Platform as a Product

Bryan Finster
and Justin Thomsen

Author
Bryan Finster
Value Stream Architect, Defense Unicorns

Justin Thomsen
Group Technical Product Manager - Developer Experience, John Deere

I was having a conversation recently with Andrew Clay Shafer about common misunderstandings we see around developer experience and the struggles organizations have with developer platforms. I suggested that if we treated platforms like real products, there might be better outcomes. Andrew disagreed. He pointed out an obvious difference between internal tools and products: A good product generates income. The best you can hope for from a platform is that it reduces the cost of building good products. However, to get to that best-case outcome from a platform, we need to operate it with the same mindset that good products have: empathy for the users and their problems.

Improving the flow of software delivery is a complex problem. People, processes, and tools are all part of an integrated system, and ignoring that system view yields poor results. Instead, we must use a holistic approach that impacts everything we do. Dana Finster and I gave a talk at the 2022 DevOps Enterprise Summit titled “The Rise and Fall of DevOps.” Our presentation focused on behaviors we’ve seen that drive success or failure and illustrated that those outcomes could be predicted by how effectively an organization forges a robust DevOps tool chain:

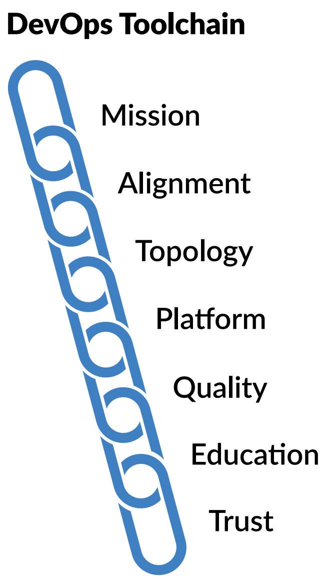

Figure 1: DevOps Toolchain

One critical link in the tool chain is effective platforms that focus on developer experience and enable teams to deliver with less effort. However, many organizations we’ve spoken to struggle with this.

For example, “you build it, you run it” is very often misunderstood to mean that a team should build and operate both their application and the tools used to deliver it. This increases the cognitive load on the team and causes tool fragmentation that makes onboarding and implementing organizational controls exceedingly difficult. Imagine attempting to automate security and compliance controls in dozens of local platforms. Worse, imagine deciding it was too hard and not automating those.

On the other end of the spectrum, we have a “platform as a service ticket” where the “DevOps team” manages the platform and all configurations. There may be a single set of tools or several, but the development teams have no control over their delivery flow and any changes to improve their quality gate require creating a service request for a “DevOps” to fulfill. The extra drag created by that hand off disincentivizes continuous quality improvement behavior. It also requires that the number of people supporting the platform grows as adoption grows.

Neither of these methods helps an organization improve delivery. If we want to leverage the advantages that platform engineering can bring, we need to deliver solutions created by people who understand and empathize with the delivery challenges.

> Our journey at John Deere is very similar to what Bryan describes here. Bryan and I have collaborated on creating our strategy and plan, learning alongside each other to refine our paths forward. Our massive digital transformation has a focus on shifting left, which surfaces hurdles with respect to technical skills and the culture of DevOps. We'll share a bit more about how we overcame some of these hurdles, and which ones we are still working through.
>
> —Justin Thomsen: Group Product Manager—Developer Experience Platform at John Deere

A Journey to Agility

"We are uncovering better ways of developing software…"
—Manifesto for Agile Software Development

In 2015, we began a pilot of continuous delivery at Walmart. We had a large legacy system supported by hundreds of developers deploying to scores of distribution centers in multiple countries. We pushed three or four releases annually, each requiring planned 24/7 support for a couple of weeks and heroic efforts to stabilize. Our leadership challenged us to find a way to deliver every two weeks. We assembled a tiger team of senior engineers, studied the book *Continuous Delivery* by Jez Humble and Dave Farley, and decided daily delivery was a better goal. Why daily? Because smaller batches of change fail in smaller ways and nothing is more effective and unconvering waste and delivery pain.

