
O'REILLY®
Compliments of
VMware Tanzu™
Changing Mindsets: The Missing Ingredient to Digital Transformation
Modernizing Software Creation for Large Enterprises
Michael Coté
REPORT

VMware Tanzu™

vmware®

Free your apps. Simplify your ops.

Ease Kubernetes adoption and run modern apps at scale. Learn to bring Dev and Ops together to continuously build, run, and manage apps your users love on any cloud you choose.

LEARN MORE

Changing Mindsets: The Missing Ingredient to Digital Transformation

Modernizing Software Creation for Large Enterprises

Michael Coté

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Changing Mindsets: The Missing Ingredient to Digital Transformation
by Michael Coté

Copyright © 2021 Michael Coté. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Melissa Duffield
Development Editor: Jill Leonard
Production Editor: Kristen Brown
Copyeditor: nSight, Inc.

Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

May 2021: First Edition

Revision History for the First Edition
2021-05-12: First Release

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Changing Mindsets: The Missing Ingredient to Digital Transformation*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

This work is part of a collaboration between O'Reilly and VMware. See our statement of editorial independence.

978-1-098-10580-8

[LSI]

Table of Contents

Changing Mindsets: The Missing Ingredient to Digital Transformation

If It's Not Working, Change Your Mindset

For many years, I've studied how organizations think about and use their own, in-house software to run their business and achieve their goals. Successful, high-performing organizations rely on software for their day-to-day operations. For these organizations, software is the primary engine of business innovation: delivery in retail, telemedicine, autonomous vehicles, and new methods of banking. These organizations think about software like toothpaste. Well, like a toothpaste company would think about toothpaste: their primary product, the thing that *is* the business and deserves considered attention and innovation. Better software keeps those organizations sparkling clean and fresh.

Joking-by-analogy aside, by "software," I mean the custom-written apps and services used for internal functions and customer-facing apps as well. More broadly, when I talk about how an organization "does software," what I'm talking about is the complete, end-to-end process, set of tools, processes, and infrastructure used to govern, build, run, and manage all that software.

This all-in set of practices, governance, stages, and tools goes by many names: a value stream,[1] a build pipeline, a supply chain, or, as I prefer to call it, a path to production.

As I've discussed in my previous two reports—*Monolithic Transformation* and *The Business Bottleneck*—using software as the primary enabler of how your organization functions day-to-day is a critical part of how organizations grow and thrive, rather than stagnate and decline. Often, all of this is pulled under the word "culture." Hence, in discussions of DevOps becoming "tech companies," and otherwise changing how your software organization operates, people will say you must change your "culture." I have mixed feelings about that term, but it's an industry favorite.

Of course, not all organizations operate and think of software this way. Most large organizations have enclaves of change, but few have scaled this new way of thinking to the entire organization. For example, one survey found that 48% of executives said they hadn't made improvements to their software portfolios in a year or more. This is despite 82% of them agreeing that improving customer experience was directly tied to revenue growth.[2] These lagging organizations think about software as a once-and-done project, not an ongoing stream of work and adaption to customer needs.

"Mindset"

When it comes to improving how organizations think of software, for many years I've focused on what I call "tactics": actual practices, technologies, and organizational patterns. These tactics are the actions, policies, programs, and practices that management at large organizations put in place to improve how they do software. Also, my interest has always been on provable tactics: using case studies and before/after numbers to demonstrate that the tactics work. I relish persuasive arguments over inspirational declarations.

1 This is a term borrowed from lean manufacturing where it means something much more precise and manufacturing specific. As with most adopted language and concepts in software development, the phrase is much looser when it comes to software.

2 From "Improving Customer Experience and Revenue Starts with the App Portfolio", Forrester Consulting study commissioned by VMware (March 2020).

To go back to toothpaste, dentists always annoy me. After they look at my teeth or my kid's teeth, they scold me and tell me we need to brush more. "You should only floss the teeth you want to keep," they'll say, giving me a tongue-lashing. Well, of course, we want to keep all our teeth—is this kind of advice I should be paying people to give me? Knowing the "what" that needs to happen is rarely the problem. Figuring out "how" we can get ourselves to start brushing and flossing more is the problem.

Similarly, over the years, I've found much of the discussion about agile software development, then DevOps, and now "digital transformation" frustrating. That discussion focuses a lot on the desired end state, not how to get there from here. Hence, my focus on actionable advice that can be proven upfront.

I've realized focusing only on actionable advice isn't enough. Changing how you think of and use software as a strategic tool in your organization isn't just flossing your teeth. Recently, as I've talked with more and more executives with titles like Head of Transformation, I've realized that one of the most actionable tactics people need to take is to think differently, to change their mindset. It's as if they first need to decide to take care of their teeth, never mind giving them actual tactics, such as the specifics of how and when to floss.

This shift in my thinking happened during the height of the COVID pandemic. The way organizations needed to operate changed quickly. I saw this when I talked with Heads of Transformation types who told me amazing stories of rapid app deployments. For example, banks needed to service emergency loan programs, medical equipment distributors needed new supply chain processes, and grocery stores needed to sell in new ways, just to name a few "overnight" changes. In each case, the original needs for quality, governance, security, and customer experience existed: the apps just needed to be done quickly. The pandemic environment created a whole new set of "headwinds," business jargon for pesky threats to how an organization operates that are unexpected, outside of its control, and needing to be addressed. These headwinds often exposed how far organizations needed to go to modernize their approach to software. For example, the time it took to get even the most minor changes made to their software often took months, if not much, much longer.

Many organizations changed quickly, however. They had to get new functionality and apps out the door in record pace. For example, as documented in an excellent case study by Jana Werner and Barry O'Reilly, one UK bank put in place changes needed for contactless payment in days, instead of what would usually take months, and introduced new call center software used for working at home in less than four weeks. There are so many other "just get it done" stories that come from organizations like Albertsons that experienced, and successfully handled, a 450% increase in digital and e-commerce sales. These companies put in new practices, governance, and tools that sped up their path to production and improved the design of their applications.

These companies all knew that they should be flossing more, so to speak, but they hadn't figured out how, or even whether they had decided to start. The pandemic forced them to shift their mind, though: it made the need to change immediate and real, not just future-looking. Using a crisis to get the wheels of change going is a tried-and-true practice, but it can be exhausting to live in that mode forever. The exhaust that this way of operating blows out, like tech debt and staff burnout, will damage your long-term health as well, metaphorically and literally speaking.

What these examples made me realize, though, is that what's missing for most organizations that want to improve how they do software is changing their mindset. The people that make up these organizations need to shift their collective minds from the old way of thinking about software to a new way of thinking about the purpose of software in their organization. Do you celebrate the completion of a software project, or celebrate when the business runs better because of your software? Do you spend weekends deploying software releases with hundreds of new features, or deploy a handful of small changes each day and then spend your weekends learning to tie new balloon animals? Do you spend most of the time in your status meetings talking about delays and bugs, or discussing new customer problems that developers have discovered?

This report is my attempt to catalog some of those mindset shifts. What does it mean to *think* in terms of products versus projects? How can we motivate people to change how they work? How can managers change the way they think about their daily tasks, goals, and their staff?

Of course, dear reader, I can't resist the urge to get as practical as possible and convert these mindsets into "next steps" and, indeed, tactics...maybe even a few before/after case studies and anecdotes here and there. My goal, though, is to help motivate you, maybe even give you some therapy, to start thinking differently. Hopefully, as you read the following, you'll get a sense of what it feels like to think differently, and also how to shift you and your organization's mindset. That shift is just the first step, though: then you have to do the actual day-to-day work of constantly improving your software and, thus, improving the business. Maybe you'll even get the time to finally learn how to twist a balloon into more than just a snake.

How This Report Is Structured

This report has three sections. You've just read the first! In the introduction, we defined our space and demonstrated the need for shifting your mindset. The second section covers the goal of changing your mindset, shifting from thinking about software as a project to thinking about software as a product. Here, you'll learn how to think about the nature of software in a new way. The third section is a collection of mindset shifts, practices, and "thought technologies" to help you think and act differently.

Also, I keep saying "you." Who is, er, "you"? When it comes to change, I've found that the most important role in a large organization is management, "executives." They're the ones who own the system, the "code" of an organization. They're also the ones who have the authority and budget to change the culture. They're my first "you." I also believe that individuals need to approach culture change with a cautious, even jaundiced eye. When individuals are asked to dramatically change, they also need help changing and, more so, *trusting* management's declarations. Those are the "yous," and hopefully you're one of them. Everyone has a stake in improving how software is done at large organizations, and I hope this report will help those of you working in large organizations improve how you do software and make life a little better.

The Goal: A Culture of Innovation

I've used the word "culture" already as if it's a well-understood notion. You'd assume it was by how frequently it comes up in the discussion around digital transformation and organizational change.

Indeed, changing the "culture" is often management's entire focus. "Culture eats strategy for breakfast," Peter Drucker's well-known phrase goes. More recently, us tech-world people like to reformulate the phrase as "DevOps won't fix your broken culture."

I use a very pat definition of culture: how we do things around here. There's a lot of excellent work in defining and categorizing culture beyond this out there, but let's use this simple version in this report. What "culture" means then, are the practices, technologies, behaviors, reward, rules, and systems that an organization follows. These can be either purposefully in place, a company's stated policy and way of operating, or "how things *actually* work around here." All that makes up culture.

When we're talking about getting better at software so that it becomes a core tool for business innovation and strategy, most of what's going on here is shifting from a project-driven culture to a product-driven culture.

So, let's start by comparing these two types of culture, these two mindsets, project versus product. If we're looking to think and act differently, we need to understand what we're shifting from first.

