sure you are on track. You work in a way that enhances technical excellence so you can accommodate change. You inspect and adapt as you proceed, on the product and the process, so that you can fine-tune your team and product.

1.2 Review the Seven Lean Principles

In *Lean Software Development: An Agile Toolkit*, POP03, Mary and Tom Poppendieck summarized their lean approach with these seven principles.

1. Eliminate waste.
2. Amplify learning.
3. Decide as late as possible.
4. Deliver as fast as possible.
5. Empower the team.
6. Build integrity in.
7. See the whole.

Lean principles help you see the whole process (for a team or a program or anything in-between). You consider when to make decisions and learn as you proceed. Lean encourages you to see the entire product.

Use the agile and lean principles as you manage risk and solve problems in the program. Consider how you can apply them to your program.

The principles help you understand how to use agile and lean on your program.

1.3 Agile and Lean Together Create Adaptive Programs

When you use agile and lean principles, you can create and steer an adaptive, resilient program. When I use the word, “program,” from now on, please think “agile and lean” or “adaptive.”

1.4 A Program Is a Strategic Collection of Several Projects

A program is a collection of projects, where the value is in the overall deliverable. Yes, each project may have a deliverable that's valuable. However, the value to the organization is when all the projects get together and deliver their product. That is a concurrent program. You may also have a serial program, such as delivering a series of releases over a product's lifetime.

Think of a smartphone as an example of a strategic collection of several projects. One project might be the feature set that allows the phone to make and answer a call. Another project could be the feature set to access and leave voicemail. Another two feature sets might be the accounting for the voice data and the download data. The texting feature set would be another project. Do you see how each set of features could be its own project?

Each of these projects might require one or more feature teams working together. The teams work autonomously, however they like, as long as they are agile or lean, delivering their completed features often. Each project and team works in parallel. Each project has its own rhythm and staff and backlog. The projects deliver a working product as a program.

Beware of a collection of ranked backlogs with no strategic reason behind the order. If there isn't a larger business objective behind the backlogs, it's not a program. You might need to accomplish all that work. And, if the ranked backlogs together don't create a coherent business value, where the entire product is more valuable than each project, you don't have a program.

Can you have waterfall teams with your agile and lean teams and still have a successful program? It depends on whether the waterfall teams have interdependencies with the agile and lean teams. Make sure you read Integrating Agile and Not-Agile Teams in Your Program.

Each program needs a coherent vision behind the program so you can create a program charter. The charter helps the feature teams take responsibility for their tradeoffs. We'll talk more about this in Start Your Program Right. With a program charter in place, the feature teams won't need to work up and then down a hierarchy.

Some people call program management "scaling agile." You could call it that. The real name is program management. In program management, you scale agile and lean collaboration practices across the entire program, so you can release a great product.

1.5 Program Management Facilitates the Program to Release

Program management is the coordination and facilitation of all of the work across the organization to release the product.

The job of the program manager is to coordinate the teams so they understand enough about each other's risks so they *can* deliver. The program manager does not and cannot do this alone. The program is all about collaboration.

Projects are tactical; they get the work done. The program is strategic. It ties the projects together to bring them to delivery.

1.6 Program Management Coordinates the Business Value

I've seen—and I bet you have too—programs where the software was all done except for one small piece. The product couldn't release because that piece was vital to the release. Or the software was done but the marketing was not. Or the hardware was done, the marketing was done, and the software was stuck.

If you employ agile approaches to programs, you get to see visible progress (or lack thereof) at the end of each iteration or the end of each feature in flow or as the teams create the product. You don't have to wait until the predicted or desired end of the program to see the risk.

That's one of the ways agile reduces technical and schedule risk. The iterations or flow help you get to done across the entire program. Each iteration helps you see how things fit together. The demonstration at the end of an iteration (or at a milestone) shows you where you have technical risk, which reduces schedule risk. In general, incremental approaches reduce schedule risk and iterative approaches reduce technical risk. Because agile combines both, you reduce both kinds of risk. For more detail on life cycles, see *Manage It! Your Guide to Modern, Pragmatic Project Management*, (ROT07).

If you use lean approaches to your program, you can reduce the work in progress, which will allow you to maximize throughput. A lean approach will enable you to see bottlenecks, reduce waste, and see what is not getting done. You need both agile and lean for a program.

You don't have to release each iteration or feature to your customers. You can decide when to release externally—that's a business decision. When you see completed work each feature or each iteration is how you know you provide business value.

1.7 Agile Program Management Scales Collaboration

In non-agile program management, project managers or functional managers speak for their project teams or functional area. They commit people, manage risks, and commit other resources, such as money. Notice that there is no program-specific view of the product or transparent coordination across the functional teams. Those programs may not have a ranked product backlog.

In program management, there is no hierarchy. Everyone collaborates and coordinates across the cross-functional teams. This collaboration avoids Coordination Chaos as in TIK14.

The program teams solve problems cross-functionally. That's a huge difference.

Instead of functional managers committing on behalf of functional teams, feature teams commit to the program. The program team has the responsibility for removing obstacles so that the program delivers the business value of the program.

Lean thinking adds the holistic view to the program. When we add lean, we empower teams and eliminate waste. We amplify everyone's learning to build integrity into the product by seeing the work in progress, sharing decisions, and having a fine-grained definition of done. This is critical, because the more people we have, the more chances we have to learn and to make mistakes. If we take a lean approach at the beginning, we start with principles that make sense for building great products.

In the same way that good project management was never about command-and-control, good program management is not command-and-control. Good program management is servant leadership. Program management enables coordination: helping the teams and projects to collaborate to deliver some specific business objectives.

Once your program has more than two teams, or you need to coordinate with multiple people across the organization, releasing your product becomes much more difficult. Program management helps you coordinate across the organization, so that everyone focuses on the goal: releasing a great product that works.

1.8 Agile and Lean Effect Change at the Program Level

Agile is about the ability to change by delivering running, tested features that are valuable to the business and learning from that work. Lean is about seeing the whole, the flow of your work, building integrity into your work, and eliminating waste. If you add the technical practices (which you must in a large program), the program makes visible the values of simplicity, respect, and courage. Everyone commits to their work. You create empowered teams.