To achieve this, we couldn't simply tell teams to deliver more frequently. We needed to change everything. Our team structure, organized around feature delivery rather than business capabilities, was continuing to degrade the application architecture. We also had no experience with the continuous integration workflow that is fundamental to continuous delivery (CD). We presented leadership with a new team structure aligned with business domains and a plan to begin rearchitecting for delivery. We created a small platform team to build self-service and opinionated delivery pipelines that aligned with the principles of CD. While the new product teams solved the challenges of continuous integration (CI) and CD, they provided feedback to the platform team to improve the pipeline defaults. This allowed the teams to focus on their domain capabilities rather than how to deliver them while we embedded the good patterns we learned into the platform to help future teams. Solving all of the problems of delivering nonbreaking changes several times a day rapidly grew engineering skills on those teams. It also had the unexpected benefit of improving morale. Teams love seeing their work used, and those pilot teams could see that several times per week rather than three times a year. We had improved mean time to dopamine.

CD: The Lever, Not the Goal

Our experience showed that focusing on solving the problem of "why can't we deliver working solutions daily" was the most effective method for driving improvement. We implemented a strategy to use CD as the primary method to scale the same engineering and business improvements across the enterprise. The challenge—and the strength—of this strategy is that CD is more than tooling. It also requires a specific workflow and mindset to deliver optimum results. One way of doing this is to create a large coaching organization and perform training with every team. However, even if enough qualified people could be found to work with teams, that method creates unsustainable overhead for any large organization. We needed to apply force-multiplying solutions to help teams self-improve. To that end, we created a centralized delivery platform organization to lead that strategy.

Centralized Platform

Standardizing tools is a double-edged sword that can result in poor outcomes if done incorrectly. If we impose too many restrictions or opinions on the platform, we can strangle innovation or force people to work around the platform to get the work done. Done correctly, we can generate economies of scale. Having one set of tools for everyone has obvious benefits. Using fewer tools means lower operations costs and less integration effort. It also means onboarding or changing between teams is easier. Further, it allows us to automate standards, policies, and security into a single platform and incentivize change.

> "The central idea is to create an environment where doing the right thing is as easy as possible. Much of the battle of building better habits comes down to finding ways to reduce the friction associated with our good habits and increase the friction associated with our bad ones."
>
> —James Clear, Atomic Habits*

Empathy, Not Mandates

When delivering any solution, the last thing we want to do is alienate our potential users. The most effective way to alienate internal users is to force them to change. Even if our solution is better, forcing them to switch would burn any goodwill we may have received from voluntary adoption. That goodwill is very important when we stumble early on, as every new solution does. Instead, we wanted to build solutions that enticed them.

* James Clear, *Atomic Habits* (New York: Avery, 2018) 147.

We knew that a global platform was the enterprise's goal and that using it would be mandated in the future. However, if we acted that way, we would drive away users with both poor interactions and poor solutions. We wanted adoption to be a pull, not a push. To that end, we focused on behaving with empathy for the problems our users had. That empathy and user-centric focus didn't end with developers. An improved developer experience does not mean we optimize only to reduce the time or effort required to deliver code to production. It means we make it harder for teams to make errors and easier for those teams to operate their products. Doing this means working with all of the other disciplines that surround coding and helping to make their jobs easier as well. We collaborated with the Security and Compliance areas to embed their concerns into the platform so that every change could be validated against their policies automatically. This took disciplined change management on our part. You cannot apply strict automated validation to applications that only used manual security and compliance processes in the past. We'll cover this more later. The result of this work was that teams were not required to use our solution. However, they would need to work very hard to meet the organization's nonnegotiables with another delivery solution. With ours, it just happened.