Software as a Project

Most organizations think about software as a tool for achieving fixed goals, ideas, and daily operations.[3] Software is an imposing and incomprehensible mechanism that runs the existing state of the business machine, changing rarely and at great expense. Think of a cash register, or "point of sale" (POS), to use the term from retail. It stays pretty much the same, you can even order one from Amazon if you don't want to get too fancy. When you need a cash register, you don't pull together a team of people and create it from the ground

3 It's difficult to back up this assertion since it covers so much subjectivity and so many teams across organizations and the world. However, a Forrester survey conducted in the middle of 2020 found that "[t]wenty-seven percent [of respondents] reported having a product team structure based on long-lived, cross-functional teams….Organizations still favor functional specialization (43%) and project management (30%) over product teams." From Charles Betz et al., "The State of Modern Technology Operations, Q4 2020", Forrester report (November 2020).

up: it's a predefined set of needs, budget, and schedule; and it just gets delivered.[4]

Traditionally, software has been managed as a project. You come up with a list of features and screens you think you need, project managers create documents and project plans to give to developers who code up those specifications, and operations people run the software. A project is delivered! When the way your organization functions is fixed and rarely changes, thinking about software as a project works well enough: it's gotten us to where we are today, after all! This mentality has several benefits:

- Software as a project is psychologically comfortable. For management, we've specified exactly what needs to happen, when it will happen, and how it will happen. A decision has been made; we can stop having all these meetings and late-night arts and crafts sessions with PowerPoint. People implementing the project have been told exactly what to do and can go off and do it. There are no questions left to chance for managers or staff.
- Software as a project can be managed by numbers and status meetings. You can always know the state of the project and get the opportunity to intervene as needed, the familiar tools of traditional corporate management.
- Much work can be done in parallel. For example, because we've prespecified the feature set and use cases, while development is writing the software, the infrastructure people can start building out production. The security people can sleep well at night because they've specified how developers need to write their software so that it's secure.
- Software as a project can be outsourced, removing the costs of in-house developers from your balance sheets. Maintaining developers is expensive, and once you've delivered the core software, you may not need to keep that many around. Over the past 30 years, many CIO careers have been made on these strokes of managerial brilliance.

4 For those of you out there saying "yeah, but": sure, if you're trying to launch a revolutionary, self-service checkout store experience, perhaps you'll develop the POS on your own. You've found the outlier! (Even then, I suspect you won't be creating the scanner that goes beep-boop yourself.)

Of course, dear reader, you can imagine that I'm now going to say that these are all just illusions, comforting dreams. The Standish Group has tracked software project success over the past few decades. By my analysis of the last 13 years of their surveys, around 60% to 70% of software projects fail by the criteria of feature set, budget, and schedule.[5] So, those project dreams work out about 30% of the time—or, perhaps, people just get lucky. It's also hard to measure if all those projects were a success based on "usefulness." For example, a 2009 Microsoft study found that only about a third of its applications' features achieved the team's original goals—that is, were useful and considered successful.

When you want to change the way your business functions more frequently—whether by choice or because competition and wild swings in the world's state (those market headwinds!) force you to change—a project mindset isn't good enough: you're always delivering what was needed yesterday and, at best, what you *thought* you needed today. A project mindset doesn't adapt to new market dynamics, to things you've learned, and then implement change.

Thinking about software as a "product" is what's needed in this kind of environment. You need to rely on software as the heart and brain of your strategy, innovation, and even daily operations engines. Let's take a look at what it means to think of software as a product.

Software as a Product

A *product* mindset about software is much less focused on delivering the planned feature set, budget, and schedule. A product mindset focuses on continually learning what the product is and continually delivering new features as determined by evolving customer and business needs. This is much like most businesses nowadays that are forced to evolve frequently rather than continue to sell buggy whips. This may seem not that much different than a project mindset, but it's incredibly different when it comes to how you think of and manage your software.

5 This is my rough estimate based on Standish Group numbers that I could find publicly available online. For a similar discussion of these numbers, see *The Cost of Poor Software Quality in the US: A 2020 Report*, which has even more software project failure numbers as well.

Product-oriented teams focus less time on perfecting the requirement-gathering and specification phases, and they handle status reporting differently. Instead of focusing on project management, a product team uses a rigorous tool of customer-driven experimentation and learning what I call "small batch" thinking.[6] This feedback cycle requires close knowledge and study of the business that the software is implementing and intimacy with the people using the software (usually, either "customers" or employees using internal facing software).

The goal of the small batch cycle is to verify the problem you're facing by creating a hypothesis, testing that hypothesis, and then observing the results. Did we pick the right problem to solve, did we create the right features in our software to solve it? Can we come up with an even better way to solve the original problem? This overall recalibration on learning what your software should do based on what the people or organizations using your software actually need or want is key to the product mindset. Often, people don't even know what they need. They have to go discover it. As Allianz's Dr. Poelchau comments on this:

> Whenever I go to where the [employees] sit, it's always interesting to see the kind of Post-its they put on their screen, because this is how the real process is run. But if you think about a waterfall approach and you ask people to write down these requirements, they would usually never have the idea to write down these kinds of requirements. That they, for example, need a screen where they can put their Post-its.

Time and again, it's finding those sticky notes that lead to big wins, for customers and the business. As another example, when The Home Depot wanted to improve the performance of their custom paint desk, they found that staff at those desks disliked the software so much that they'd started using sticky notes instead. The product team only discovered this when they went and observed the paint desk in operation. We'll see a similar story in "Case study: Measuring mayonnaise" on page 10.

6 I've covered the small batch cycle much more in-depth in *Monolithic Transformation* with a few examples. I make no claims at all of having invented this concept, quite the opposite: it's just a simplification of lean product management as practiced by Tanzu Labs (formerly named Pivotal Labs) and many other agile practitioners. Also, see Matt Parker, *Radically Collaborative Patterns for Software Makers* (O'Reilly Media, 2020) for more discussion.

A project mentality will have difficulty uncovering all of these "implicit" features because the project mentality is prescriptive. For example: "the software will allow you to search for three years of billing history, put a calendar search in in the UI and then you're done." A product mindset is more exploratory and focused on learning what people need. For example: "We hypothesize that customers will most often only want to see the most recent three bills. Therefore, we should prominently list links to the past three months bills and de-emphasize the general, calendar search. Let's use A/B testing to deploy that new UI to production to a subset of users to validate or invalidate our theory."

And it's not only the initial discovery of those implicit, sticky notes—the learning never ends. When you're doing things in a product way, you're not delivering static value, you're delivering ongoing usefulness, ongoing improvement, and ongoing understanding. This ability to explore how customers are solving their problems with your software is a positive, "leaky abstraction" back to the business. Knowledge gained in the small batch cycle can be fed back to change and improve business strategy: you're finding new things the customers will buy, new reasons they will continue being customers—and discovering new competitive advantages.

Shifting from software as a project to a product is what drives so much of the operational change and so many of the mindset shifts. Let's look at an example of an organization that used a product mindset.

Case study: Measuring mayonnaise

While you may not know about them, you likely benefit weekly from food services companies. They deliver food to restaurants, run campus and corporate cafeterias, and otherwise help with whatever food and feeding needs you have. At one of these companies, management wanted to lower costs and introduce more consistency in meal preparation by putting recipe books on tablets. This would lead to lower costs and better customer experience: following recipes closer would match both ingredient portions and the delicious recipes that headquarters was dreaming up. To do this, management asked one of their software teams to digitize the recipes so they could be viewed on tablets.

Thus far, this is a project mindset to software: the product team is given a specific set of requirements to implement. Instead, this team had a product mindset. The team's approach was to first observe the actual end users and research the problems they had. For a week, the product team of developers, product managers, and designers showed up early in the dark of morning to watch the kitchen staff prepare and cook food. These product team members were curious and people-centric—and had the autonomy to verify the problem to solve.

The team observed that, for sure, recipe handling in three-ring binders was not ideal. But the kitchen staff spent an incredible amount of time on a different process: measuring the temperature of mayonnaise, chicken, and other food that needed to stay at a specific temperature. The process of measuring and recording the temperature in binders interrupted staff frequently and simply took up a lot of time.

Using a product mindset, this team had spotted a more important need than management originally identified. The team worked on digitizing the temperature-measuring process, saving staff a lot of time, and also making it easier to pass health inspections. This delivered on the original business goal of lowering costs and increasing customer experience: failing health inspections can damage ongoing business, preventing spoiled food that had to be thrown out, and, not to mention, getting food poisoning probably is way up there in poor customer experience.

This example shows that the initial ideas about what your software should do are often not exactly right. Teams operating under a project mindset would likely have just delivered what was asked, not caring to discover what was actually happening in the kitchen. Subject matter experts (SMEs) and business analysts who write up those initial project specifications are relying on past experience. The experience can be valuable, but they also need to continually update their assumptions and discover new customer needs and opportunities. The actual product team is often the best positioned to do this, suggesting that those business analysts should work more closely

with the team, if not be part of the team in the role of a product manager.[7]

The teams went on to digitize menus and do other apps. But this approach to adapting your plans based on the measurable outcomes of your small-batch process is key to understanding the difference between the project and product mindset.[8]

The New Meatware

As we saw in "Case study: Measuring mayonnaise" on page 10, teams that work on software as a product are less focused on implementing the exact feature set, expressed in a set of assumed requirements they're given and more on continually discovering how their software can deliver business value. They follow the hypothesis-code-release-observe-verify small batch process. This is "innovation," creativity.

As such, people on these teams tend to have different mindsets than project-driven teams. Participants tend to be innovative, risk-takers, and people-centric.[9] To support these types of teams, management tends to have a different mindset to software as well. In IT, we're always thinking about technology improvements: faster and cheaper hardware, and software that makes us more productive and runs the business. Technology is always important, but what we need to focus on more right now is the third technology: meatware. That's the tongue-in-cheek phrase I like to use for the way an organization operates, thinks, the practices they follow, and, I suppose, "culture" as defined earlier.[10]

Let's look at some of the new mindsets and meatware needed to shift to a product-oriented way of doing software.

---

7 Thanks to Jon Osborn who raised this point about business analysts primarily drawing on past experience.

8 This story is based on an interview with Jonathan Sirlin in *Pivotal Conversations*, Episode 113, "Product Management, with Jonathan Sirlin" (September 2018).

9 These attributes are discussed in detail in *Monolithic Transformation*.

10 Historically phrased "peopleware" by Peter Neuman and later a book of the same title by Tom DeMarco and Timothy Lister.

Failure == Learning

The most important mindset shift you and your staff must make first is how you think about failure. How your organization's culture thinks about failure will determine your success in transforming to the product mindset. Not all failure is good, to be sure. Some failure comes with harmful side effects and some failure is catastrophic. But there is a type of failure that you should seek: learning.