You will get increased speed as a byproduct if you have the ability to change. You will get speed if you reduce your work in progress (WIP) and waste.

No management can mandate agile and lean at the program level. Feature teams who can adapt and work together with a product focus create the agile and lean program.

As a result of transitioning to agile and lean in the teams, and using adaptive program management, you will obtain better delivery-to-market speed as a result.

1.9 What Program Managers Do

The program manager is the voice or the face of the program. The program manager represents the program to the PMO (Project Management Office) or to senior managers in the organization. As a program manager, I reported to the Operations Committee, a team of senior managers.

The program manager facilitates the collaboration across the organization. The program manager is a servant leader. Program management doesn't drive anything to completion; program managers enable the program participants to finish their work.

1.10 Take a Product Perspective

You may have noticed I have been talking about your "product." You might have applications that you refer to as "systems." You might integrate several systems from other vendors. Some of you might have something else.

I take a product-centric view of things. I suggest you do, too. If you think all the time, "Who is the customer for this?" you might have some insights about how to use agile and lean to deliver.

"I Think 'Product' Now"

I used to think about systems or applications. I've been a program manager doing in-house financial applications for years.

When I started to think about "products" instead of "applications" a funny thing happened. Other people started talking about product, too. The product owners on the program started to talk about their customers differently. They started to name their users, with specific personas. I did not expect that to happen.

Our stories got smaller. Our feature teams produced more value, because they got to done on smaller stories faster. All because I started talking about "product," not "application."

—An experienced program manager

Okay. Now you know what an agile or lean program is. Let's talk about how you might organize your program.

1.11 Principles of Agile and Lean Program Management

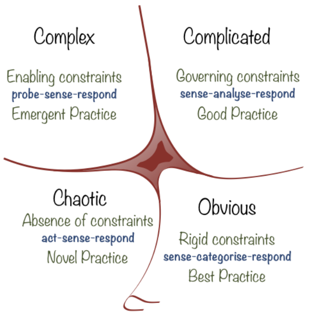
1. Take a product perspective. The principle is: “Business people and developers must work together.”
2. Agile and lean approaches encourage a holistic approach to the product where you can change more easily to meet current needs. The principle is: “Welcome changing requirements. This is a competitive advantage.”
3. Program managers are servant leaders. The principles are: “Build projects around motivated individuals,” “Trust them to get the job done,” and “Empower the team.”

2. Consider Your Program Context

You and all the members of your program will make multiple decisions on a daily basis. The Cynefin Framework is a way of thinking about your context with the intent of guiding your actions. I use Cynefin to think about how I solve problems: Can we use good practices that everyone else uses? Do we need to experiment to know how to proceed? Do we have so many unknowns that we don't know where to start?

2.1 Cynefin Helps with Decisions

The Cynefin Framework (SNB07) is a sense-making framework you can use to solve problems. Use it to guide your approach to your program.



Cynefin Framework

Based on the fact you are working in a program, you are not in the Obvious context. A program, by its very nature, is at least in the Complicated context, because of the number of communication paths.

If everyone is in a single physical location, you may be in the Complicated context. In the Complicated context, you can see straight cause-and-effect relationships among the different stresses in your program. If all your teams are experienced agile or lean teams, who know how to deliver small stories each day or so, you might be in the Complicated context. You understand what your unknowns are. You can use known and reasonable practices for organizing and working on your agile program.

As soon as you and the people in your program are not in the same location, you are no longer in the Complicated context. You have moved into either the Complex or Chaotic context. That’s because your communication will have delivery or communication lags and other interferences. Problem causes or effects may be unclear and even unknown, if only due to communication lags.

If people on your program are multitasking, or if you have people or teams who can't commit to the program, or if many of your feature teams are new to agile, you are at least in the Complex context. You may be in the Chaotic context. In either of these contexts, the unknowns create many risks and potential problems.

In my experience, if you can say, "We have done work like this, but never at this complexity or with this many teams, or never as distributed as we are now," you are in the Complex context. You have many unknown unknowns. You will have to manage the risk of those unknowns.

As you look at the Cynefin Framework, ask yourself: what context reflects your reality? How will that context help you decide whether you should sense, probe, or act as an experiment first?

If you are in the Complicated part of the framework, you need experts to solve the problems in your program. I'm not talking about experts that create bottlenecks by working alone. Instead, develop a community of experts—maybe most of the people on your program, working in their Communities of Practice—to help solve the problems.

If you are in the Complex part of the framework, consider these actions: What experiments will you use to probe, to discover your unknowns? And, what problems can you solve to move the program back to the Complicated part of the framework, where you can know your challenges?

Cynefin is not a two-by-two matrix where you locate your program, use that to make decisions, and never return to the framework. Instead, especially with programs of nine teams or more, different parts of the program will have different challenges. The more unknowable the challenges, the more that part of the program is in the Complex part of the framework. As the teams deliver features, they learn more. That part of the program moves to the Complicated part of the framework.

Sometimes, teams in the Complicated part of the framework finish features. As they learn, they uncover a huge “gotcha.” That might cause them to be in the Complex part of the framework until they run some experiments to see what they can do.

As a program manager, how can you identify issues early when you encounter Complex again? How can you help the program move from Complex to Complicated?

There are no easy answers. There is no recipe. This is work. It’s the reason why we need program management, to recognize and solve problems across the organization.

The Cynefin Framework reveals why agile program management can be difficult. As teams complete their features, the product owners need to update the roadmap and the backlogs. It’s possible the program will finish before expected. Completing—or not—other projects or programs may affect the organization’s project portfolio. Certainly, one team’s feature completion might affect the ability of other teams to deliver.

Regardless of your context, a program is emergent. With emergent projects, you can’t plan everything at the beginning. You can see a roadmap, plan a little, and continue learning and adapting as you proceed. You might want to keep the same vision of the product, but teams (with their product owners) might select different work. Or, as your customers/product owners see the product, they might want to change the product direction.

If the teams don’t complete features on a short, regular basis, no one can understand what the program status is. If the core team doesn’t solve problems that allow the program to create a *product*, you have plenty of risks, many of them unknown.