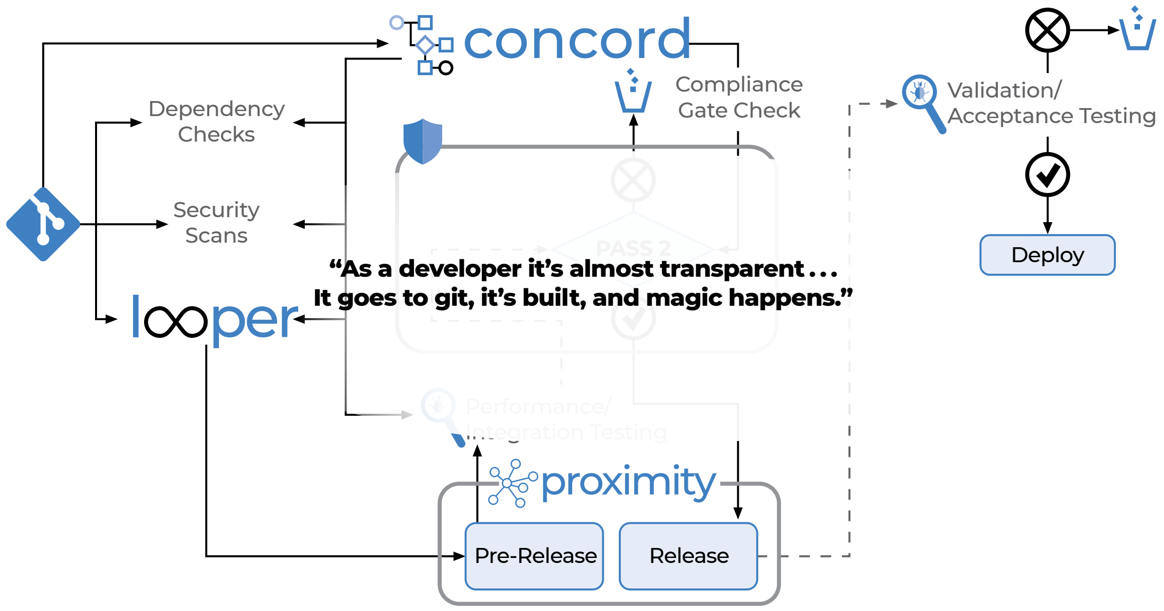
concord
Compliance Gate Check
Validation/ Acceptance Testing
Deploy
Dependency Checks
Security Scans
looper
"As a developer it's almost transparent… It goes to git, it's built, and magic happens."
PASS 2
Performance/ Integration Testing
proximity
Pre-Release
Release

Figure 2: Scaling CD to Walmart

As we established the developer experience product family, we knew we did not want our emphasis to be on enforcing tools. We wanted to treat the developer experience platform as a product, not a means to tell developers how to deliver software. We chose to follow Bryan's pattern of creating opinions and making those the easiest way to do the right thing. We created "as a service" products for CI/CD, observability, API integrations, cloud, and containers. The platform opinions are formed around things such as trunk-based development, ephemeral workloads, loosely coupled architecture, zero trust security, and continuous delivery.

—Justin Thomsen: Group Product Manager—Developer Experience Platform at John Deere

Scope and Organization

Another problem to consider is how to avoid solving the wrong problems. As we discussed earlier, internal platforms do not generate income. Our value proposition is lowering costs elsewhere for other value streams. For a platform with enterprise funding, it's easy to fall into the trap of "That would be neat. Let's solve that problem too!" In the process, we can become top-heavy with features that solve phantom problems or only help a small fraction of the organization. "Wouldn't it be cool if we had a developer portal with drag-and-drop service creation?" It might, but is there a need? If it does not dramatically lower the cost of development for a majority of teams, we have only created an ongoing expense for something that looks good on a conference stage. In this way, we can quickly eliminate our value proposition and spend ourselves out of existence. We need clarity of mission and vision.

We had a clear mission: help drive the organization's CD strategy. We also had a clear vision: irresistible developer experience. Next, we needed to define our scope and organize for success.