As an example, over the past few years my kids and I have been learning the norms of biking in the Netherlands. They are much different than American norms, mainly in that bikes rule the road instead of cars. Often, myself or my kids will do something wrong: a common one at first was coming to a complete stop at an intersection with oncoming cars. We'd stop at the intersection, the bikes behind us would stop, the cars would stop, and we'd deadlock the entire intersection. People got upset and us Texans were left baffled.

What we needed to learn was the more fluid nature of an intersection, to recognize when cars needed to yield and when not and build up the intuition of how all these moving parts fit together. Things get more complicated when two–or more!—bike lanes intersect. We often failed at this, and my kids would get very upset: sometimes they would get off their bike and throw it down, storming off. I'm not sure where they learned this from...ahem. I've been telling them that this is a great learning moment, that they've gained valuable knowledge. And, indeed, after many initial "failures," they're much better at fitting into the flow of biking now—and me, too!

Equating failure to learning seems cutesy, sure. However, when you think that some idea or task has "failed," catch yourself and ask if this failure is actually just the part of learning or, less likely, catastrophic failure. There are types of failures that are clearly bad—production crashes, security breaches, running out of printer paper before the big meeting, and so on. But if a new feature you've introduced in your app isn't getting the result you expected, think of that kind of failure as learning. Even in the case of "bad" failure, there's often much to be learned for next time.

One of the favorite mindset-shifts product people like to point out is that a series of small learnings controls the risk of big, fatal failure. Figure 1 illustrates this concept.

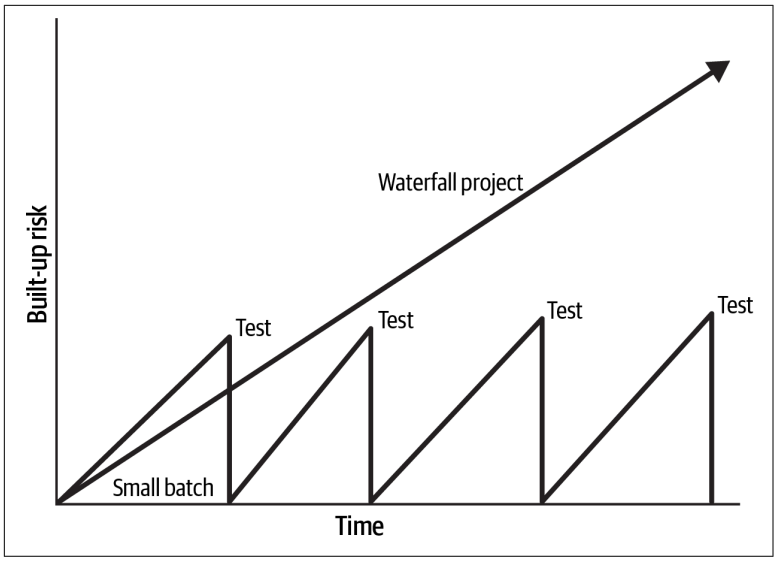

Figure 1. Trickle of risk versus big-bang risk

The zigzag line shows a series of small releases, each validating or invalidating the theory of new features. If the feature turns out to be not so useful, you've had a minimum amount of "failure" and can move. This is in contrast to the straight line pointing up at a 45-degree angle. Because you're delaying releasing software, you accumulate more and more features which are assumptions of what's valuable. When you release this big batch, you have a pile of potential "failures" that could create a much bigger pile of failure than that trickle of learning with smaller releases. Doing frequent, small releases lets you manage risk around learning and innovation.

This trickle of failures—ahem, pardon me—*earning* is exactly what the product approach to software wants and is based around! Most ideas will "fail," to be sure, but if you don't try new things, you'll never come up with apps and features that help your business and customers.

Leo Lawrence of Air France-KLM shows an example of this thinking and the business benefits:

> About 10 percent of the ideas actually end up on the market. A recent example of this is the hand baggage check in the KLM app. Travelers can use augmented reality to see whether their hand luggage meets the set dimensions. This function went live last month.[11]

When traveling, I've been stressed out by that problem many times, and features like this improve my customer experience, or put more simply, make my life easier. Similarly, the company introduced 3D views of the business class seats when checking in online "with the idea of stimulating the sale of these chairs," Lawrence says. Even if 90% of these teams' ideas "failed," they learned their way to several good ones that directly improved the business and led to new revenue.

Management needs to help change people's mindsets over to embracing learning, and also to feeling that it's safe. For example, executives can help out a great deal by simply asking "What did you learn this week?" rather than "What's the status of the backlog?"

As ever, with this kind of shift, management needs to be the first ones to show the new way of thinking: they should start telling their organization things they've learned, both "successes" and "failures." Finding these learnings should be easy: you'll be experimenting with all sorts of new ideas and trying out new practices and technologies as you transform your organization. Sharing how these go with your organization will show that your mindset has shifted "failure" to "learning." As Richard Watson puts it, this is shifting from defining success as being right in our predictions to success in learning why our predictions were wrong, and hopefully, eventually, right and helpful.

Traditional finance models pose the next challenge to shifting to this small learnings and pivots approach. Traditional corporate finance requires up-front promises of return on investment and planning in 12-month increments. As I argued in *The Business Bottleneck*, when you account for the preplanning and building up alliances for annual budgeting, this window is actually more like 18 months. That

11 This is translated from the original Dutch with Google Translate.

period of time will be a waterfall that drowns out many of your efforts.

Sadly, there are not that many happy stories about transforming finance. That task is a new, ongoing challenge for organizations that want to change to product-driven software. Short of solving the problem, I've seen organizations do two things:

- Put in a layer of abstraction, a "facade" even, between your software organization and finance. You still plan for and take large, annual chunks of money, but set up a more product-driven compatible funding model. This doesn't solve the problem, but keeps you moving.
- The CEO or board approves mavericks to experiment with new finance methods. You should probably not rely on this for more than an experiment that, if successful, you'll bring back to the mainline business. Otherwise, as discussed in "Take everyone with you" on page 46, this brings in the problem of creating a "NewCo" that breeds jealousy and uncooperative staff in the "legacy" business.

Finance is going to be a problem, no doubt about it. For further discussion of how to try to change your finance people's minds, see my previous report, *The Business Bottleneck*.

Avoid over-learning; or, Zombie governance

While it's important to understand and learn from "failure," you need to be careful of what I'll call over-learning. You see this frequently when there's too much governance leading to too many meetings you need to go through to change something. This over-learning means that you're trying to prevent all problems you've encountered in the past. After several years, the problems aren't likely to occur again and you're spending time on irrelevant fears.

I think of this like mortgages. In the US at least, mortgage documents go into what seems like hundreds of pages. If you take the time to read them, you realize that they document every single problem that's ever occurred with a mortgage going back to Hammurabi. These mortgage documents are accepted absurdities in American life: no one reads them, you're not going to get changes made to them, so if you want a home loan, you just sign them.

Changing Habits

A lot of what we're talking about when we discuss culture change is changing habits. If you think about habits, they've been instilled over time based on a reward cycle. You keep doing what works, or at least because it worked well at some point. Because habits are enforced by success, they're like a river. There was first a little drip, then a little flow of water that slowly, very slowly, carved out a channel. Once that channel is carved out, it's pretty much set in its ways. When you're changing habits, you're trying to change the course of a river, maybe even filling in a canyon!

And it's really hard to divert and change it. So, think about that metaphor when you're trying to change the way that people operate and change the way they think—you also need to slowly, but surely, carve away at something. That is, when you're transforming how your organization does software, you need to create new habits with positive reinforcement. You need to make doing the new things easy and rewarding, slowly shifting from the old ways.

There's been a lot of interest and research into habit-forming and training recently. The book *Atomic Habits*[12] is a great overview. As the book discusses, when you're trying to change a habit, what's important is to do small, incremental changes and new actions every day. It doesn't really matter if you're successful or not at any given time or task. But doing these habits daily is what lets you practice and pick up those habits.

Pairing is a good example of this habit-forming loop. In pair programming, developers work in pairs: "two keyboards, two mice—one computer."[13] Having two sets of ideas and brains working on the wicked problems of coding has been shown to result in higher quality, more frequent software. Pairing helps spread knowledge in the team, preventing fief building, bottlenecking on The One Person Who Knows How It All Works. Pairing is, as described in "Favor seeding change over big bang change" on page 45, a key tool for scaling up change with the seeding method.

12 James Clear, *Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones* (Avery, 2018).

13 As described in *Radically Collaborative Patterns for Software Makers*. See that book for much more discussion on pairing, especially addressing that initial reaction many people have: "That sounds bonkers!"

Using the "start small" approach to building new habits, every day, you could do a little bit of pairing: first thirty minutes, then a few days later an hour, and so forth until you get up to four or five hours a day. As you're doing this, don't focus on doing it "right," focus on learning and simply doing it every day. You'll get better at it over time. But more importantly, you'll build up that habit.

This same starting small, but frequently, approach can be used to shift mindsets, redirect, and carve new rivers.

Motivation Hacks

Your job when you're leading a transformation is to change how people behave. A key problem is that people don't want to change. The success of corporate change initiatives is hard to gauge, but the success rate is rocky enough that it can seem scary to most employees. Also, people may not believe that change is needed. The *status quo* is there because it worked: the way people operate—your culture—has gotten the organization this far. For most of the people in that organization, things are going great. They get paid; their career is advancing; they can take 90-minute lunches; their work is fulfilling. Whatever the case is, highly skilled knowledge workers tend to be satisfied in their job. Skilled tech workers are in such high demand that they can choose to leave. Some don't, even if they're grumpy, because there's little downside to staying in their current job, but also because there's little upside to changing.

Changing is difficult and carries risk. What if it doesn't work? When new corporate ideas have failed in the past, people were punished by being passed over for promotions, and if things went especially poorly, allowed to go "spend more time with family" or take on one of those "strategic advisory" roles.

While upper-level management may see the danger of staying the same in the face of fierce competition and market headwinds, most people don't care about "headwinds." The tall walls of their cubicle hold back the wind.

Finding ways to motivate people, then, is one of management's chief problems.

Positive feedback; or We can all agree on money

Industry folklore says that money isn't the chief motivator. In my experience, this is true once a baseline of compensation and career management is in place. Many nontech companies I talk with regard their IT workers as "cost centers," ever on the lookout to outsource them to bring down costs. Technical career ladders are often short. The pay given in nontech companies is often much smaller than actual tech companies. Staff at nontech companies are rarely given large equity grants. Compared to the high pay and common practice of equity grants in tech companies, IT staff at regular enterprises can feel like they're underpaid. Money only "doesn't matter" to people who already have it.

