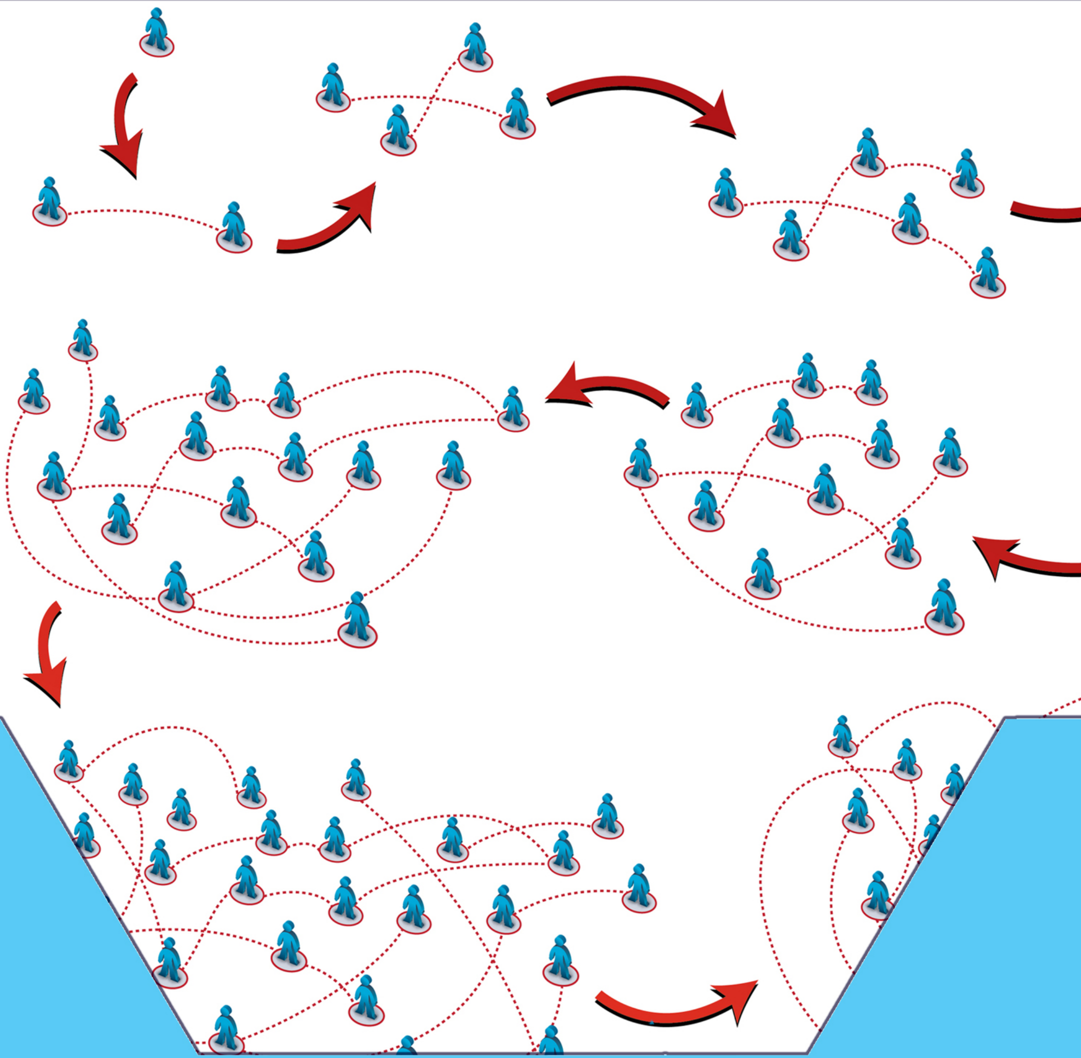
AGILE AND LEAN Program Management

Scaling Collaboration Across the Organization



AUTHOR OF "PREDICTING THE UNPREDICTABLE"

JOHANNA ROTHMAN

Agile and Lean Program Management

Scaling Collaboration Across the Organization

Johanna Rothman

ISBN 978-1-943487-04-2

Practical ink

For my family. Thank you for your support.

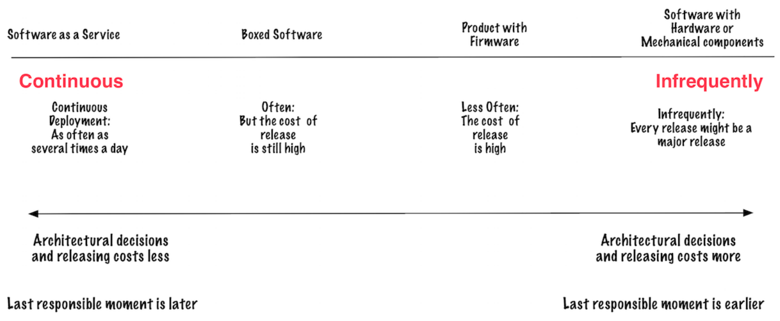
Contents

12. Shepherd the Agile Architecture

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

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

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


Software as a Service
Boxed Software
Product with Firmware
Software with Hardware or Mechanical components
Continuous
Infrequently
Continuous Deployment: As often as several times a day
Often: But the cost of release is still high
Less Often: The cost of release is high
Infrequently: Every release might be a major release
Architectural decisions and releasing costs less
Architectural decisions and releasing costs more
Last responsible moment is later
Last responsible moment is earlier

Release Frequency and the Cost of Architectural Decisions

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

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

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

12.1 Architects Write Code

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

In software programs, we are accustomed to having enterprise, solution, or application architects. Often, those people do *not* sit with the project teams. Instead, they proclaim the architecture from afar, early in the program.

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

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

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

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

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

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

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

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

12.2 Many Developers Become Architects

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

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

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

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

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

12.3 Encourage Iterative and Incremental Architecture

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

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

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

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

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

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

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

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

You have several options. Consider the options in Architects Can

Help Expose Risks and Break the Architecture with Purpose.

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

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

12.4 Architects Can Help Expose Risks

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

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

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

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

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

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

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

12.5 What the Program Architect Accomplishes Daily

Architects lead by doing. Sometimes they do the hard work to pay down technical debt that's been accumulating for years. Sometimes they do the hard work of seeing how the features are evolving into an eventual framework, or two or three. And, when you have 200 or 300 or 400 people on a program, all over the world, working in 2-week iterations, you may well need people who explore just ahead of feature teams, so that the feature teams are free to develop features.

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

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

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

Consider these options for an architect's daily work:

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