Manage by principles, not practices.

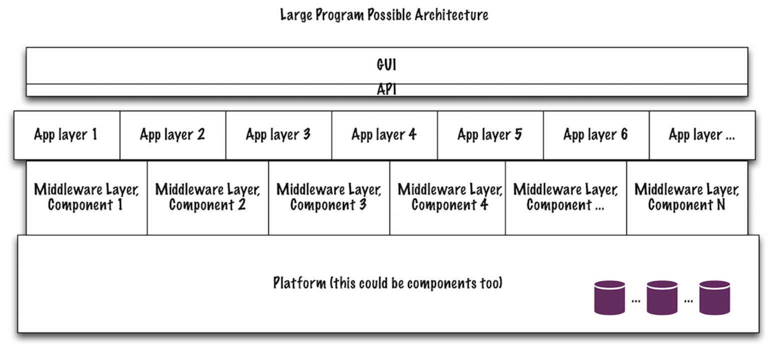
With your risks, consider principles for your program, not practices.

I could try to create a recipe for you, but that won't work. Think and recognize your context.

2.2 Understand Your Product's Complexity

Your program is unique. Your program may have complexity in a variety of areas: architecture, pressure to release, where the people sit in relationship to each other, the languages everyone uses, and each team's agility.

In my experience, the overall architecture of your product can drive much of the complexity. The more complex the architecture and the larger your program is, the more complexity you will have to manage. Here are some program architectures I have seen.

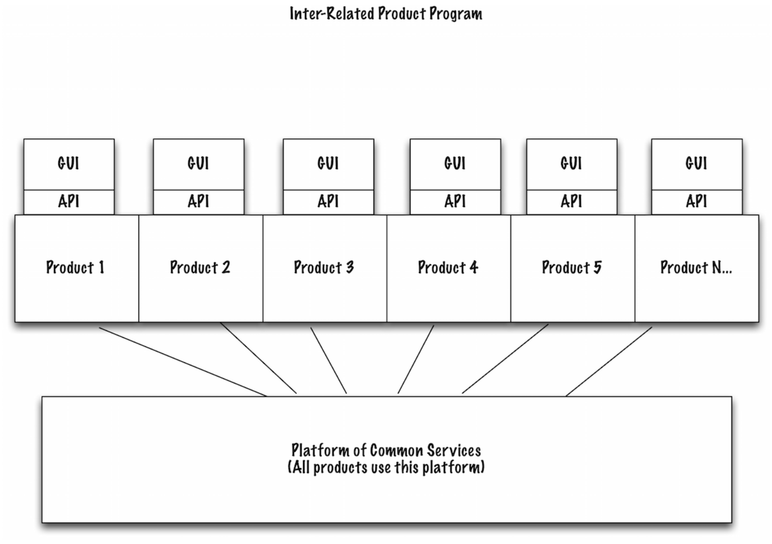

Large Program Possible Architecture
GUI
API
App layer 1
App layer 2
App layer 3
App layer 4
App layer 5
App layer 6
App layer ...
Middleware Layer, Component 1
Middleware Layer, Component 2
Middleware Layer, Component 3
Middleware Layer, Component 4
Middleware Layer, Component ...
Middleware Layer, Component N
Platform (this could be components too)
...
...

Large Program, One Coherent Product

In this case, you have one large product. It's not integrating other products or systems. Your program creates the entire product. It's big with multiple feature teams, which is why you have complexity in your program.

As an example, an operating system might look like one coherent product. Maybe a large web-based store might look like one coherent product.

Inter-related products are different. If you ever say, “Platform and layered products,” you have an example of an inter-related product.


Inter-Related Product Program
GUI
API
Product 1
Product 2
Product 3
Product 4
Product 5
Product N...
Platform of Common Services
(All products use this platform)

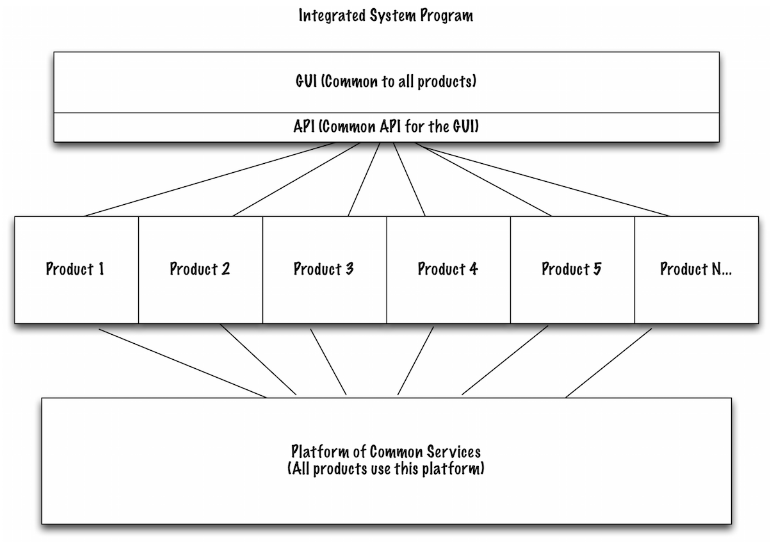
Inter-Related Product Program

In this case, you have a platform of common services with what feel to the customer as separate products. The GUIs may have their own look and feel, but the GUI is not common across your program’s product.

As an example, a smartphone is an integrated system product. Each app on the phone has its own GUI where you set the preferences and use the app. Each app uses services from the phone’s operating system.

Sometimes, inter-related products integrate other products into the one product.

It’s more likely if you integrate other vendors’ products into your own, that you have an integrated system program.


Integrated System Program
GUI (Common to all products)
API (Common API for the GUI)
Product 1
Product 2
Product 3
Product 4
Product 5
Product N...
Platform of Common Services
(All products use this platform)

Integrated System Product Program

In this case, customers buy your entire product. The product still has the platform of common services. However, you have one coherent GUI that the products have to integrate with. You might be integrating systems or hardware from vendors.

These programs tend to need programs of programs. Different products will run on their own schedule. Unless your vendors are also agile and lean, you may have to manage integration risks.