Let's look at ways of using money to motivate.

Money. Monetary rewards are one of the better ways to motivate people to change. First, it demonstrates that the organization is serious about change. Changing compensation structures is difficult and working on that will show management's commitment to change. Second, awards are a very clear way to give people feedback that they're doing the right thing. Third, everyone likes rewards, especially money.

Revisit your awards. Most organizations already have a reward system in place. However, these systems usually only reward triumph over adversity and short-term heroics. Tying rewards to delivering actual business value doesn't seem like a common practice when it comes to IT staff. The culture of innovation we're interested in consists largely of failure, or "learning," as people in more caustic organizations always tell me to put it.

Rewarding the Boring. To motivate people in that system, you can't just reward the rare success and against-all-odds triumphs: We fixed production at 3 in the morning! Even more than being thankful for such efforts, you need to reward steady success with the process. If you reward only the "heroics" of solving critical errors and problems, you'll probably encourage your staff to *only* focus on that type of work, devaluing the work it would take to prevent such problems in the first place.

In addition to looking at what activities you reward, look at the actual reward itself.

Don't just rely on gold stars. Many existing rewards are the equivalent of gold stars in kindergarten: of no monetary value. I want to posit the theory that in a for-profit organization, rewards that don't include money are not rewards. Ask yourself: Wouldn't you like to have money rather than a Galileo thermometer etched with the phrase "Best Innovator of 2QFY23"? The same is likely true for nonprofit organizations, but often people in these organizations really are in it for something other than the money. Still, money never hurts.[14]

Grant equity. Another monetary reward to consider is equity in a company. Stock options of restricted stock units (RSUs) is a powerful motivator. Plus, equity is long-term and business outcome focused. Equity is worth more when the company as a whole is successful over the long-term, rather than just one part of the company in the short-term. Senior executives and boards understand this when they use huge equity grants as part of executive compensation: they want executives to care for the entire company's business outcomes. This works for employees as well. I've worked at tech companies all my life where equity is a common type of compensation. Over the years, my coworkers and myself were all very aligned to the overall success of the company, to business outcomes.

Clearly, getting a budget for more monetary rewards and compensation can be difficult. I have no balm for that. It is difficult. The only guidance is to strongly connect metrics to the mechanisms that drive bonus-funding and do the work to change compensation-think if needed. Those market headwinds, though!

Beyond gold stars

There's also what people call "soft compensation." Mostly, this seems to be "not money." Software people are highly skilled workers and in chronic short supply. They can demand a lot, and, if they're not satisfied, can usually find another job quickly. So, job satisfaction is something they can take seriously in addition to raw pay and simply being employed.

Let's look at some examples of nonmonetary motivators.

---

14 In the 2020 Stack Overflow developer survey, "better compensation" was the #1 reason for looking for a new job.

Doing quality work. In my experience, the product-centric way of working is very rewarding to people, boosting their job satisfaction. Getting closer to the person using your software, seeing those users benefit from the features that developers added is meaningful and rewarding. The product team will also be elated to see the rest of their organization jump with glee as they craft and ship software that helps the business. Most people enjoy being useful. Very quickly, the small batch process becomes a virtuous cycle. That rapid feedback gives your product team a weekly sense of how all their work is paying off and making people's lives better.

Flow. People often underestimate the value of speeding up release cycles, going from idea to coding to running in production. Technical staff, especially developers, love removing toil and speeding up releases. Like most people, software people enjoy being in a state of "flow": removing obstacles to doing their work and moving at a slightly rapid, slightly challenging pace without having to bang their head constantly against bureaucratic walls. This means speeding up their path to production and eliminating as many meetings and governance checks as possible. As we'll discuss later, this doesn't mean skipping such governance, but instead focusing on automating it. As one of my past bosses, Andrew Clay Shafer, always put it, developers love a small mean time to dopamine. Reducing that "mean time" will be incredibly motivating.

Quality of life. Despite the stereotype of late-night coders who have little interest outside of work, quality of life ranks high for IT staff. While not complete, my definition of "quality of life" is "not working more than I want to." The more you work, the less time you have for life. An easy type of reward, then, is to reduce the amount of overtime people work. In the 2020 Stack Overflow developer survey, 75% of developers said they worked overtime at least one to two days per quarter, while 25% said they work one, two, or more days over a week! It's not just developers, of course, but everyone involved in your software's end-to-end process.

To motivate people to change, start pointing out how the new practices, tools, and even enterprise governance results in less overtime and less time working. Be careful, though, to avoid falling into the "work smarter, work longer" trap. Once you can do the same amount of work in less time, don't just pile on even more work! Few

people will be motivated to change if it means they need to work more.

Open source contributions. When it comes to developers, you can also use the soft compensation of working on open source projects. Building up a track record in open source projects is both good for developer's career management and also gets them positive praise and attention from other developers. Squirrelly as they may be about it, like anyone else, developers appreciate praise from their peers.

Autonomy. Unless you're a golem, you probably don't like people telling you what to do. Your staff are the same. They, of course, want to work toward the common product goals you have and collaborate. But most of the people involved in your software's end-to-end process would like a huge degree of freedom in choosing the tools they use. Looking at the Stack Overflow survey again, "languages, frameworks, and other technologies I'd be working with" is the top "soft benefit" in choosing a job:[15] every craftsperson values their tools. Now, allowing each product team to choose their different, unique tools is a quick trip to disaster-by-variability. Organizations need to balance limiting choice and chaos, putting in place guardrails within which staff can choose what to use. Autonomy doesn't just mean tool choice, though. In fact, the more valuable type of autonomy is the autonomy given to figure out which problems to solve and how, as we say in "Case study: Measuring mayonnaise" on page 10.

Reward moving pixels on the screen. When it comes to open source and autonomy, there are two traps to watch out for with developers.

First, you need to make sure that developer's spend time on what's valuable to the business: their product. When it comes to customer-facing apps, I like to think of this as "moving pixels on the screen":[16] changes made to customer-facing applications likely directly drive business value. As a rule of thumb, then, try to focus developers on moving pixels rather than building frameworks and infrastructure.

15 To be specific in what "top" means, the survey question "asked the survey respondents if we control for compensation, benefits, and location, what three characteristics would most influence their decision to choose one job offer over another."

16 My friend Robert Brook, enterprise architect at UK Parliament coined this phrase over a breakfast of oatmeal and blood pudding a few years ago.

Developers often can't help themselves, though, and tend to favor writing code that has little to do with pixel moving. Indeed, the most respected developers tend to work on back-end frameworks and systems: that is, the very tools other developers use.[17]

Second, when it comes to autonomy, you can't have each product team choosing their own stack of software. In a large enterprise, variability is a productivity and resilience killer. Centralizing and standardizing the back-end code and systems creates many, many of the productivity and security benefits you'll need. Indeed, at revered companies like Facebook, Google, and Netflix, developers are typically *not* free to choose the programming languages or frameworks and software development tools they use. These companies maintain a good, often updated stable of tools and services, but developers are required to choose from that stable. They say that constraints drive artistic creativity, and the same is true for the creative act of making software.

Management's New Mindset for Staff

In a product-focused organization, you're setting up each product team to be the "owners" of the software or the product they're building. Following part of lean manufacturing thinking, the idea is that those closest to the work are the ones who know how to best do the work. And, once you've connected the product team closely to the "customers" of the software, they're even more situated to know what's best for the software. The product team may even know better than the business, as examples like "Case study: Measuring mayonnaise" on page 10 show.

As a manager, this means shifting how you think about your working relationship to staff. In the project-mindset, management is "herding cats," trying to control a chaotic, resistant group of people to a specified end—often, feature set, schedule, and budget. Over the years, we've made this notion of "herding cats" into a cutesy phrase but packed into that little phrase is an exasperated manager who's

17 No generalized statement goes without back-peddling. Yes, there's a tremendous amount of "back-end" code that helps move those pixels on the screen, especially performance and security. Working on developer tools that improve developer productivity is valuable. However, if you find that each of your developer teams is working on such code, something is likely going wrong.

actually saying, "I can't get these developers to do what I tell them!" And it's that mindset that must change—perhaps you're not there to tell them what to do.

Prescribing how to get to that management mindset is fuzzy, but there's a short list of what it looks like once you've stopped thinking about your teams as unruly cats. There are three things you give over to product teams:[18]

Trust

A manager trusts that the teams will solve the right problems in a way that meets the business goals. This can be the most difficult mindset for management to make; management is often used to command-and-control where people are told exactly what to do—that *project* mindset. With the kitchen example, management trusted that the team could discover the most useful problem to address—measuring the mayonnaise. This trust isn't blind or free of responsibility. Management's ongoing trust is based on the team's accountability to producing business value. Both management and the team need to establish the appropriate metrics to constantly reinforce this trust.

Voice

The product team is given the ability to speak up about what they think is the right next step, the right focus, or just an idea to experiment with—and even to object to what management is telling them, without fear of punishment. In the kitchen example, the team was given a voice to speak up and say that digitizing recipes books wasn't as important as measuring the mayonnaise.

Autonomy

Management allows—trusts!—that the product team can operate on their own and make their own decisions, without micromanaging their every move. Management is, perhaps, more interested in verifying that the customer and business benefits, rather than exactly *how* the product team writes software and manages their process. Management also needs to clear the way and time for that autonomy by removing barriers for the team.

18 These are recommended in DORA's 2018 *Accelerate: State of DevOps* report. Also, see further discussion of these three items in *Monolithic Transformation*.

> Take on the mindset that you're constantly clearing the field for the team to work in rather than controlling the work they do.

You'll have to keep a close watch on your default management reflexes as you cede this much control over to your staff. In fact, try to train yourself to be comfortable with delegating. You can start with small decisions at first, and then slowly go up to larger and larger decisions.

The fish smells from the head

Pushing down all these responsibilities can seem disempowering for some managers, scary even. You'll have to work with "middle management" to get over this fear. Look at yourself and other executives and make sure they're being genuine about shifting responsibilities down. Management needs to change over from the idea that they know and control everything into the mentality of a learning organization, as we've been discussing here.

