VMware Tanzu®

Voices of the Vanguards:

Our Guide to Measuring Software Delivery Through Metrics

vmware®

Background

In September 2022, Michael Coté, Senior Technical Staff, VMware Tanzu, led a live webinar discussion with three VMware Tanzu Vanguard customers to get multiple perspectives on the topic of Measuring Software Delivery Through Metrics. This guide is a recap of the panelists' thoughts, experiences, challenges, ideas, and advice.

Introduction

Michael Coté: There have been lots of interesting conversations and thoughts about measurement metrics over the past 10 or so years, especially in the DevOps space. This discussion is more about software delivery and the full process, all that's involved in actually enjoying (or worrying about) building your software, discovering what's there, and deploying it and running it—as well how you are thinking about, reporting on, and improving that process over time.

We will review the popular metrics, performance metrics for checking on the health of your applications and the way they're performing, as well as the kinds of metrics that you use to track how an application is running or items like number of defects.

Whether you're building the software that you use to run your organization — internally, or for a customer or partner — you're thinking about the following:

- What is the state of the software running in production?
- What is the state of the "factory" and the process that is creating this software?
- What is our operational health as far as the potential we have in the future to keep operating?
- How can we make sure that we're getting better?

This report is divided up into four sections:

Industry-Leading Metrics: DORA, Flow, Value Stream

Tech Debt Metrics

The Human Side of Measurement

The (im)possible task of measuring customer experience

01

Industry-Leading Metrics: DORA, Flow, Value Stream

First, we'll compare and contrast some more popular, well-known metrics.

The first group of metrics at the top of this chart is from the State of DevOps reports.

Metric	Description	Examples
DORA	DevOps Research and Assessment metrics	• Deployment frequency (DF) • Mean lead time (MLT) for changes • Mean-time-to recovery (MTTR) • Change failure rate (CFR) • Cycle time from idea to production
Flow Metrics	Measure the flow of business value through all the activities involved in producing business value through a software value stream.	• Flow velocity • Flow time • Flow efficiency • Flow load • Flow distribution
Value Stream	Value stream mapping typically uses 4 core metrics to analyze the flow of work through each stage in a value stream	• Lead Time (LT) – Time to complete the stage from intake to handoff • Process Time (PT) • Complete & Accurate (%C/A) • Value Added (VA)

DORA Metrics

DORA metrics were codified in the Accelerate book by Nicole Forsgren, Jez Humble, and Gene Kim that came out in 2018. Some of these metrics include:

- The frequency that you deploy software
- The number of failures, defects, or errors that are in each deploy (do you have to roll it back?)
- How long does it take the team from an idea to actually getting something to production?

Each of these metrics are getting to the bigger question: what is our ability to deploy software? And when we deploy it, are there errors in it? Then what is the mean time to repair when something does go wrong? How quickly can we fix it?

These DORA metrics are good metrics to start with, a baseline when you're looking to improve how you're building your software and delivering it. They are going to give you an indication of the health of your software delivery. There are two main reasons why it's important to deliver software frequently:

- It allows you to modify your software, add new features to it, in an iterative way. This means you can experiment with new ideas to see what works. Instead of waiting 6-12 months, you can more quickly change it and make it better. To me, this is key to how you use software to innovate your business.
- But also, there's an interesting side effect, where if you're releasing very frequently, you're forced to only release a small amount of software, which in theory makes it easy to repair when things go wrong, because there's less stuff to mess with.

Flow Metrics

The next set of metrics are flow metrics, which come mainly from Tasktop CTO, Mik Kersten (and friends), bestselling author of Project To Product: How to Survive and Thrive in the Age of Digital Disruption with the Flow Framework.®

Flow metrics are focused on the health of your factory. How many units of work, and which units of work, are you getting through the pipeline? A huge amount of this is borrowed from lean manufacturing. So the metaphor is always that we have a factory line or a process that is assembling things, delivering them, shooting them outside the factory doors (deploying your software), and getting it out there for people to use.