2.3 Know Which Program Teams You Need

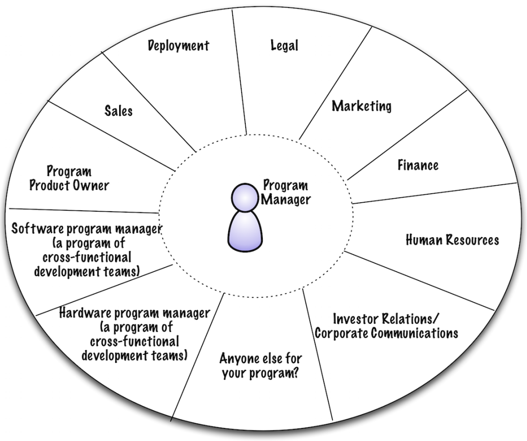
Every program needs the ability to work across the organization. You might need a *core team*, the cross-functional business team that has members from all around the organization. The core team helps coordinate the efforts that make the entire product a successful deliverable.

If you have more than two feature teams, you might also need a *software program team*. The software program team helps deliver the working software. The software program manager is a delegate to the core program team. That means that the software program manager must have a program team of his/her own. This is true for a large program.

You, as a program manager, need to understand which program teams you need. Does your program require both a core team and a software program team? Do you need a core team program manager and a software program team manager?

You can only manage one program team. One of the problems I see in too many agile programs is that they have neither a core team nor a software program team. They have many feature or component teams. They might have Scrum-of-Scrum meetings, but no real forum for solving the deep problems or managing the risks that can occur across the organization.

Each program team has a responsibility to solve problems that the teams it represents can't solve by themselves. The program team, whether it is a core team or a software program team, works across the organization, solving problems and removing obstacles for the program.


Deployment
Legal
Sales
Marketing
Finance
Program Product Owner
Program Manager
Human Resources
Software program manager (a program of cross-functional development teams)
Hardware program manager (a program of cross-functional development teams)
Investor Relations/ Corporate Communications
Anyone else for your program?

What Your Core Team Might Look Like

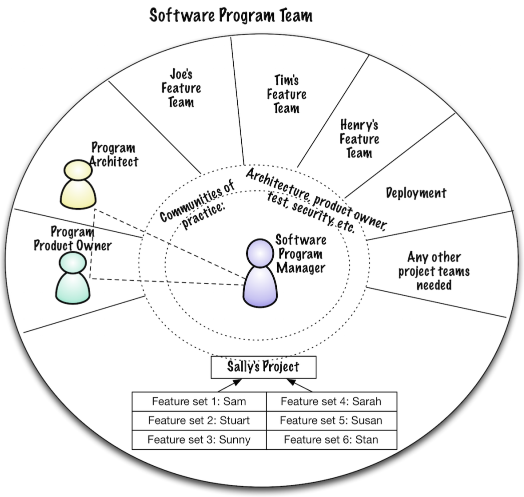
If you are coordinating and collaborating across the entire organization, you are managing or are a part of the *core* team. If you take a look at the What Your Core Team Might Look Like, you can see that there are plenty of potential participants on this program team.

Aside from the program manager, there is the software program manager, the potential hardware program manager, the program product owner, as well as the sales, deployment, legal, marketing, finance, human resources, and investor relations project managers. And those are only the people I could imagine. There might be other or different people in your organization.

Do You Have A Process Program?

Sometimes, organizations run process improvement projects, such as transitioning to agile, as if they are programs. That's fine. In this, your core team will look different. You might not have feature teams in your program.

> Know what kind of a program you have. Not all programs are the same. Use a core team that makes sense for your program.



What Your Software Program Team Might Look Like

Take a look at What Your Software Program Team Might Look Like to see a prototype composition.

Notice that the program product owner and the program architect might work as a triad with the software program manager to make risk decisions. Does this mean that the program product owner does not work with the core program manager?

It depends. It depends on who needs the program product owner. Maybe you need a product owner team, and the program product owner works with the core team and the technical product owner works with the software program owner. It depends on what your program needs.

Look at the program architect. Your feature teams need their own architects on their teams. Sally’s project—which is its own program—needs its own architect. That architect better talk to the program architect. And if there’s a hardware architect, that architect better talk to the program architect. So you might need a cross-functional community of practice architecture team, as illustrated in the “communities of practice” architecture team.

If you are a program manager, first, are you on the core team? If so, do you have everyone you need? Does that team have responsibility for deployment? (I don’t care who has responsibility for deployment, as long as someone does.)

If you are not on the core team, are you on the technical team that works across the technology? Does this team have responsibility for deployment? I’m being a little touchy about deployment because I have consulted to programs where no one was responsible for deployment and they only discovered it when I asked, “Who’s responsible for deployment?” I thought I was being stupid because I didn’t see it. No, no one had thought about it. Oops.

Why do you need all these program teams? The core team might require a different rhythm than the software program team. Since the core team often has senior managers or senior people on it, I recommend the core team use kanban to reduce the WIP (work in progress). The software program teams can use iterations if that works for them. Maybe they also use kanban; it doesn’t matter. The two program teams address different risks at different levels.

The core program team is much more strategic. Often program managers at this level manage budgets and project portfolio issues. They are the ones to say, “Wait a minute. The software program can’t succeed. We need to merge these two products.” That’s a project portfolio issue.

The software program team is more strategic than a given project, but is not as likely to manage budget or a project portfolio issue.

Program management, especially for many teams (think more than 20 teams) is about making sure you have a product that delivers the business value you want from all that effort. So the software program will have its own risks and rhythm, which is separate from the core team's risks and rhythm.

If you are a program manager, make sure you know which team you are trying to manage (coordinate and collaborate), so you can be most effective. Remember, you can only manage one program team. (See Don't Manage More Than One Program Team Yourself for more details.)

2.4 The Core Team Provides Business Leadership and Value

The core team sets the agenda and the vision for the program. The core team helps the feature teams when they need it, and stays out of their way when they don't need it. The feature teams provide status to the core team, so that the core team can share status across the organization. The core team can adapt their risk management, including the program roadmap, if necessary.