"I've learned this funny sentence living in the UK: 'The fish smells from the head,'" as Jana Werner, then Head of Transformation at Tesco Bank, started describing this mindset shift to me recently. Leadership has "to ask questions and to not know the answer, and get comfortable with uncertainty. The big switch is from thinking that a leader needs to be seen to know everything, understand everything, and have an answer to everything." That's a lot to ask for individual staff, but especially for managers who've built their career on those three everythings. Leadership needs to demonstrate this, even contrive situations that show them acting in this new way.

"The people working for these leaders also think they need to know everything and understand everything and have everything in control," Werner goes on, describing how important it is for upper management to model this mind shift. "If you [upper management] can be seen as vulnerable and comfortable with uncertainty," she goes on, "and with finding your way through it, then the people that work with you can get into that same space and be more comfortable sharing."

Otherwise, you'll quickly encounter the infamous "frozen middle": an obstinate middle-management layer that blocks your dreams of change because they don't believe it, or maybe even fear the change.

Trust Your Team

To risk being political, management's higher pay and taller career ladder than for individual contributors shows that most organizations value management far more than individuals. As an individual contributor, at least, it *feels* like that. Whether we try to be progressive or not, this HR signaling can create the HiPPO antipattern: the Highest Paid Person's Opinion wins the meeting, no matter how ornate and empirical the slides and charts from the "lower-level" staff are.

Relinquishing that unconscious mindset is extremely difficult, but it's key to getting people on your product team to take ownership of their software and start innovating, instead of just doing what they're told. I'm not saying that you, management, shouldn't give input and direction—indeed, I'm a huge fan of management enforcing the terms of the system, setting the constraints. You'll need all the ideas, experiments, and tenacity you can get, so make sure you encourage thinking, ideas, and contribution from all your team.

For building and maintaining that trust, see "Use Metrics to Build Trust" on page 35.

Servant Leadership

The concept of "servant leadership" is very popular now. The idea of managers enabling, rather than, well, "managing" the staff on their team is clearly key to the type of culture we're shooting for here. Don't tell anyone, but, despite many people giving me the, uh, "opportunity," I don't actually like managing. Giving advice *to* managers, for sure! So, to go over what servant leadership is, I asked one of the best servant leaders I know, my boss, Tasha Isenberg, to explain her mindset and, of course, the related management tactics.

Case study: Servant leadership according to Tasha, by Tasha

My job as a manager is to ensure my people succeed. If my people succeed, everyone wins. The company's objectives are more likely to get met, my people are happier and satisfied, and I'm more fulfilled when my people get credit for the meaningful work they do!

So, everything I do is to ensure my people have the tools needed to be successful. What does it mean to ensure your people have what they need to succeed? What do you actually *do*? Here are a few

things I have been practicing through the years that have made a difference.

1. Trust your people. This is the most important thing to remember and do. I make every decision with the belief that my people will do the right thing.

I get that it's easy to say you trust your team, but what does that really mean? First, I don't just blindly trust someone; I establish and foster mutual trust with them over time. But how do you start with a new hire, or if you are starting with a new team?

I think the key to building trust, if you don't know how to start, is simply to give your person a project. Lay out key objectives and timelines together, then keep tabs on their progress. I don't mean micromanaging here; I mean leading with empathy. Believe in them and encourage them; check in and see how you can help them; talk with them about how they are struggling and then help them find an answer. Even if you don't know the answer, give them resources or access to people who would know the answer. Equally important: give them time and permission to figure it out. Trust is a mindset; if you don't trust your people, why are you even a leader in the first place? Put another way, establishing trust with your people is part of your core job: you have to work at it, rather than just expect it to happen or be plopped on your lap.

2. Clear the crap. If you want your people to succeed, you need to give them the time to do that. One of the most important parts of my job is protecting my people's calendars. I act as a filter between my team and the rest of the organization, and, really, the rest of the world. At all times, I know who is working on what, who is overwhelmed, who wants more work, and I adjust projects and requests for their time as needed.

Another part of this is helping my people keep up with what's going on in our organization. In an organization as big as ours, there's always *a lot* going on. I consume as much knowledge as I can when it comes to general company updates, strategy, policy, and other company goings on, so that I can filter it to fit what is important to my team and keep them updated. Because of this, my team goes to less meetings, freeing their time to be productive with their latest project they are working on. They're able to spend their time creating the most value for the company.

3. Give people passion projects. Do you want developers to be happy? Give them support and time to work on their passion projects. Whether that is contributing to an open source project or working on something completely not related to their actual work, acknowledging their passion projects, taking an interest in them, and most importantly, giving them time to work on them is an investment that will pay off in your team's job satisfaction and retention.[19]

4. Share feedback. In my experience, showing my people how their work is actually making a difference is incredibly motivating. So, I make it a point to seek out feedback and then share it with them. Give concrete, consistent examples of how they are moving the company needle and delighting people, whether it's coworkers, people in their community, or customers. If people outside your team aren't offering that feedback, go find it yourself. Be your people's strongest advocate. Hold them up on your shoulders so they can shine the brightest.

5. Offer incentives. At the end of the day, people want money. That's what's most important. Does your company have an incentive program like an on-the-spot award? Find out what rewards and incentives are available and always work to get your team those awards. Tell everyone to expense their lunch sometimes, help them get the equipment they need to perform at their best; when there are stressful events and times give your people work flexibility or time off. It may seem like adding all these little things up will cost too much, but in fact, all these small things add up to a happy, motivated, dedicated, harder-working team that you can trust to do the right thing for the company.

6. We Are Family! (You know, the song!). Your team is dedicating most of their time to this job. They interact with others on your team sometimes more than they do with their own family. You need to create a team environment where people actually want to be around one another. Your team should be built around supporting each

19 Coté: This is slightly contradictory with my "moving pixels on the screen" imploring. However, I believe Tasha is right here: you need some "fun" projects and autonomy for your developers and other staff. Many of these can be carefully directed into being overall helpful for the business, for example, publishing design guides and contributing to open source that fills a gap in infrastructure you use, such as for compliance.

other, wanting to share ideas, and work together. Create time for watercooler talk and virtual team events where everyone can get to know one another on a personal level, outside of work. Find out what each person likes and create a small team event around that topic and have that person take the lead!

Just recently, for example, one of my people who is an Australian and passionate foodie, organized for everyone to get an Australian-themed treat box. He hosted an online event where he told us about each treat as we ate them together. It was a blast! Events like this make the team happier and stronger. After all, when your team has eaten vegemite together, few things will be able to pull them apart.

Revisit Your Bozo Bits

I've noticed that management often holds a dim view of staff as well. They wouldn't put it this way—but, you know me, if nothing else, I'm expert at turning anything into a cynical view. You see this dim view of staff year after year, survey after survey, about new technology adoption: each year, management says that lack of skills is one of their top three barriers to change.

Most of the people responding to these surveys have lost trust and belief in their staff. It's as if management has flipped the bozo bit *on their own staff*. The bozo bit is an unforgiving pattern described early in the software development lore. It says that once a person commits some, well, stupid act, others will "flip the bozo bit" on them, permanently thinking "anything this person does is stupid, and they should be ignored: they're a bozo."[20]

The bozo bit is a wickedly funny bit of computer nerd humor (we're, you know, not a kind bunch, ahem...), but people often don't look past the punch line. The point of the bozo bit is that it's an *antipattern*: it's not good to write people off! My suggestion is that you simply be aware of people who've flipped your bozo bit and force yourself to revisit those bozos, maybe every 30 or 90 days.

If skills are such a problem, and hiring new staff is such a blocker, you need to look at the existing staff you have as the best, easiest-to-grow staff. This will mean trying out new programs rather than

20 The definition of the bozo bit is attributed to Jim McCarthy and Denis Gilbert, in *Dynamics of Software Development* (Microsoft Press, 1995).

sticking with your current training program—it's not getting results, after all! As BT's Rajesh Premchandran says:

> You can't have a top-down diktat saying "march towards the cloud and let developers figure out their own learning part." We've invested heavily in certification programs, getting online e-learning platforms that allow career paths to be developed around the cloud and agile and DevOps and all of the associated skills.

Indeed, isn't one of management's top responsibilities ensuring that their staff have the tools, knowledge, and ability to do the best job possible?

Trust Your System

Following the rigor of the small batch process provides a safety net for trusting product teams. When you've put in place the mindsets described in this report, in particular the rigor of the small batch process, you can trust the small batch model, along with trusting that the team is following it.

The small batch process recursively proves itself in each cycle: did something valuable occur with each release, or are we still confused about the value of any given feature? If you have no idea, you can see if the steps in the process were followed: find a problem, create a theory of how to solve that problem, specify how you would measure success, write and deploy code to actual users, measure the success of the theory or not. If the product team is not following those steps, then you've found a fix, and can try it again. Over time, if the small batch process doesn't get results, then you've proven that it doesn't work in your organization—you can try something else! If the process does work, you've got proof that trusting your staff is a good idea.

To sum up: by trusting the system, you can trust the staff.

Shift Everything Left; or, the End-to-End Mindset

Value streams are one of the key insights and tools software thinkers plucked from lean manufacturing. In software the lean concept of value streams has come to mean something more general than in manufacturing, but plenty helpful: all the activities performed go from a feature idea to people using that feature in your software. We sometimes call this a "build pipeline," "software supply chain," or a

"path to production"...whatever you like: the industry plays around with new terms every few years.

For us, the important part of a value stream is the big-picture, end-to-end mindset. In my estimation, each time the software development process takes on more activities in the path to production, software quality and overall usefulness improve. Agile software development and DevOps are examples of this effect.

Around the year 2000, agile practices like Extreme Programming (XP) and Scrum widened the end-to-end view beyond just coding. Agile brought in project and product management (developers worked on stories and embedded product managers on their teams) and QA (developers not only wrote unit tests but started working on acceptance tests). Next, early DevOps thinking in the late 2000s introduced the idea that all the configuration, monitoring, and even infrastructure needed to run an application in production was *part of the application itself*. This was the notion of "infrastructure as code." Now, the end-to-end view has come to encompass everything: building, running, and managing software.

At first, you may think that this means that the product team controls and does the work for that entire process. While small startups or isolated teams in large organizations can be full-stream owners, in large enterprises this isn't practical. There are back-office and enterprise resource planning (ERP) systems to integrate with, line-of-business owners to work with, different geographic regions to operate in, and so on. However, someone needs to "own" and tightly manage the business outcome of application.