Borrowing from lean is this idea that there are many different activities. But for the purposes of this, there are two activities you're looking at here: one, actual work, like whether it's coding or working on your applications to get it out the door, that configuration and things like that. And then two, there's wait time: that time you're waiting to have a meeting with someone, or more everyday things like waiting for someone to come back from lunch or vacation. Efficiency is an interesting metric, because it tells you how you're doing as far as eliminating wait time or waste.

Value Stream Metrics

Value stream metrics emphasize the complete end to end process of getting software out the door. When someone has an idea for a new feature, your product team is thinking about how they would build it, discussing what it is, how they would solve that problem, coding it and testing it, packaging it, verifying it, deploying it, maybe staging it, doing all that sort of stuff to then finally deploying to production. Then they have people that use that software and observe what they're doing. But this whole process, end to end is really what a lot of these value stream metrics are looking at.

I'm not suggesting that you should start using all of these metrics right away. But chances are, that if you're looking to improve the way that you're deploying software, you'll be using a fair amount of them, or versions of them that you have customized. And if you understand what these metrics are, and why they would be valuable, it gives you one way of understanding how you can improve your software process: not only things to measure, but also what "good" looks like.

02

Tech Debt Metrics & Other Panelist Experiences

Jonathan Regehr: One of the huge reasons that Garmin started down the journey with Tanzu Application Service and adopted that platform, was because we knew our deployment frequency was way too low. We knew we needed to bring it up, and we wanted to track that.

And we found that this was a lot more difficult to track than we wanted. I think one of our audience members also asked, "how do you track the source data residing in multiple places?" That's actually one of our problems. We had some teams that were making tickets when they would release and they would detail all the things they did. And other teams weren't doing that. And we're following a little bit more of an agile flow. And so it is very difficult to track deployment frequency, especially if you think of the agile methodology where to truly be agile, your process is constantly changing. As that process changes, you have to constantly figure out how you're going to track what a deployment is and what is in a deployment. There are challenges to doing all that, and it does make deployment frequency difficult to track. I think at some point, we sort of decided we were just going to look at how many times we called the CF push [the command developers use to deploy their applications to the Tanzu Application Service, which is based on Cloud Foundry].

Michael: Several years ago, I had someone ask me a similar question and they were a Tanzu Application Service user. To agree with you, as they dug deeper trying to find release frequencies, they encountered exactly what you're saying: "Well, how do we define a release? Like, there's so many things going

on?" It's difficult, especially across teams, to say, "this is a release versus this is a patch." So definitely, you have to know what your terminology means, what the items are. Paul, do you have any reflections on these metrics?

Paul Pelafas: Yeah, I do. At the previous role that I had, the company had recently been through a digital transformation. So this is moving from a waterfall world to more agile methodologies and setting up some product teams that were centered around very specific domains. So as you can imagine, this was a huge investment for the company. We were wanting to know:

- Were we getting a return on investment?
- Making things better? Worse?

Metrics were very much at the center of trying to understand those questions. In our case, our product teams were often the guinea pigs for the new way of working in the organization. So seeing how quickly and how efficiently these teams could work with some of these new tools, and just workflows and processes was really important.

We focused very much on DORA metrics, and how quickly we could move from ideation into production and see real value from the code that our engineers were producing. We could also see how quickly we would respond to different issues that arise because in software, there's a human element, and there are always going to be defects or bugs. How quickly we can respond to those and make changes that have a positive impact to our consumers is really, really important. We focused very much on the DORA metrics, and these were our guiding light for our product teams.

Jonathan: You mentioned cycle time, and that to me that feels a lot like the deployment frequency. You also mentioned release size and that was one of our big things. We had these huge releases, long cycle times, and then you hit production with those things. And let's say something goes wrong. Everyone all of a sudden says, "I've got to figure out what the problem is." It's very difficult to find issues in that regard, because your release is so big that many different developers have code running in there.