Software delivery enablement (SDE) was part of a larger infrastructure organization. Our scope of responsibility was all of the capabilities from version control to delivery. We would interface with, but not be responsible for, capabilities such as operational observability and workflow management solutions. With a known scope, we leveraged domain-driven design to quickly deliver our goals while remaining flexible about how we implemented them. We defined the discrete capabilities, version control, CI, security, artifact versioning, delivery metrics, etc., and organized cross-functional teams around each.

Each team's product owner was responsible for aligning the team's road map to the overall platform goals. Teams were trusted to deliver the capabilities they were responsible for and held accountable for the outcomes, including operational stability and user experience. By giving this trust, platform leadership established a culture of ownership that fostered innovation and improved user interactions with each product within the delivery platform.

By organizing around capability domains with the goal of being able to replace underlying infrastructure without impacting the user experience, we were not locked into early technology choices. We could, and did, change the tools used to deliver these capabilities.

An Irresistible Developer Experience

Because one of our goals was to spread knowledge of continuous delivery workflows, the platform made practices like trunk-based development the easiest way to work and provided feedback in the form of scores for how well CD was being executed. It also made other workflows more difficult, either intentionally or by simply not explicitly supporting them. Using practices such as GitFlow, which were incompatible with our goals, resulted in increased toil. However, if we eliminated the ability to use other workflows entirely, it would prevent adoption for any team that was not already on the path to continuous delivery. We took an approach that allowed teams to trade simplicity for flexibility as needed while also ensuring that security and compliance could not be avoided. Our abstraction layer over the tools also allowed us to maintain a consistent user experience while changing tools as our needs changed.

To be effective, we needed to avoid the platform-as-a-service-ticket antipattern, so we prioritized the ability for teams to configure their delivery flows according to the needs of their products. We provided extensible templates that allowed teams to use simple, declarative commands to configure common tasks such as test coverage reporting. The base templates exposed those simple commands while also enforcing data collection, security scans, compliance validation, and business rules. For example, we could create and enforce rules to block unapproved changes during major sales events. However, if a team needed more complex behaviors for their specific application, they still had the ability to create scripts for those behaviors since not all twenty-year-old code is architected for continuous delivery patterns. We were also responsive to input from teams using common technologies, but not yet supported. As an open-source first organization, we encouraged inner sourcing. We would happily accept code submissions to improve our support for those technologies as long as they with the overall mission.

> At the beginning of our journey, we had very tool-focused, siloed products before shifting to the opinion-based model Bryan talks about. This shift allowed us to create boundaries around what lives within our scope and what does not. Connecting our various silos together is an ongoing challenge that we are tackling through a combination of a holistic paved path in a one-click cloud-native platform and individual building blocks of that path for teams who have already begun their continuous delivery journey. This approach makes it easy to do the right thing while allowing flexibility for teams to do things on their own if they choose to forge their own path.
>
> —Justin Thomsen: Group Product Manager—Developer Experience Platform at John Deere

Speed with Safety Nets

Focusing on a good developer experience does not mean we ignore our responsibilities to keep the enterprise safe. It means we make keeping the enterprise safe as easy as possible by embedding security and policies into the platform. However, if the teams' previous delivery solutions relied on manual security verification, it's exceedingly unlikely they met our desired security profile. If we simply blocked the delivery of their current applications until they improved their security, then our platform would never be adopted for anything other than greenfield development. We needed to enable them to keep the business running while adopting our platform, but also increase our security profile and automate compliance.

"Poka-yoke translates as 'error proofing' or 'avoiding error.' One of the techniques of poka-yoke is to add simple fixtures, jigs, or devices to constrain the operations so that they are correct."

—Don Norman, *The Design of Everyday Things*[*]

To make it easy for teams to deliver secure solutions, we needed to make it continuously more difficult to be insecure. We also needed to do it with empathy for the delivery goals of the teams. The easy thing would be to simply implement the final standards and tell the teams to pick up the pieces, as most of their pipelines went red. "Why didn't you follow the standards before? Suck it up."