I rarely find a person or team who's responsible for that outcome end-to-end. For example, I was speaking with a group of ten or so senior IT managers who represented a large bank's various IT functions: everything from end-user computing, to networking, to developer infrastructure. They each seemed to execute their responsibilities well, but they complained that it was hard to coordinate across silos. There were too many hand-offs; groups couldn't coordinate technology decisions.

This complaining comes up *all the time* in discussions I have with large organizations. To me, this smacks of a headless pipeline: there isn't anyone who owns and is responsible for discrete business outcomes and also can make management decisions about all the activities in the path to production. No one seems to "own" the entire

"factory," so to speak. Sure, the CIO is ultimately responsible for "everything," but at a large bank, it's unreasonable to think that a CIO could actually be that hands on across hundreds of applications while also worrying about every other IT concern.

If heads of departments keep complaining about silos and can't seem to coordinate, you probably have a headless pipeline. Rather than create yet another silo to solve this problem, you should be the first owner for the end-to-end process. This starts with putting in the work to discover and document the path to production. First, pick an important application, like search on retail, loan applications in banking, or one-off unemployment payments in government. Then, get a large whiteboard and chart out every activity required to get a simple feature, maybe even just one line of code, out the door. Key to this is finding out and tracking how long each activity takes, including hand-offs between groups. This will take time: you'll have to go rustle the bushes to find these numbers and verify them.[21]

Once you've drawn this entire flow, you'll likely find that there is, as one executive put it, "a whole lot of stupid up there." You are the owner of that stupid and you are the one who can manage and direct fixing it. If you assign that task to one of your staff, or worse, one of the groups in your organization, they won't get beyond their own department. The critical mindset shift here is twofold:

- There is an end-to-end path to production, and likely no one has ever discovered and charted it.
- That path to production must be owned, and activity managed to perfect how software delivers business value.

More than likely, you'll have a few optimization epiphanies:

- Despite years of automation, we still have so many manual and ticket-driven processes. Each silo may be optimized and automated (or not!) but connecting all that automation together is often lacking.
- We have excessive governance, resulting in long wait times between activities in the path to production.

21 For a starter kit, check out Matthew Gunter's "Crossing the Value Stream."

- While valuable, security checks and policy slow down our process, especially when security is prescribed at the beginning and then verified at the end of the process.

Recently, I've noticed the last two the most. As organizations are scaling up their new method of doing software, they're banging up against those last two points—governance and security. These two functions have yet to fully "shift left" like QA and operations in agile and DevOps. In each case, successful organizations are spending the time to work with auditors and security staff earlier in the process, hence, "shift left." While these relationships have been oppositional in the past, working together earlier in the process removes much of that opposition and serves both the needs of the auditors and the product team.

Cloud-native technologies and practices like Kubernetes and buildpacks are also helping speed up and make better security and governance. With these new platforms, you can tightly control and verify what code goes into production, giving security more control and assurance over your software. Using cloud-native technology like this to put up guardrails and automate governance enforcement is broadly called "DevSecOps" and has many successful examples to draw from, including from high-security organizations like the US Department of Defense.

Start Small, Stay Small

Without exception, every executive I talk to thinks they have a key, unique problem that's preventing them from changing: their portfolio of apps, their organization, everything about them is much too big to hope to change. Many people at large organizations that I talk with are experts at telling me why their size and organization's age makes it too hard for them to change how the organization operates. They follow Eeyore's maxim: "it's not much of a tail, but I'm sort of attached to it."

Being an outsider, my perspective is that, actually, every executive at every organization thinks they're the only Eeyore in the world. In fact, the impossibility of change is a universal "problem." I put the word problem in scare quotes because the successful organizations had and have these same problems. The successful organizations figured out how to move past Eeyore's defeatist attitude.

Being too big should never prevent you from being successful. In fact, large organizations have huge advantages: well understood businesses and existing customer bases, competitive advantages, large pools of money and funding, brand names that drive awareness, easy credit and fundraising abilities, and scale that means even small improvements drive large revenue and profit improvements.

When it comes to software, the mindset shift successful organizations go through is thinking much smaller than they're used to. Large, mature organizations are used to large projects. They start off by telling you their customer base size, the number of employees they have, their large revenue, the number of applications they manage, how many petabytes of data they process. Organizations like to boast about how big of a deal they are. This is fine: it's healthy for us all to think that we're important.[22]

This mindset, however, is once again a leaky mindset: your sense of bigness will negatively affect the way you think about your software portfolio and, worse, about your abilities to improve.

First, you tend to think that you need to do everything at once. At the very least, you get stuck in analysis paralysis again by the sheer scope of everything: how could we possibly replatform all that COBOL to Rust?

Second, your friends in finance also will cause problems. Changing how you do software is expensive, but only in the short term. Finance will want you to put together a business case that shows an excellent profit versus alternatives, including doing nothing. As you're putting together your annual financial planning spreadsheet, late at night, eating another round of kapsalon with your team, you'll eventually give into the urge to show that changing over all those COBOL apps in the first year will result in tremendous payoffs. Only paying attention to short-term return on investment (ROI) will limit you to short-term success.

Third, your organization may be tempted to just cart the whole project off to an outsourcer who promises skills and risk management at tackling such a large portfolio at a lower cost than your employees. What was once an Azathoth-scale problem is now

22 My therapist would likely want me to say "...to *know* that we're important…," but I'd like to keep them employed longer so I'm not ready for that healthy diction yet.

someone else's problem. The last thirty or more years of outsourcing this scale of a portfolio has mixed results. There are plenty of good staff working at outsourcers, but what you'll find when you adopt a product mindset is that you want to employ staff ongoing, as part of your organization, rather than being dependent on another organization whose incentives are not aligned perfectly with yours. Rather than focusing purely on cost, begin every outsourcing strategy and discussion with the assumption that the outsourcers are there to help transform your organization, to teach *you* by working with you on applications. Once your staff are skilled, the outsourcers should leave. Don't cry for the outsourcers: there are plenty of other organizations that need such help.

Successful organizations change their mindset to start and stay as small as possible. They follow the maxim that a journey of a thousand miles starts with one step. Management at these organizations pick one app to develop from scratch or modernize, then another, then maybe two more, then five more, and so on. This gets them boot-strapped and gets them started.

More importantly, it allows the management to learn what works. When you're switching over from a project to a product mindset, you won't know exactly what to do, both meatware- and software-wise. You'll need to apply the small batch process to learn. This is especially true for learning the truly unique needs of your organization.

Use Metrics to Build Trust

Metrics are an important tool for transforming your organization. As with any type of management, metrics will often be one of the primary, consistent types of feedback you get about how your organization is functioning and how your change initiatives are doing. Metrics will tell you, in short, how *you're* doing at your job. Let's look at three ways to use metrics and then a few examples of metrics specific to culture change.

Three types of metrics

First, metrics show progress on the path to your new way of operating. How much value are we contributing to the business? How much easier are we making the lives of our users? See "Business success is the ultimate measure" on page 37 for more.

Second, metrics give you feedback on which changes work and don't work. This is core to the small batch loop and being a learning organization. Not all learning can be quantified with before/after metrics or even hypothesis-driven change but try to measure and quantify operating changes as much as possible to track what works and doesn't work.

Third, and maybe most importantly, you can use metrics to build trust. When you're transparent with the reasoning and evolution of your metrics, staff will feel more comfortable and safer. They'll understand how they're being managed, but also see that you're adapting to what works. When you share why you've chosen each metric and how you evolve them, staff will see that you too are learning and adapting: you're doing what you're asking them to do. Don't just publish dashboards, publish explanations of what's on them. These metrics are for you, management, but your staff are key stakeholders for the metrics—often, the metrics are *about* them in the first place!

Of course, you'll still use metrics for the traditional reasons: to know how your organization is functioning, report up the management chain, and find trouble areas that need more attention. Since you're already gathering and reporting on metrics, you can use the habit-stacking trick to change behavior: you're already collecting all sorts of metrics, so it's easy to add metrics.

A word of caution here: be sure to prune the metrics you collect regularly, maybe even every six months when you're getting started. Large organizations tend to pile on more and more metrics over the years, as if those gauges and charts are soft bedding to promote better sleep. Each metric is costly. Think of the cognitive load and time it takes to gather the data needed for a metric, remembering what it means, interpreting the metric, and making a decision based on that metric. Even worse, older metrics are often no longer relevant to your current goals but will still drive your current behavior. Make sure that you frequently prune and remove metrics that no longer apply or that are not worth your organization's time to keep alive.[23]

23 This point comes from Rich Lane et al., "Measure What Matters in Modern Technology Operations", Forrester report (February 2021). That report has more suggestions on helpful metrics, including business value-related ones.

Finding the exact metrics that work for you will be a small batch, learning process. Let's look at three metrics and walk-through examples of shifting your mindset about using metrics.

Business success is the ultimate measure

Usually, the most important type of metric to track is how you're positively affecting the business. Is customer satisfaction up because your software is easier to use? Are changes to your software making it easier for people to open new accounts? Do people buy more airline seat upgrades after you've deployed that 3D seat view? People often describe this type of metric as "business outcomes."

Find and use these metrics to report on your software capabilities and changes you're implementing. This "tip" may seem so obvious as to be useless, but one Forrester survey estimated that just 24% of developers measure their success in terms of business value metrics such as customer satisfaction.[24] In some cases, measuring outcomes can be very binary. In an omni-channel strategy, your grocery store can either do curbside returns, or it can't.

Using revenue is often the best business metric to track, but it's not always easy to connect revenue (or savings) to the software used in those businesses. You could track increases in sales or account sign-ups after new features are added to your app, but without proper A/B testing you're playing with correlation fire.

When tracking revenue generation directly isn't possible, many organizations use proxies like the time it takes a "customer" to complete a workflow. Usually, a metric like this is easy to track in software because you can note the time a task was started and the time it was completed. Depending on the task, it could be days, or just minutes. Workflow productivity metrics like these are handy for government agencies that don't really track "revenue." Instead, they can track how long it takes to fulfill various services, like barber license renewals, records requests, and so on: "time to delight

24 Twenty-four percent of surveyed global developers say that their teams focus development teams on business value metrics like sales influenced, revenue generated, or costs avoided today, while 19% of global developers say that their teams use business value metrics like earned value or ROI to measure success and progress. From Jeffrey Hammond, Diego Lo Giudice, and Christopher Condo, "Digital Transformation Requires Development Transformation", Forrester report (December 2020).

citizens" to put it into fun phrasing.[25] These metrics, of course, work well for for-profit organizations also.