Maybe something got lost in a merge somewhere and so it made bugs a lot more difficult to squash. So I put a high value on the short cycle time. Short cycle times benefit the customer, but they benefit your code quality as well. Because you're that much closer to when the code was written. And therefore, you're that much closer to remembering what the problem might be.

Paul: Yes, that's a great point, Jonathan.

> "Our organization went from quarterly releases to multiple times daily. You can't get there overnight. So measuring how much more efficient you are over a period of time, you can see trends, numbers, and data to help influence where you're going."
>
> Paul Pelafas

Our original aspirational goal for teams was, "Can you deploy once a week?" and then tracking that against all the teams. Then it was moving to "can you deploy once a day?" And after that it's multiple times per day, because you're right, the smaller release sizes make it much easier to fix things as they come up.

> "76% of executives said they are too invested in legacy applications to change."
>
> Source: "Improving Customer Experience And Revenue Starts With The App Portfolio,"
>
> Forrester Consulting, commissioned by VMware, March, 2020. Survey conducted July to Oct. 2019 with 614 respondents and six CIO/SVP interviews.

Michael: Legacy software is an area that I've been spending a lot of time on in the past year. Your existing software and legacy software is often the software that's mission critical, that makes you all the money, that's vital. And the definition I use, the jokey one about legacy software is it's software that you have to change, but you're really afraid to change, right? Because if you weren't afraid of changing it, you would just call it software, you wouldn't put the word legacy in front of it.

I think, especially in large organizations, the more successful ones that have managed to stay alive for a while, there's always technical debt and legacy software to deal with. And it's one of the more elusive things to measure, right? Jim, you've been thinking about how to get a sense of the tech debt that you have. How have you been working on this?

Jim: Basically, whether you've got a legacy system (or even a new system that is going to become legacy very quickly), it's going to accumulate debt. So tech debt is sort of euphemistic for a lot of things. And people say, "well, that's tech debt, what are we going to do about it"? and shrug their shoulders.

But let's talk a little bit about what it really seems to be, at least from our world and what we've done with it. And when we got on the Tanzu Application Service train (and it's great, it's fantastic), we saw the things that everybody talked about, that everybody was jumping into our system. And they were basically bringing all kinds of things to the party, big apps that needed refactoring apps that really weren't compliant with what we wanted. So there are lots of areas where tech debt can accumulate because they're not norming around the things we wanted to initially.

As we evolved, we got our cycle time down; it was great. But we needed to take care of some things like changing platforms, or changing language versions, compilers, non conformance projects for architecture, configuration, and security vulnerabilities. These are always going to keep coming at us. So we actually did something about it. But to me, that's what technical debt is.

> "Why is it important to be discussing these productivity metrics? Because they're hidden. They're very hard to see, so it affects everything that we talk about. So when are they addressed?"
>
> Jim Kohl

Some teams, they'll deal with these problems on a quarterly basis. And they've got to justify why they're taking these things on. But for other things, we work on a weekly cadence, and then we address it right away. But what we've done is what maybe some organizations do — we look through all the projects that we have on our portfolio. And we have software that looks at the configuration and flags issues that we find that are problems with the configuration. Or if we find vulnerabilities we take feeds in from our Checkmarx partner, or if we have any other sources that give us information like Tanzu Application Service, we can look there. We take all this data, and we actually create issues on the team's tracking boards. There are quantitative things that can be measured, so that's something we've done. Some teams, I imagine, never capture it, and they keep shrugging their shoulders. But it's less of a hidden cost to us.

We actually look at this other work we're doing and we ask teams to commit time, because they're high priority and medium and low priority issues. The high priority issues, we want them to resolve very quickly. There's also this notion of a technical debt ratio. We have not evolved to this extent, but it's remediation costs versus your development cost times 100%. It's a little trickier to measure. And I'm sure, Coté, that in your Legacy Trap eBook, you probably could talk a lot about that. The main point here is that it's something that is often ignored or not accounted for. It's important to think about how that factors into those metrics.