Your core team delegates are essential to your program's success. Ask yourself—and maybe the people who could be on the core team—these questions:

- Do you have the authority to commit to budget decisions for this program? Can you approve spending?
- Do you have time to commit to this program?
- Are there people or project teams that you can commit to this program?
- Can you commit other people or resources to this program?
- Can you commit to the success of this program?

When the core team members are committed to the program's success, they can solve problems more easily.

Here are the responsibilities of the core team before the program starts:

- Write the program charter.
- Create the agile roadmap.
- Create the program backlog, the backlog of features.

This is what the core team does during the program:

- Iterate on the agile roadmap.
- Iterate on the program backlog, the backlog of features.
- Solve cross-functional business problems.
- Solve problems that escalate from the software program team.
- Monitor product status and risks.
- Clear program obstacles for the teams.
- Decide when the product is ready to release.

The core team does all of this because they are responsible for the business value of the program. Because the core team is cross-functional, the people on the core team can help different departments and projects understand how they are interdependent.

The core team embodies the principle of business people and developers working together.

2.5 Do You Need a Core Team?

You might not need a core team if you have a small program. You might need a software program team with a few cross-functional people, such as the person who shepherds the product to release, maybe called Deployment or Release.

Imagine you have a web-based product. Maybe you only have four or five feature teams. Those feature teams want to know what

Marketing and Deployment are also doing—and they want to know what all the software teams are doing, too. In that case, maybe you can keep just one program team and have all the necessary people on that team.

Try to keep the number of people on your program team to ten or fewer. Otherwise, it's difficult to make decisions.

If you have the core team and the software program team as a mixed program team, be aware that you may have trouble solving problems and managing risks. The people on the mixed program team will want to solve problems at different levels, and you'll have too many people on your core team.

2.6 Principles of Consider Your Program Context

1. Consider your complexity. If your organization wants to start a brand new agile program on a highly complex architecture with geographically distributed teams when you have not succeeded with agile at the team level, your risks will be different than if the entire organization has agile experience. The principles are: "Amplify learning" and "See the whole."
2. Define which program teams your program needs. The principle is: "Business people and developers must work together."
3. Consider how you will help your program team deliver what the organization requires. The principle is: "Eliminate waste."

3. Organize Your Program Teams

Each program is unique. Once you understand your program context, you can decide how to guide your program to success.

3.1 Create Your Core Team

Your core team are your allies across the organization. They are the people who will help you move the product from an idea to a fully completed and shippable product.

You need one person from each *necessary* function across the organization, and not more than one person.

Back in What Your Core Team Might Look Like, you saw a picture of a possible core team. You might not have a hardware program manager, for example. Maybe you don't need anyone from investor relations to release your product.

No matter what, make sure that you have some form of deployment, either in your core team or in the software program team. It doesn't matter which team that person sits on, as long as you have a deployment person somewhere on a program team.

It doesn't matter what you call the deployment person either: Release Engineering, DevOps, IT, or something else. Make sure you have someone whose responsibility is to make sure your product successfully deploys.

Your core team needs people who can commit their time, and their department's time to solve problems, as well as budget. If you can identify the specific person you need, that's the best. If you only know the role, that's okay. But you also need to know the level of that role in the organization.

Make sure you ask the questions in The Core Team Provides Business Leadership. That way, you know you have the right people at the right level on the core team.

In the past, I have had problems gathering everyone on my core teams. The sales guy didn't want to commit to a biweekly meeting. He was "too busy." The MarComm woman was too frantic doing other things. The corporate lawyer never answered his email.

I needed those people for a successful product release. Without them, we couldn't know if we had the right dates, if we all understood the risks. We would be up the proverbial creek.

How could I make it worth their while to come to the core team meetings?

I have done these things:

1. Asked a specific person by name. "Mary, can you please work on my new program with me? I enjoyed working with you on that last program. I'd like to do it again." When you ask a person *by name* for help, that person is more likely to say yes. If you put out a request on email, everyone feels free to ignore it.
2. Asked a senior manager to assign "the best person" in his or her department. I've had this conversation with a senior manager more than once: "John, you know that I'm the program manager for the XYZ program, which is the company's #1 priority. I need someone from Training on the core team. I need your best person, not just to represent Training. That person will develop new training materials as Software develops the product, and as Deployment gets ready for release. We will be ready as an organization to release, all on the same day. That's why I need your very best person."

I use the Voice of Reason, along with the Steely Eyed Glare and a big smile. I almost always get what I ask for.
3. I use influence, where I think, "What will make it worthwhile for this senior manager to give me what I want?" This is why you need to know why the organization wants to start the program. I have discussed the program's deliverables with an eye to the business benefit for the senior manager.

I always tell the members of the core team that we are the best people in the organization to work on this product—that we can collaborate and shepherd this product to its final release. That's why the organization chose us.

I believe this. Why else would the organization spend money on the program?

3.2 Beware of Forgetting Core Team Members

When I coach program managers, sometimes they discover they have overlooked people or key players for the core team. Sometimes they forget because the program is small and the core team is rolled into the software program team. Sometimes it's because the program manager invites the correct people and they don't respond in a reasonable amount of time. Sometimes, there's a power play at work at a higher level in the organization. Sometimes, it's another reason.

Do you have some commonly "omitted" roles on your programs? Review your core team members. Do you have everyone you need, to release a product?

If not, ask people to participate, or enlist the senior manager in that area to help you find the right person for your core team.

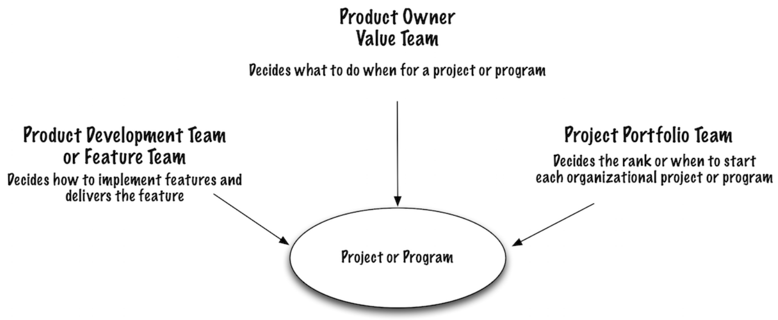
3.3 The Product Owner Role Is Key to the Program's Success

It doesn't matter if we talk about the program product owner or a product owner on a team. Each product owner has a key role to play on a program.