To keep the enterprise safe, serve the needs of the business, and help the teams, there needs to be a more reasonable approach. Whenever we do anything that may cause disruption or add friction to their delivery flow, we need to over communicate. Before implementing a new security or compliance gate, we broadcasted on Slack, email, and anywhere else we knew teams might see the information that a new mandatory gate was going to be implemented. After a month or so of broadcasting, we implemented the new pipeline gate as a warning message in their pipelines that included information about when the warning would be switched to an error that would halt their pipelines. Only after that did we block non-compliant builds. Even then, we worked with areas with legacy applications that could not migrate fast enough to provide an exception while helping them come into compliance. In this way, we continuously increased the security and compliance profile of every system that used our platform.

Hundreds of delivery teams accelerating change into the system brings concern that teams may be missing security, compliance, and audit requirements. In 2023, we are focused on creating a minimum viable commit to production workflow. This aligns to both what Bryan mentions and what we heard from many others at DOES 2022. Governance as a service allows teams to continue delivering more often and ensures they are doing it in a safe way. The service will cover requirements for compliance, security, and audit while helping to drive cloud cost savings.

—Justin Thomsen: Group Product Manager—Developer Experience Platform at John Deere

Making Metrics Matter

Our explicit goal was to transform how the enterprise worked. There was a push from the CTO for that change where he challenged every team to solve the problem of delivering to production daily. This was not about the speed of delivery—it was a challenge to solve the engineering and communication problems that prevented daily delivery to improve the entire organization. However, you cannot improve if you do not know where you are and how to get to where you're going. Metrics reporting was an important part of our platform strategy and one of the earliest capabilities we delivered.

The delivery metrics views had two important uses. First, it gave teams visibility into the status of their pipelines. A single view of pipeline health is crucial to a team's productivity because a broken pipe-

[*] Don Norman, *The Design of Everyday Things* (New York: Basic Books, 2013) 233.

line cannot provide quality feedback, so fixing a broken pipeline is the highest priority for a team. Second, the metrics showed the teams how well they executed CD behaviors relative to what "good" looked like. For example, if you as a developer used trunk-based development and integrated code at least daily on average, then you got five stars for source management. We continuously reviewed the behaviors the scoring caused to guard against perverse incentives. We would show a balanced set of metrics so that when gaming the metrics occurred, they were gamed in favor of our goals. When we saw undesired outcomes, we would adjust how we displayed the information or, in some cases, use education to help prevent management from using the scores to compare or blame teams.

We would frequently receive requests to adjust the scoring to provide good scores for things like delivering once per month because "our users don't want changes more frequently than that." Since this didn't align with the organization's improvement goals, we referred them to other resources we offered to help them migrate to CD. More on this later.

Building the "Easy Button"

Here is my source code
Run it in the cloud for me
I do not care how
—Onsi Falhouri, "Pivotal Cloud Foundry Overview"[*]

Our next task was to implement a multi-tenant cloud solution that would let us deploy containerized workloads to the public cloud, our private cloud, and the data centers that reside in every store and distribution center. We wanted a solution where the teams did not need to know or care where their applications were running, and we wanted it to be even easier to use. After all, platform engineering is about delivering products to development teams that allow them to focus on the problems they are trying to solve instead of solving the problems of how to deliver the solution.

In 2019, we released the Walmart Cloud Native Platform (WCNP). Shifting from deploying onto virtual machines to deploying containers in Kubernetes would usually require every team to slow down and learn new technologies and make costly mistakes along the way. We wanted to make that transition easier. With WCNP, if you wanted to deploy a React application, simply tell WCNP to deliver a React application owned by your team and where to send alerts with ChatOps. WCNP would handle everything else. It configured the container, logging, monitoring, and alerting, and assigned a domain name. From then on, all interaction could be handled with ChatOps, including approving delivery to production, if not configured for continuous deployment.