Michael: Another thing that I've encountered over the years that I've been able to write up with some of my coworkers recently is this idea of developer toil. It's, as the name would imply, annoying stuff developers have to put up with. There's a way of pulling metrics out by using surveys, essentially, which I think is an interesting practice.

03

The Human Side of Measurement

- Teamwork vs. Individual blame
- MTTR/MTTI example

Michael: When we were discussing this, something that came up very frequently was that you will have good and bad metrics, and hopefully, you'll figure out which ones are working well.

> "But especially as you're programming, building your art, your organization, you're creating the architecture for your factory. It becomes very important to pay attention to how these metrics are affecting people, humans - both good and bad (and especially the bad)."
>
> Michael Coté

There are good ways that metrics can affect people, and drive them, and y'all have a bit to say about that. There are bad ways, as well. Jonathan, why don't you start us off with the idea of "blame-storming," as we used to call it. The blame game is something that metrics are, unfortunately, a ready weapon to use in any sort of blame battles that you have, which one has to be cautious about.

Jonathan: Indeed, yeah, I've got a lot to say on this topic. I'll try and keep it brief. But I want to focus for a minute on the MTTR versus MTTI. So MTTR is a mean time to resolution: when you have an issue and you want to put your systems back to normal as quickly as possible. We'll see how fast you can do it — that's mean time to resolution. That's a great measure. That measure should constantly be getting faster, as your teams get better at solving issues.

But there was a period of time where mean time to innocence became something that teams tracked. And that's the whole thing of each team frantically looking at their own system going, "Oh, it's not me, my system is fine." And leaving the room or going back to coding. But mean time to innocence — I don't see any value in that. Because essentially, you're slowly shrinking the number of minds that are solving the problem. With really good metrics, and really good monitoring and alerting, I'll plug Aria Operations here. If you can put your metrics together really well, everybody can see at the same time where the problem is.

If you all can look at the same metrics and all see very quickly that the problem is a storage problem, or a network problem, or whatever it happens to be. Instead of all of the people hounding that one team, now everybody can collaborate together and solve that problem at the same time.

The blame topic is an interesting one, too.

Jim Kohl: Well, one of the things that I'm concerned about is that we, as engineers, like to optimize things. So it's a very interesting proposition to work on these things. However, we have an inherent distrust of management, and how these metrics will be used against us.

> "If you want to have fail fast teams, you have to make sure that even if someone makes a mistake, that mistake is handled in a way that your (human) teams are comfortable failing. If they're not comfortable failing, because someone failed and the director walked out of their office and went "who did it?" your teams are going to be a lot more cautious. You may have measurements and you can tell who's doing what, but nobody's taking any risks — and therefore your company is going to move a lot slower."
>
> Jonathan Regehr

Paul: Yeah, that's so true, Jim…there has to be a trust element there, otherwise, a lot of this stuff falls down. I know a big thing in applications and software development is gamification, so we did a lot of this pitting our teams against each other to see who could deploy most frequently. That only works if there is trust from the development teams to the stakeholders and understanding that there are going to be things that happen and if we can respond to them quickly and not cause major disruptions in the business, then that's okay. And that's to be expected. I don't see how this works unless you have that buy-in because, whoever mentioned that earlier about just an inherent distrust, you're definitely not wrong. It's real and if the people who are writing the checks are okay with the teams being able to fail and gracefully fail and recover, then that's when you build that solid relationship.

Michael: Building up trust in the organization is important to get around the thing everyone knows: once you have a metric, people will figure out how to

game it, and how to make themselves look good. So what have you seen? What are some ways to get more people to trust the metrics for furthering the genuine thing that they're trying to do, to give you those health gauges?