The program product owner, sometimes known as the PPO, or the delegate to the program team, shepherds the business value of the roadmap and of the releases. The PPO represents the product owners to the rest of the program team. The PPO is a servant leadership position. Your organization might call the PPO a product manager.

The product owner, sometimes known as the PO, is the person on the feature team who understands what the customers want and translates that understanding to stories. You may also need a subject matter expert such as an architect or senior technical person to work with the PO. The PO, or the PO team, or the PO and the program architect assess which features to do first, the technical challenges with those features, and the Cost of Delay for each feature.

If you have a highly complex product with many feature teams, consider a product owner value team.


Product Owner
Value Team
Decides what to do when for a project or program
Product Development Team or Feature Team
Decides how to implement features and delivers the feature
Project Portfolio Team
Decides the rank or when to start each organizational project or program
Project or Program

Different Teams

In this scenario, the feature team (product development team) implements a feature. The product value team decides on the product roadmap and ranks features—what to do when, for the program. The project portfolio team decides when to commit to a project or a program.

These teams work from their perspective—feature team, product, or organization—when they make decisions.

The product owners across the program work as a product value team, or a product owner team. When they work across the organization, they can limit the number of interdependencies and continue to update the roadmap based on what the teams complete.

Anyone with product owner in his/her title must understand the value of the feature under discussion, and the customer base for this product. If a product owner doesn't understand the customer base for this product, the PO will not make good decisions based on value. The PO will not be able to answer any of the questions about risk, such as, "What do our customers need or want?" for this product.

The product owner must also understand the product technology. If the PO understands the technology, the team can explain, "We can do things this way or that way." Or, "If we implement this feature first, we can save time on that feature second." Or, "We can implement the flow this way and save time for the user." See the discussion at Product Manager vs. Product Owner.

When the PO understands the customers and the technology, you have a great PO. Without both, you miss the boat.

The product owners on the teams know where the product is. They can see the product evolve every day. They have the responsibility to make small decisions *every day* that help the product grow.

Use the intelligence of the product owners as a team to create and update the agile roadmap. The smaller the features are, the easier this is.

If you have the PPO attempt to dictate the product direction without feedback, this becomes quite difficult. Your program will lose momentum. If the all the product owners join in collaboration with the PPO to create the agile roadmap and the release definition, based on what they've decided the teams should deliver, your program will maintain its momentum.

3.4 Organize the Software Program Team

Now that you have a core team, let's see what your software program team might look like. You might need to review the image What Your Software Program Might Look Like.

The software program team has feature teams alone, if they can be alone. Joe, Tim, and Henry all have stand-alone feature teams.

If more than one team works on a feature set, you might need a program for that feature set. Collect those teams into a program. Sally has a small program of collected feature teams.

For a software program, scale *out*, not up. There is no hierarchy of project managers, unless the teams desire it. There is no hierarchy of Scrum Masters. The teams are equal and work together.

The software program team has these responsibilities during the program:

- Solve cross-functional technical problems.
- Review, monitor, and manage risks in the feature teams.
- Solve problems that escalate from the feature teams.
- Monitor product status.
- Clear program obstacles for the teams.
- Provide consulting to the core team about risks, product decisions, and more.

The software program team does not have to get large. When I run programs, I email the program team meeting agenda (a problem solving meeting) to everyone on the program, and say, “Here are the people I need to attend. Everyone else: let me know if you are attending.”

Avoid Coordination Chaos

If everyone has to participate in every meeting, your program will never deliver a product.

If you have 20, 30, or 40 teams, ask the feature teams to organize as “feature set” teams. That will help several teams to become small programs inside the larger program.

Request that one person—the program manager for that feature set—participate in the software program team meetings. That feature set team can then decide how they want to make the information transparent to and from all the feature teams.

You may discover your organization has a natural limit to how big a program can get for the software program or the hardware program. Some managers would like to throw people at the program, in the hopes that they can work faster because there are more people. You may discover that it's so hard to coordinate the communications and interdependencies, it's not worth the bigness. It's worth having the program take longer while using agile roadmaps, so the program teams and the program manager can manage the coordination.

Every organization has a sweet spot for program size. If you are starting with program management, try to keep your program to nine teams or fewer. Learn how to program-manage at that size before you start with a larger program. You might not need more teams than that.

Do You Need a Hardware Program Team?

You'll notice I mentioned a hardware program team. If you have a product that requires a hardware program team, or a mechanical program team, organize those program teams. Remember, once you have more than a couple of project teams, *consider* organizing a small program for them. My rule of thumb is that once I have four project teams, I have a small program. That's my rule of thumb. Yours might be different.

I consider firmware a special case of software. I also ask that the firmware be updatable just as if it were software. That way, we have the maximum flexibility for release decisions.

3.5 Don't Manage More than One Program Team Yourself

Some program managers whose organizations are transitioning to agile are not always clear about which program team they are managing. Sometimes, that's because the organization doesn't always realize they need more than one program team.

You might think you can manage the core team and the software program team, especially if you only have three to five project teams for the software part of the program. It's tempting.

Don't do it.

You can manage one program team and have one kanban board. Make sure everyone sees all the status, risks, obstacles, everything. Or, you can have two program teams, and you will need to choose which program team you manage.

Once you have three or more teams, they tend to need a program manager of their own. That's a guideline, not a rule. In your program, the software teams might need a program manager for two teams, especially if they are geographically distributed. They have more obstacles and interdependencies.

If you try to manage more than one program team, you will not succeed at facilitating the necessary collaboration across the organization. You will not be a useful servant leader and you will let people down. Don't do it.

3.6 Principles of Organizing Your Program Teams

1. Know which program teams you need. The principles are: "See the whole" and "Simplicity."

2. Your core team, if you need one, is as large as you need it to be, and as small as you can make it. The principle is: “Amplify learning.”
3. Make sure the core team consists of everyone you need to release the product. The principle is: “Business people and developers must work together.”