Teams did not need to learn how to build efficient containers or anything about Kubernetes to deliver solutions. The platform hid that complexity from them. If they needed more complex pipelines,

[*] Onsi Fakhouri, "Pivotal Cloud Foundry Overview," 2016, https://youtu.be/7APZD0me1nU.

they had the ability to build their own containers, and WCNP would handle just the delivery and operational monitoring setup. Again, this gave teams the option to handle more complexity if needed. Feedback from product teams was overwhelmingly positive. The developer experience also enabled teams to run quick, disposable experiments with almost no effort. Lowering the cost of change is critical to innovation.

The Power of Branding and Culture

Building a brand is important to building awareness that helps increase adoption. People respond to branding and logos. If we want to spread the word that we have a better solution, then naming something "The Continuous Integration Server" looks like we aren't seriously invested in making it better for them. We spent time building brands. Every product within the platform had a brand and logo. Each had a distinct mission statement that mirrored Walmart's mission.

Figure 3: Example Brand logo
Source: https://concord.walmartlabs.com/index.html

Not only did this drive external awareness, but it also increased morale on the product teams. It gave every team an identity that increased mission awareness and pride in their products. Stickers have power. Ours could be found on the laptops of some top developers, and our brand was recognized.

It's not enough to talk the talk; we also tried to walk the walk daily. Continuous delivery is more than tools; it's a culture. It requires a commitment to trust and sharing to improve delivery performance, and we tried to exemplify that culture. For example, if we had an incident that affected users, we didn't hide this or make excuses. We would broadcast on Slack that there was a problem and give frequent updates on the status. After the incident was resolved, the team or teams who owned the issues would gather and write a postmortem. Those would be published publicly and announced just as loudly as the issue announcement. "This is what we learned. This is what we are improving to make the product better." We also used the tools we were building to help build the tools. For example, Concord was used to deploy changes to Concord.

Taking Documentation Seriously

Supplying tools that reduce the complexity of delivery for our users is not enough. Good documentation not only improves the user experience but also lowers the cost of daily operations by reducing the level of one-on-one support. We took a disciplined approach to this by creating a team to curate our docs and create self-service training material for the platform. The word "team" may be overstating things somewhat. For the first year or so, the team consisted of one person, a skilled engineer who was also skilled at creating effective training material. Later, the team doubled in size to two. The team would record live classes on how to use the new tools while emphasizing CD behaviors. New classes would be held to update the recorded training as new features were released. The team also curated the platform's documentation website. However, they were not responsible for creating the documentation. Each product team was responsible for submitting updates and improvements as the platform matured. The documentation and training team would review for style and content and accept the change if it met their standards.

Even if documentation isn't ignored, one of the common problems is maintaining it. Often it is written once and then quickly becomes obsolete or little effort is spent on making things easy to find. Either scenario increases operational costs as the volume of hands-on user support needed to compensate grows with adoption. We intended to run as lean as possible and to protect our time for growing platform capabilities, so training and documentation were our first line of defense for user support.

Product Support

When operating any product, there will be a need for user support. Especially for internal platforms that are funded by the money they help other teams save, keeping operational costs low is essential. As one of our users, if you asked for help, the flow worked this way:

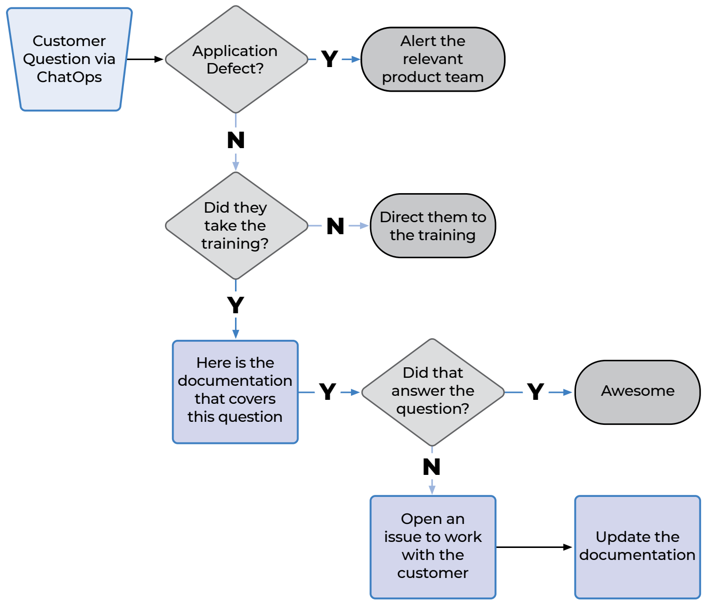
Customer Question via ChatOps
Application Defect?
Y
Alert the relevant product team
N
Did they take the training?
N
Direct them to the training
Y
Here is the documentation that covers this question
Y
Did that answer the question?
Y
Awesome
N
Open an issue to work with the customer
Update the documentation

Figure 4: Product Support Work Flow Example

The point of entry for help with any of the platform tools was not a request to open a ticket. That results in a terrible user experience. Instead, the entry point was through ChatOps. Questions were asked and answered in real-time and publicly whenever possible. This not only improved the developer experience but also helped others in the channel who did not ask the question. By focusing on making the documentation discoverable and using it as the primary way to answer questions, we were continuously testing its accuracy and improving it. By 2020, the platform supported delivery pipelines for the vast majority of approximately 1,500 development teams, or about 18,500 developers. Globally, the platform support team consisted of eight people divided between the US and India to maintain twenty-four-hour coverage for the platform. Sometimes the support questions would be about continuous delivery methods of working. In those cases, the platform support would respond, "Have you spoken to the Dojo team?"

Platform as an Agent of Improvement

Gamification of delivery metrics was one of the first capabilities we built. As you may imagine, adding gamification sparked many conversations with teams as they adopted the platform. The goal was not to hold teams accountable but to help teams understand the difference between how they were working and the workflow we were helping to achieve. As the number of requests for "How can we improve our scores?" increased, we organized a team dedicated to helping them learn continuous delivery.

In 2015, Target Stores* announced their DevOps Dojo and the impact it was having on their ability to improve delivery. This was quickly followed by Capital One, Verizon, 3M, and many other Fortune 100 companies creating their own Dojos. The Dojos used immersive learning techniques, including a strategy of "let's use your work to learn how to do your work better," emphasizing shrinking feedback loops to uncover pain.

Because of the mission we were on, our Dojo was not focused on general improvement. Ours was targeted at helping developers solve the problem of daily production delivery in their context. We did not start with agile process coaching, we started with engineering. We built a team of senior developers with proven technical leadership ability and experience executing daily delivery. We had gone on the journey on our previous teams, knew where the pitfalls were, and knew how to guide to better outcomes. We started in 2018 with two people. By 2019, the team had reached its maximum size of five. Rather than waiting for a large space to bring teams to, which would never happen in our environment, we focused on things that multiplied our impact.

We did the math: if we could find fifty qualified people to work directly with teams, it would take us eleven years to help every team. We had to find other ways. While we did many six-week engagements

* Heather Mickman & Ross Clanton, "(Re)building an Engineering Culture: DevOps at Target," Presentation at DevOps Enterprise Summit, 2015. https://www.youtube.com/watch?v=7s-VbB1fG5o.

pairing directly with a team to help them, our goal was to use the knowledge we gained by helping them to document and share the common problems teams had. As with the other product teams, we leveraged our documentation. We created easy-to-consume playbooks that covered every challenge we saw teams struggle with: work decomposition, testing patterns, application architecture, team architecture, CI workflows, etc. We began working on a new metrics view that would leverage pipeline delivery metrics to automatically suggest playbooks to help improvement, which we called "coaching as a service." We also put a plan in place to train other leads across the enterprise on how to do what we were doing. Because of its close alignment with how we were already working, we began leveraging Gary Gruver's "Engineering the Digital Transformation" certification program to train EDT White Belts and then work on cohorts of Green Belts to scale our methods horizontally rather than trying to grow our organization. Our mission was to make our team obsolete so we could return to product development.