For example, in healthcare, you might track people's ongoing completion rate of refill prescriptions as Shields Health Solutions did. Shields helps hospitals and others fill prescriptions. A shocking number of people, about 50% to 65%, don't stick to their prescription schedule, sometimes because it's hard to get a refill. So, when Shields was looking for a product-centric metric to track, they tracked adherence to plan rate, as well as the related metrics that tracked how quick it was to sign up for prescriptions and overall enrollment rates for patients. Using these metrics, the product teams could judge the success of each feature and modification they added, but management could also track the success of their new practices and technologies.[26]

Taking another step back further from measuring business value directly, organizations also use productivity and technical metrics. For example, metrics that track how much work developers are getting done, how few defects get into production, mean time to repair, and other metrics that show the "health" of your software capabilities.[27]

More than likely, you should be tracking these metrics to know your organization's functional health. However, be careful that you don't assume these metrics are sufficient to measure business value. It's easy to use productivity and technical metrics to talk about how skilled your organization is at the work tasks it does, not whether all that work amounts to anything valuable for the business and customers. If you're really fast at delivering garbage, you're still delivering garbage. This is true for measuring culture change as well. For

25 See Chapter 9 in *Digital Transformation at Scale* (London School of Economics and Political Science 2018) by some former UK Digital Service Department staff for more discussion of government metrics.

26 The results were good. The industry average rate for sticking to a prescription plan is somewhere between 50% and 60%. After applying the product-approach to software, Shields saw an 83% efficiency improvement in the patient sign-up process, measured a 46% average increase in patient enrollment rates, and an average adherence rate of 92%. See more in the Shields case study.

27 For much more discussion and advice on these types of metrics, see Nicole Forsgen, Jez Humble, and Gene Kim, *Accelerate: Building and Scaling High-Performing Technology Organizations* (Revolution Press, 2018).

example, one popular metric, training and certifications, "doesn't tell me whether we work in a different way," as Jana Werner says.

eNPS

Employee Net Promoter Score (eNPS) asks employees a simple question: would you recommend this job to your friends? It measures employee satisfaction, more or less, and its simplicity makes it a handy tool.[28] I like eNPS because it adds in more nuance than a simple one-to-five rating. When people answer NPS, they're putting some skin in the game (will my friend regret taking my advice?) which gets them to think more.

You should use eNPS to monitor employee satisfaction, but you can also use it as part of your own experimenting with new meatware. Comcast's Neville George explains with an example:

> We brought in a whole bunch of people from different teams to work on a project for a short period of time. And when that project ended, the eNPS rating was through the roof, people loved it. And it was an indicator for us to say these are things that we should invest in. Because it certainly felt like people loved doing things, learning things new.

As a cautionary note, people who don't want to change and find all this transformation business annoying will give a lower eNPS score. Hopefully, this will be a minority of responses, but be aware that "sticks-in-the-mud" will give low scores.

Staff churn rate

Measuring your organization's retention rate is another useful metric for tracking culture change. If you're building a good organization, more people will want to stay than leave. Monitoring churn will not only tell when things are going wrong, but when the decisions you're making are working.

There's a certain level of churn that you want, though. While I'm not suggesting practices like stack ranking, where you force a ranking of employees and, commonly, layoff something like the bottom 10%, allowing for a small number of employees coming and going has a positive effect on your culture. The positive aspect is getting new

28 The mechanics of NPS are slightly more complex than a yes/no question, but we'll keep it simple here.

people, new ideas, and new practices into the system. These are especially helpful if your organization's culture has been isolated and unchanged for years. Indeed, one of the ways to scale change in large organizations is to encourage internal churn: taking select people from teams and seeding them into new teams, as discussed in "Favor seeding change over big bang change" on page 45.

One large organization put in place an extreme version of churn for their long-term staffing plans. This organization encouraged new college graduates to work on a three-year term to build up their technical skills and get experience with a product-driven mindset. The company couldn't compete with the salaries of tech companies or local companies with deeper pockets, so they used this training as incentive. In fact, the organization expected that many of these three-year employees would churn out to better-paying jobs with this experience. Often, that experience was more training and experience for the person, and several came back to the organization to fill leadership positions. Of course, several of these three-year term people ended up staying as well. This gardening and growth pay off in the long-term, not only bringing in new ideas and people, but also seeding the skill sets needed in the local pool of hires.[29]

Vision, Mission Statements, and Other Things That Should Be Tools

I'm not a big fan of corporate vision and mission statements. They have great intentions, but the need to be all things to all the people often leaves the statements vapid and not useful for daily decision making. They seem like gold plating, even if sincere.

And yet, vision and mission statements are inescapable. If we must have them, you should think about them as tools. Doing so requires shrewd thinking to get the tool just right. As with most things, when it comes to building a product-centric culture, rigorous testing and adapting as you learn what works and doesn't work is key.

Vision

An organization's vision can be a tool used to make decisions. Your vision should describe the problem you're solving for existing

29 This story comes from JT Perry.

customers and new customers. It should guide how people choose between different options.

First, vision can be a shorthand for strategy, telling people what industry or realm of concerns they work in. When it comes to deciding what features to have in your software, what kinds of apps to work on, this filtering out is incredibly powerful.

Developers are always trying to build software that has nothing to do with the business of moving pixels on the screen. They'll tell you they need to build their own middleware, their own build pipelines, and, nowadays, their own Kubernetes platforms. This is—I'll say it—flat-out wrong. If your company's vision and strategy is to supply people groceries, you can exclude "build and maintain a multiregion, general purpose Kubernetes cluster management backplane." That software is available from numerous other sources: you don't need to spend time and money building and maintaining it on your own. That software doesn't move pixels on the screen, food in the supply chain, or tasty food into customer's mouths. Similarly, grocery stores don't build their own refrigeration systems, cash registers, or delivery trucks.[30]

Second, when your product teams are working on the relevant things, vision should help them decide between different options, breaking any deadlock or analysis paralysis. I like to cast this vision-utility into an extreme example: vision should help the product team decide the best of two bad options. That's where a good decision-making tool shines. Choosing between two good options always results in a, well, a good result. When it's a choice between a good and a bad option, there's little thinking needed. But when you have to choose the least worst option you need a precise tool. We'll see an example in "Case study: Better banking by banking less" on page 42.

Third, when crafting your vision, put it in terms of your software's users, not your organization's goals. As the *VMware Tanzu Labs Product Manager Playbook*[31] puts it:

30 Well...mostly. As if to be the exception that proves the rule, Picnic, a Netherlands-based grocery delivery company, has highly specialized delivery trucks that are tailored to their delivery-only business model.

31 The *Playbook* covers additional ideas for crafting a vision, I've only listed a couple here.

> The product vision should answer who the product is intended for, what needs or desires the product satisfies for its users, and what benefit(s) those users can expect to experience by using the product.

The *Playbook* also notes that your vision should be iterative: "[i]f your vision is no longer aspirational or motivational, or if it doesn't ring true for customers, articulate a new and better vision statement." This agility of your vision statement, and even mission statement, is key to evolving. If they stay static or are set in stone (often literally!), changing will always be more difficult.

Finally, your vision and strategy should be achievable. "Don't make it too long-term," Jana Werner says, "it needs to be clear that something can be achieved by people in that strategy, and that's not something that the next generation does." Certainly, you can think in the five-year blocks what traditional strategy-think emphasizes but figure out a way to give your team short-term wins as well. "Winning" early is key to changing habits, even small wins.

Use your vision to focus the product teams on how your software, your business, is making your user's lives better, even if that's just to order food when they don't want to cook.

Case study: Better banking by banking less. The vision I always hold up as an example here is from DBS Bank in Singapore: "we are the bank that wants you to live more, bank less," as Siew Choo Soh put it.

This is a very practical tool because it says, "do the thing that makes our customer's interaction with us as quick as possible." People don't want to hang out in their online banking system like they might in a WhatsApp thread or in Instagram. And, when dealing with money, you often want to act quickly with little hassle. The longer a retail banking transaction takes, the poorer the customer experience. Adding in an address book of previous transfer recipients is helpful —automatically saving previous recipients and auto-completing their information as the user types would be even more helpful.

This "live more, bank less" vision can also drive security decisions as much as feature decisions. First, security is a feature for retail bank customers. To make sure they don't lose money, they'll put up with a lot more annoying security features (like two-factor authentication) than in, say, photo-sharing applications. But those security features should also be driven by the live more, bank less vision.

For example, some banks use a bizarre handheld device for two-factor authentication. In recent years, some of those banks have moved to QR scanning, coupled with a pin number for authentication and authorization. Once banks add in Apple Face ID to this workflow, secure banking becomes even faster, allowing you to get busy living instead of having to get busy banking.

Mission statements and values

If I'm skeptical of vision statements, I'm even more skeptical of mission statements and corporate values. Mission statements and values are supposed to be another compass, almost a set of morals that the organization lives by. Committees and corporate virtue-signaling too easily turn mission statements and values into facile, laminated badge-holder inserts. Of course, my skepticism comes from how many bad vision and mission statements I've encountered. Done well, these statements are powerful tools that act as direction for daily actions.

Decision-making tools. I'm sure it'll rankle some of the mission statement warriors out there, but you should probably start by making your vision and mission statement the same thing. Keeping track of two different things, with one cascading from the other, is a lot to keep track of. Pop quiz! Can you recite your organization's mission statement and vision? If you can, it's probably because it's carved in the wall opposite your desk right now.

Second, make values into operating principles: a list of detailed and prescriptive norms. There's a place for values that speak to character and overall good corporate behavior. Sure, we all value being virtuous, integrity in business, and making people happy, not least of all ourselves. Unless your C-suite was recently carted off to jail for forgetting basic, universal values, use most of your values more as principals: tools for directing how people should make decisions.