4. About this sample

This sample contains the first three chapters of *Agile and Lean Program Management: Scaling Collaboration Across the Organization*. This is the entire Table of Contents for the full book:

If you enjoyed these three chapters, I hope you decide to buy the entire book. If you have questions, please email me.

If you are not sure if this book is right for you now, please join my email list, Pragmatic Manager, on my website. That’s where you can learn more about how to use practical approaches to managing your product development.

Glossary

If you are not familiar with the terms I’ve used, here are the definitions.

Adaptive: Any approach that allows you to adjust your practices or behavior to the current reality.

Agile: You work in small chunks, finishing work that is valuable to the customer in the order the customer specifies. The value of working in an agile way is that you have the ability to change quickly, because you complete work.

Backlog: Ranked list of items that need to be completed for the product.

Cost of Delay: The revenue impact you incur when you delay a project. Aside from “missing” a desired release date, you can incur Cost of Delay with multitasking, or waiting for experts, or from one team waiting for another in the program. All of these problems—and more—lead to delay of your product release.

Community of Practice: A way to share knowledge among people who belong to different teams, and share the same interests or function. For example, in a program, you might have an architecture community of practice that helps any developer learn how to evolve the design of the product. A test community of practice would provide a forum for testers to discuss what and how to test.

Flow: The team takes a limited number of items to complete, and uses the WIP limit instead of a timebox as a way to control how much work the team takes.

Generalizing Specialist: Someone who has one skill in depth, and is flexible enough to be able to work across the team to help move a feature to done.

Hardening Sprint: If a team does not complete all the work they need for a release, they may need a hardening sprint to complete all the testing for a release. This is an indication the teams are not really getting to done each iteration. They have work in progress past the end of the iteration.

Inch-pebble: Inch-pebbles are one-to-two day tasks that are either done or not done.

Iteration: A specific timebox. For agile projects, that time is normally one to four weeks. In programs, I like even smaller iterations because you want feedback more often and want to build momentum.

Kanban: Literally the Japanese word for "signboard." A scheduling system for limiting the amount of work in progress at any one time.

Lean: A pull approach to managing work that looks for waste in the system.

MVP: Minimum viable product. What is the minimum you can do, to create an acceptable product? This is not barely good enough quality. This is shippable product. However, this is minimal in terms of features.

Pairing: When two people work together on one task.

Parking Lot: This is a place to put issues you don't want to lose but don't necessarily want to address at this time.

Spike: If you cannot estimate a story, timebox some amount of work (preferably with the entire team) to learn about it. Then you will be able to know what to do after the day or two timebox.

Servant Leadership: An approach to managing and leading where the leader creates an environment in which people can do their best work. The leader doesn't control the work; the team does. The leader trusts the team to provide the desired results.

Sprint: An iteration in Scrum.

Swarming: When the team works together to move a feature to done, all together.

Technical Debt: Shortcuts a team takes to meet a deliverable. Teams might incur technical debt on purpose, as a tactical decision. Technical teams can have architectural, design, coding, and/or testing debt. Program teams might have risk or decision debt—the insufficiency of work for managing risks or making decisions.

Timebox: A specific amount of time in which the person will attempt to accomplish a specific task.

WIP or Work in Progress: Any work that is not complete. When you think in lean terms, it is waste in the system. Note that you do not get credit for partially completed work in agile.

Annotated Bibliography

[ADZ12] Adzic, Gojko. *Impact Mapping: Making a big impact with software products and projects.* Provoking Thoughts, 2012. Understand what you want to build.

[ADZ14] Adzic, Gojko and David Evans. *Fifty Quick Ideas to Improve Your User Stories.* Neuri Consulting LLP, 2014. Many teams struggle with user stories. This little book can help you improve what you plan to deliver.

[AMA11] Amabile, Teresa and Steven Kramer. *The Progress Principle: Using Small Wins to Ignite Joy, Engagement, and Creativity at Work.* Harvard Business Review Press, Boston, 2011. They have completed the research that says we like to finish work in small chunks so we can make progress.

[BEL06] Belshee, Arlo. *Promiscuous Pairing and Beginner's Mind: Embrace Inexperience.* IEEE Computer Society, 2005. We often think pairing has to be between similar experienced people. Not true.

[BRO14] Brodzinski, Pawel. *Minimal Indispensable Feature Set* at http://brodzinski.com/2014/12/minimal-indispensable-feature-set.html, 2014. What do you really need to build? How little can that be?

[BRO95] Brooks, Frederick P. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)* Addison-Wesley, Boston, 1995. Learn from a master.

[DER06] Derby, Esther and Diana Larsen. *Agile Retrospectives: Making Good Teams Great.* Pragmatic Bookshelf, Dallas, TX and Raleigh, NC, 2006. The classic work about retrospectives.

[DWE07] Dweck, Carol. *Mindset: The New Psychology of Success.* Ballantine Books, New York, 2007. This book discusses the fixed mindset and the growth mindset. If you have the fixed mindset, you believe you can only do what you were born with. If you have the growth mindset, you believe you can acquire new skills and learn. The growth mindset allows you to improve, a little at a time.

[EDM12] Edmondson, Amy C. *Teaming: How Organizations Learn, Innovate, and Compete in the Knowledge Economy.* Jossey-Bass, San Francisco, 2012. How self-organized teams really work, and what we need to make them work in different cultures.

[FOW03] Fowler, Martin. *Who Needs an Architect?*, IEEE Software July-August 2003, pp 2-4, also at http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf. Read this if you want to understand what architects can do for you, and what they should not do. Wonderful article about the value of architecture.

[GLI15] Gonçalves, Luis and Ben Linders. *Getting Value out of Agile Retrospectives: A Toolbox of Retrospective Exercises.* Leanpub, 2015. More retrospective exercises for your consideration.

[GRE02] Greenleaf, Robert K. *Servant Leadership: A Journey into the Nature of Legitimate Power and Greatness, 25th Anniversary Edition.* Paulist Press, New York, 2002. The original and definitive text on servant leadership. The forewords and afterwords provide significant value to understanding how servant leaders work.