We not only helped teams learn how to use the platform effectively but also acted as eyes and ears for the area. We asked questions like "What challenges are teams having?" and "What problems could we solve better, or are we not solving yet?" We also took the stance that we didn't care if you were using our platform or not. The enterprise goal was to drive CD, and we would help in any way we could, including helping to remove roadblocks that were outside of a team's control and had nothing to do with the tools they were using. By 2021, the dojo was an effective product for scaling team improvement and a trusted source of information for better software delivery.

John Deere's transformation business unit and dojo experience created a solid foundation for the success of their agile operating model. As we continue our digital transformation, we can build upon the gains of agile and scrum to drive even further into DevOps and grow our developer experience platform products. To do this, we realized that to continue to challenge and change our culture, we have to help teams adopt the platforms we build. In late 2022, we expanded our developer experience product family to include developer advocacy. This new area is focused on marketing the developer experience platform products, providing centralized easy-to-use documentation, improving the end-to-end developer experience, curating learning paths to grow skills for the platform, and encouraging frequent feedback loops and experimentation.

—Justin Thomsen: Group Product Manager—Developer Experience Platform at John Deere

Outcomes

When the Software Delivery Enablement organization began, there were multiple version control systems and technologies. There were dozens of competing tools for building and delivering applications. The MVP of the platform only handled a couple of simple delivery problems. We used a strategy of expand-and-contract to move the mission forward. We assumed the operational responsibility (and costs) for their existing tools, built the capabilities we were missing to support their needs, helped them migrate, and shut down their old systems. If some teams were reluctant to move after we had migrated most of the area or were not given the time to work with us, we offered to give the operational costs back to their vice president.

Fast forward four years, and over 80% of the enterprise's delivery ran through SDE's platform. Our tools delivered nearly 10,000 applications to every Walmart environment. With thousands of builds and thousands of teams being supported around the clock, SDE still operated with an eight-person, tier-one support team distributed for "follow the sun" operations and a total staff of around eighty people spread across the small product teams that made up the platform capabilities. Because most support happened in public on our Slack channels, often questions would be answered by our power users, further lightening the load on our teams. We saw this behavior frequently on the Dojo channel, where questions about good CD workflows would often be answered by engineers in other areas before one of us had the chance to respond. We always thanked them. Sharing is one of the most important cultural practices for improving delivery, after all.

Be the Platform You'd Want to Use!

Every internal platform is a product; some are just not very good. Many are a buffet of tools with a fragmented experience and no clear goals that provide no safety nets and require extensive knowledge to use correctly. Others operate with the mindset, "These are the approved tools, and you will use them."

> "…a lot of the platforms I come across right now are still in a state of focusing on the SysAdEx… or the SAdEx as I like to call it."
>
> —Bryon Kroger[*]

Bryon isn't wrong. Platforms optimized for SADEx to make the lives of the people operating the platforms easier will reduce the effectiveness of the teams using them. Starting with a user-centric mindset is critical. However, since many platform organizations grow from infrastructure organizations. It's uncommon for them to be trained in that focus. If we want to be a useful platform, we must learn that mindset.

Start with the mission. One of my favorite missions is, "People have problems. We should help make their lives less problematic." Next, set your sights on a vision and goals that align with your mission: "Provide an irresistible developer experience to our users to make it easier for them to deliver daily value. Make security and policy so easy to implement that they are features, not impediments."

If we start with the right mindset and take product management seriously, we can make a positive impact on the lives of our users, their customers, and our organization's success. All of these take investment, but the cost of doing them is far less than the cost of ignoring the need.

* Bryon Kroger, "I'm reminded of Onsi Fakhouri's haiku…", LinkedIn comment, https://bit.ly/SADEx.