For example, the UK's Government Digital Service (UK GDS) documented 10 design principles for software:

1. Start with user needs.
2. Do less.
3. Design with data.
4. Do the hard work to make it simple.

5. Iterate. Then iterate again.
6. This is for everyone.
7. Understand context.
8. Build digital services, not websites.
9. Be consistent, not uniform.
10. Make things open: it makes things better.

If you're used to the usual corporate values, these may seem too "low level"...which is the point! Higher-level values are noble, but often lack the utility that lower-level values do. As some former UK GDS staff wrote in *Digital Transformation at Scale*:

> [T]he principles were not written to replace the civil service's own four long-established and admirable values: honesty, integrity, impartiality, and objectivity. They were written to do something those values were not designed to do—provide instructions for how to actually deliver things.

Values-as-principles should, once again, help product teams make decisions.

Filter talent and retain employees. Mission statements and values can be used to filter the employees you're hiring and reinforce the loyalty and churn of your employees. Because software talent is so rare, employees are obliged to spend a lot of time catering to their employees' identities: software people have so many options for employment that they don't have to take a job just to have a job. They can choose a job because they believe in their employer's purpose and the work they do there. This means that staff can just as easily leave that job when they stop identifying with their employer. Employees' alignment with those values becomes part of their compensation.

You can use values as a way to signal the types of people you want, and those you want to exclude. Do you want hard chargers who will chase business opportunities? You value growth, entrepreneurialism, people who want to "build the future," and solve problems with nonconventional approaches. Do you want people who take a slower, more considerate approach? You value empathetic people who want to learn and investigate, helping the customers enrich their family and lead better lives.

To start, I suggest looking at how you operate today and the type of people you employ. What ways of working are most effective at achieving your goals and create the kind of day-to-day work environment you want? Use those characteristics as your values. If those values seem reprehensible, then, you know—fix them.

Next, think about how you would like to operate, what you aspire to. For example, if you want to shift from the project to product mindset, as outlined earlier, you want your people to be curious, innovative, people-centric, and to take risks. These are things you value. Put those two lists together and see what you have.

As ever, when you're starting, revisit this list often and see if it's working. You could be overly empirical and apply the small batch loop to the values, assigning measurements to take to see if they're "working." I wouldn't have enough passion or energy to do that, but perhaps that's why I'm not management material. In the first year, I'd revisit these values and principles at least every quarter. After that, figure out how often to visit them—twice, or just once a year. Never forget that in the first year, and perhaps beyond, you're learning and adapting.

Using mission statements and values like this will allow you to select out people who don't share your mindset and help keep staff that do have the right mindset.[32]

Scaling Change

In every discussion I have with executives about digital transformation, the same question comes up over and over: how do we scale change? As you know by now, I never pass up on the chance to avoid giving just one answer. In this case, I give three.

Favor seeding change over big bang change

First, as already covered, you have to start small ("Start Small, Stay Small" on page 33). In the context of scaling change, starting small is important because you'll build up an understanding of what works in your organization, but also because you'll "train the trainers." For

32 This idea of using mission statements for staff recruitment and retention comes from an interview with Brian Armstrong, *Conversations with Tyler*, Episode 115 (February 2021).

your initial two or five apps, you should purposefully choose some team members who are, at least, *mildly* outgoing and interested in helping other people. Often, this tutelage tendency is a prerequisite for more senior roles, so you likely know who these people are.

After some definitive, clear success on your initial projects, take these people and use them as "seeds" for new teams. These seed people will be trusted advocates and trainers for your new practices and technologies. Hopefully, coworkers will trust the peers at their same level, and your seed people will have firsthand experience changing from the "as is" to the "to be" state. And, of course, when new people are trained, you get more seed people who can help transform more teams, and so on.[33]

Take everyone with you

Second, focus on transforming your existing organization rather than leaving behind your "legacy" organization. Too often, when IT departments are changing how they do software, they create a different organization, set apart in a different building with much better lighting and pale wood desks. This in itself is fine. However, management needs to be very careful—and genuine!—to make people understand that this sunlight and attractive office furniture is for everyone, not just the chosen few. Moving everyone over may take time, but the intention isn't to leave people behind.

Aside from being, well, humane, there are tactical reasons to do this:

- Hiring new staff is difficult. Hiring new staff that want to work on your "legacy" platforms and frameworks will be even more difficult. If you're worried about the ability to recruit the talent you need, perhaps you shouldn't be so quick to give up on the people you already employ.
- Your new software will need to work closely with your existing software, and slighted people will be the biggest barrier to that integration work. If you're working on a mobile app for changing seats on a train or automating grocery returns, you'll need to work with the teams that own the backend systems that

33 The *2019 Accelerate State of DevOps Report* has some relevant analysis worth looking at of how effective this seeding practice is versus other practices like big bang (all at once), bottoms up, and center of excellence.

handle those requests. If those teams, understandably, don't like you because they feel like you abandoned them in those old, poorly lit cube farms...are they going to make things easier for you? The answer is left as an exercise for the reader.

- The legacy teams will become the new bottleneck in your end-to-end process. Even if the people on the legacy teams share your mindset, having to follow the older process and use the older technology will slow down anything new you try. There are ways of jerry-rigging the interaction between new and old, like putting API facades in place and applying the strangler pattern;[34] but don't get too dependent on short-term fixes.

It may take years to change over all the teams, and perhaps you never will. But don't fall into the trap of thinking about "NewCo" versus "OldCo." You don't really need to prove that the NewCo model will work. As Richard Watson put it, we know that starting from scratch without the constraints of the existing organizations works—that's what a tech startup is!

Instead of focusing on just a "NewCo," create an "AllCo." Your goal should always be to improve everything, lest you create resentful staff ready to nudge a monkey wrench into your gears of digital transformation.

Marketing and training

After your initial few teams have proven out your new practices and technologies, you'll need to start thinking about training new teams. But it's not just training, it's marketing. When you're scaling change to hundreds of apps and thousands of staff, you'll need to market the change to them, explain the new practices and tools, and also win them over.

At first, you should expect to dedicate at least one full-time person to the role of traveling advocate for your new culture. I've seen this at numerous companies, including BT Group (formerly British Telecom), where the advocate arranged and put on day-long workshops

34 This pattern name can be a bit worrisome for your legacy staff if they think too deeply about it. It's named after the strangler fig that slowly covers and then kills the "legacy" tree underneath the new vines. That said, regardless of the name, the mechanics and intention of the pattern are good and proven out.

on Canvas, BT's brand for their new practices and application platform. As more and more teams transform, you should start doing quarterly, internal conferences. These conferences should have that training and education content from the advocate phase but need to also contain talks from teams following the new culture.

What you're doing here is following the classic marketing playbook: win people over with case studies and the words of their peers instead of the seller, here, management trying to win staff over. For example, one common objection people have is that their app or service is too old and complex to apply a product approach to.

In addition to the pure business benefits, with these types of objects in mind, you should pick some of those difficult applications early on. When you successfully change that project, you can hold it up as an example in your internal marketing and conferences. For example, in telling the story of modernizing their payment system, Air France-KLM's Oya Ünlü Duygulu says, "[f]rom the organization side, there is no more fear of big changes. If such an old application as EPASS can transform, then it's possible for any application."

The word "marketing" probably wasn't on your list of skills when you started climbing the IT department career ladder. But, when it comes to transforming how you do software marketing, it's vital to scaling.

Give Yourself Permission to Change

> *This is the first time that we've been able to take control of our own destiny and actually go after what our customer needs...and making sure that what we're developing is actually going to have some benefit.*
>
> —Erika Green, Director of Technology Product Management, Dick's Sporting Goods

If you take nothing else away from reading this report, hear this: I give you permission to change how you work. This is a cheap mind trick, but it's something you probably need to hear if you're modernizing how your organization operates. The people who you work with likely also need to hear this: "Hey team, you have my permission to change. You might even like it!"

For years we've all known how we need to operate day-to-day and even why we need to change how we manage the software life cycle. And yet, based on the conversations I've had, so many large organi-

zations have yet to change. The people I talk with aren't, you know, *happy*. They know they could be doing a better job; they just can't figure out how to collectively change. As a leader, you'll have to change their mindset, which means you'll have to change yours as well.

Rather than relying on external conditions to force change, wouldn't it be better to control and apply the change yourself? While the heroic responses to pandemic conditions of the past year demonstrated that many organizations could rapidly solve business problems with software, many of the people I spoke with said the pace of change was unsustainable. They were seeing, or could see, their organization getting burned out. If you decide to be proactive with changing how your organization thinks about and produces software, you can escape the burnout that comes from being reactive to external forces.

I've found that when I can't change my habits or start living in a better way, my problem is all in my mindset. Before I can start improving, I have to go get my mind right. More often than not, what I find is that I have to give myself permission to think in a new way. I've gotten stuck thinking that I'm not allowed to behave differently no matter how poorly my current way of thinking is serving me.

For example, during the first part of the pandemic, I was going stir-crazy at home with my family 24/7. I was used to traveling frequently for work and spending long hours sitting at a keyboard writing and reading about, er, "digital transformation." Oh, and also our third child was born. I felt a huge drive to be ever present, both for work and family. One day, seeing me sitting on the edge of the bed, just staring off into nothing, my wife said, "you should figure out getting some alone time." She was right, and I'd been thinking about that every hour of every day. But I didn't think I could give myself permission to be alone, even though I knew I needed it to keep sane and functioning as a parent and partner. Even after she told me this, I *still* had to build up the energy and will to give myself permission to do it. It was a major mindset shift.

While I've never managed an IT organization, from what I can tell it's much easier than holding down a full-time job, administering home schooling for two kids, caring for an infant, and maintaining a long-term relationship, while all at the same time trying to avoid getting coughed on to stay alive. So, if you need it for something as,

er, “easy” as completely changing over how your organization thinks about and manages software, you can have it: you’ve got permission to change. Be sure to tell your teammates too.

About the Author

Michael Coté works at VMware on the advocate team. He focuses on how large organizations are getting better at building and delivering software to help their businesses run better and grow. Most recently, he's published *Digital WTF* on this topic. He's been an industry analyst at RedMonk and 451 Research, worked in corporate strategy and M&A at Dell in software and cloud, and was a programmer for a decade before all that. Coté does several technology podcasts (such as *Software Defined Talk*) and writes frequently on how large organizations struggle and succeed with Agile development and DevOps. He blogs at *cote.io*, and is @cote on Twitter. Texas Forever!