White Paper: **November 2021**

Changing Mindsets: The Missing Ingredient to Accelerating the Digital Business*

Transforming software creation for large enterprises

vmware® | intel

* Excerpted from the report "Changing Mindsets: The Missing Ingredient to Digital Transformation." VMware. Michael Coté. 2021.

Table of contents

Introduction: 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.

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."

While most large organizations aspire to this type of change, few have scaled this new way of thinking to the entire organization. For example, one survey found that 48 percent of executives said they hadn't made improvements to their software portfolios in a year or more. This is despite 82 percent of them agreeing that improving customer experience was directly tied to revenue growth.[1] These lagging organizations think about software as a once-and-done project, not an ongoing stream of work and adaptation to customer needs.

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.

> One of the most actionable tactics people need to take is to think differently, to change their mindset.

This shift in my thinking happened during the height of the COVID-19 pandemic. The way organizations needed to operate changed quickly.

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. The pandemic environment created a whole new set of headwinds that often exposed how far organizations needed to go to modernize their approach to software. For example, pre-pandemic, 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. They had to get new functionality and apps out the door in record pace. Companies put in place new practices, governance and tools that sped up their path to production and improved the design of their applications. The pandemic forced them to shift not only their processes, but

1. Forrester and VMware. "Improving Customer Experience and Revenue Starts with the App Portfolio." March 2020.

their minds; 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.

What's missing for most organizations 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?

The goal: A culture of innovation

Every company aspires to a culture, or 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. 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. Software is an imposing and incomprehensible mechanism that runs the existing state of the business machine, changing rarely and at great expense.

A project mindset isn't good enough: You're always delivering what was needed yesterday and, at best, what you thought you needed today.

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.
- 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.
- Much work can be done in parallel. 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.

The Standish Group has tracked software project success over the past few decades. By my analysis of the past 13 years of their surveys, around 60 to 70 percent of software projects fail by the criteria of feature set, budget and schedule.[2] So, those project dreams work out about 30 percent 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.[3]

When you want to change the way your business functions more frequently—whether by choice or because competition and market headwinds force you to change—a project mindset isn't good enough. A project mindset doesn't adapt to new market dynamics or innovate based on lessons learned.

A product mindset focuses on continually learning what the product is and continually delivering new features as determined by evolving customer and business needs.

Software as a product

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.

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

2. 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: CISQ. "The Cost of Poor Software Quality in the US." 2020.
3. Online Experimentation at Microsoft. 2009.

as determined by evolving customer and business needs. 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.

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. 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.

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 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 month's 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."

With product-based software development, the learning never ends. You're not delivering static value, you're delivering ongoing usefulness, ongoing improvement, and ongoing understanding.

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.

The Small Batch Development Process

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.

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.

Key mindset shift: 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. When you think that some idea or task has "failed," catch yourself and ask if this failure is actually just part of the learning or, less likely, catastrophic failure.

A series of small learnings controls the risk of big, fatal failure.

There are types of failures that are clearly bad—production crashes, security breaches 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.

Management needs to help change people's mindsets to embracing learning, and also to feeling that it's safe to fail. 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.

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. So 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.

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 start with an idea and end up with a user utilizing a new feature. We sometimes call this a "build pipeline," "software supply chain," or a "path to production."

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 such as 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 start-ups 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 an 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 10 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 and groups couldn't coordinate technology decisions.

This indicates 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 "everything."

If heads of departments are sharing negative feedback 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 as CIO 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.

How CIOs can own the end-to-end process

First, pick an important application, such as 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.

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.
- 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" such as 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 such as 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."

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." 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.

Successful organizations change their mindset to act 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, in terms of both culture and software. You'll need to apply the small batch process to learn. This is especially true for learning the truly unique needs of your organization.

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?

First, from a practical standpoint, it's important to build your digital infrastructure upon a solid foundation. It's important to utilize a trusted platform and work with partners who can provide you with the flexibility and scalability to implement change reliably and efficiently.

Second, from a workplace culture perspective, you have to start small. 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 your initial two to 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 first few initiatives, 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, co-workers 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.

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, a "NewCo," 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.

Instead of focusing on 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.

Give yourself permission to change

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 lifecycle. And yet, based on the conversations I've had, so many large organizations 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.

So I'm telling you now: 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. 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!

Item No: 1080490aq-wp-exec-sum-chng-mndsts-uslet 11/21