[HAC02] Hackman, J. Richard. *Leading Teams: Setting the Stage for Great Performances.* Harvard Business Review Press, Boston, 2002. The classic work about what a team is, including what a self-managing or self-organizing team is.

[HAM14] Hammarberg, Marcus and Joakim Sundén. *Kanban in Action.* Manning, Shelter Island, NY, 2014. Terrific introduction to kanban.

[KEI08] Keith, Kent M. *The Case for Servant Leadership.* Greenleaf

Center for Servant Leadership, Westfield, IN, 2008. Useful because it's short, sweet, and specific.

[MOA13] Modig, Niklas and Pär Åhlström. *This is Lean: Resolving the Efficiency Paradox.* Rheologica Publishing, 2013. Possibly the best book about how managers should consider agile and lean. A wonderful discussion of resource efficiency vs. flow efficiency.

[PAT14] Patton, Jeff. *User Story Mapping: Discover the Whole Story, Build the Right Product.* O'Reilly, Sebastopol, CA, 2014. A terrific way to explain your stories to yourself. This book will help you move from epics and themes to stories your feature teams can build.

[POP03] Poppendieck, Mary and Tom Poppendieck. *Lean Software Development: An Agile Toolkit.* Addison-Wesley, Boston, 2003. The first book to provide a lean approach to software.

[REI09] Reinertsen, Donald G. *The Principles of Product Development Flow: Second Generation Lean Product Development.* Celeritas Publishing, Redondo Beach, CA, 2009. A classic for understanding batch size and weighted shortest job first.

[BCD05] Rothman, Johanna and Esther Derby. *Behind Closed Doors: Secrets of Great Management.* Pragmatic Bookshelf, Dallas, TX and Raleigh, NC, 2005. We describe the Rule of Three and many other management approaches and techniques in here.

[ROT07] Rothman, Johanna. *Manage It! Your Guide to Modern, Pragmatic Project Management.* Pragmatic Bookshelf, Dallas, TX and Raleigh, NC, 2007. If you want to know more about how to estimate task size, establish a project rhythm, or see a project dashboard, this is the book for you. I have references about why multitasking is crazy in here.

[ROT09] Rothman, Johanna. *Manage Your Project Portfolio: Increase Your Capacity and Finish More Projects.* Pragmatic Bookshelf, Dallas, TX and Raleigh, NC, 2009. Sometimes, program managers encounter project portfolio decisions with the feature set, or the request for people to multitask. This book helps you manage all the work in your project portfolio. I also have more references about why multitasking is crazy in here.

[RE14] Rothman, Johanna and Jutta Eckstein. *Diving for Hidden Treasures: Uncovering the Cost of Delay in Your Project Portfolio.* Practical Ink, 2014. A book about Cost of Delay and how to see how those costs affect your project portfolio.

[ROT15] Rothman, Johanna. *Predicting the Unpredictable: Pragmatic Approaches to Estimating Project Schedule or Cost.* Practical Ink, 2015. What you need to know about estimation and what to do when your estimate is wrong.

[SHI08] Shirky, Clay. *Here Comes Everybody: The Power of Organizing with Organizations.* Penguin Books, New York, 2008. Why collaboration works, even when people don't know each other. It's fascinating. Where I first learned the term "small-world networks."

[SIN09] Singer, David J., PhD., Captain Norbert Doerry, PhD., and Michael E. Buckley. *What is Set-Based Design?* at http://www.doerry.org/norbert/papers/SBDFinal.pdf, 2009. A readable paper about what set-based design is and how to use it.

[SNB07] Snowden, David J. and Mary E. Boone. *A Leader's Framework for Decision Making* in *Harvard Business Review*, November 2007. The introductory article about the Cynefin Framework.

[SH06] Subramaniam, Venkat and Andy Hunt. *Practices of an Agile Developer: Working in the Real World.* Pragmatic Bookshelf, Dallas, TX and Raleigh, NC, 2006. I first learned about the term "PowerPoint architects" from Venkat and Andy. I'd seen those kinds of architects, of course. If you want to become an agile developer, or an agile architect, start here.

[TIK14] Tikka, Ari. *Coordination Chaos* at http://www.slideshare.net/aritikka/coordination-chaos-41883070, 2014. Learn how experts can make life much more complicated.

[WIR11] Wirfs-Brock, Rebecca. *Starting with Landing Zones* at http://wirfs-brock.com/blog/2011/07/20/introducing-landing-zones/, 2011. How to trade off architectural qualities on the way to done.

[SHU14] Unknown. *Secret to Shutterstock Tech Teams* at http://bits.shutterstock.com/2014/05/08/the-secret-to-shutterstock-tech-teams/ What a real manager does with real teams.

More from Johanna

People know me as the "Pragmatic Manager." I help leaders and teams see simple and reasonable alternatives that might work in their context—often with a bit of humor. Equipped with that knowledge, they can decide how to adapt how they work.

If you liked this book, you might also like the other books I've written:

Management Books:

- *Practical Ways to Manage Yourself: Modern Management Made Easy, Book 1*
- *Practical Ways to Lead and Serve—Manage—Others: Modern Management Made Easy, Book 2*
- *Practical Ways to Lead an Innovative Organization: Modern Management Made Easy, Book 3*
- *Behind Closed Doors: Secrets of Great Management*
- *Hiring Geeks That Fit*

Product Development:

- *From Chaos to Successful Distributed Agile Teams: Collaborate to Deliver*
- *Create Your Successful Agile Project: Collaborate, Measure, Estimate, Deliver*
- *Manage Your Project Portfolio: Increase Your Capacity and Finish More Projects, 2nd ed*
- *Agile and Lean Program Management: Scaling Collaboration Across the Organization*
- *Diving for Hidden Treasures: Uncovering the Cost of Delay Your Project Portfolio*

I'd like to stay in touch with you. If you don't already subscribe, please sign up for my email newsletter, the Pragmatic Manager. Please do invite me to connect with you on LinkedIn, or follow me on Twitter, @johannarothman.

Did this book help you? If so, please consider writing a review of it. Reviews help other readers find books. Thanks!

Johanna