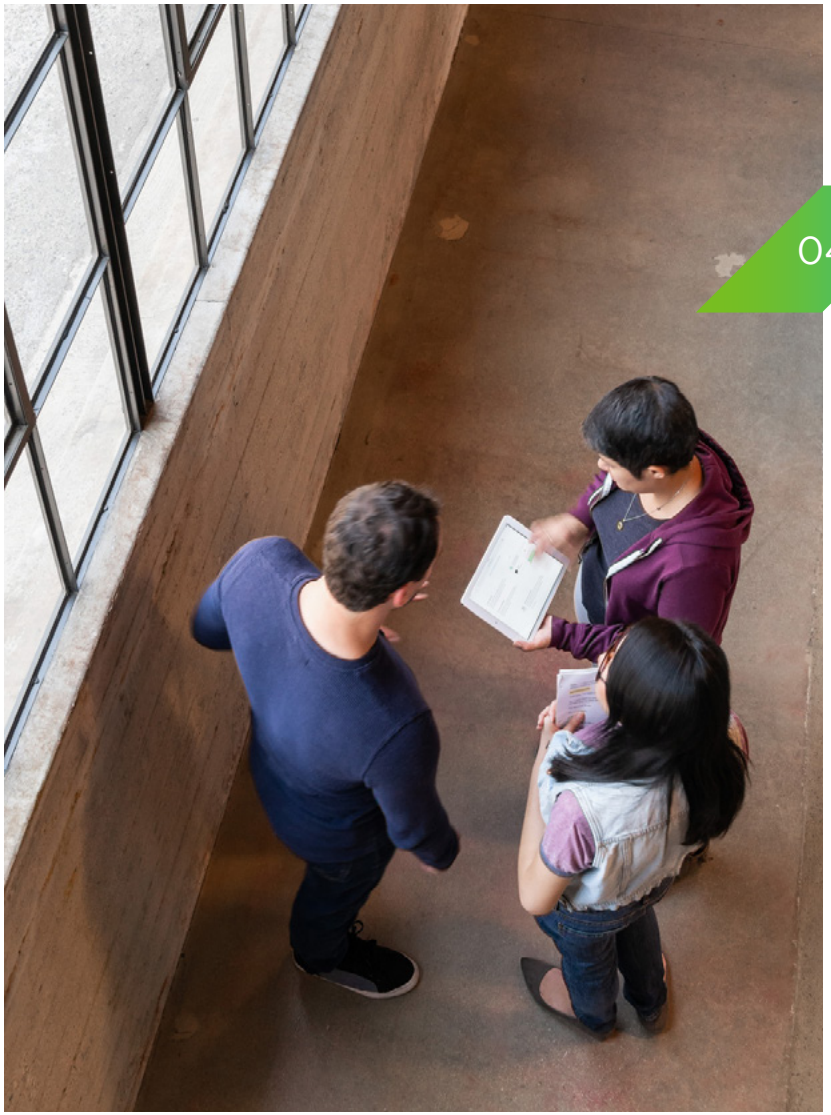
Jim: I think you need to avoid comparison. But it's so tempting to do — you've got all the dashboards and with all the data lined up that's the first inclination. Metrics may not scale up as well as you might think, just as with sprint velocity, each team has its own way of doing things. For example, you might have apps that are huge, monolithic things, so some of those metrics are going to be skewed. Because you have to deploy more frequently to get things changed versus, say, a microservice.

Some teams may be more diligent about addressing tech debt, and it's going to take them a little bit longer to deliver that feature. Those are some areas where you may not see much when you look at these metrics, but if you can dig deeper that might be able to help other teams or set a focus on what is expected. If the objective is to compare teams to find trouble areas, it's probably a lot more effective to put it in the team's hands so that they can figure out better ways to work and make their own improvements. Sharing data can be useful to visualize the same trouble areas occurring across teams and then presents a great opportunity to devote devops energy to resolve them.

Paul: That's definitely true, Jim. That's why I always have found velocity to be a bad metric as opposed to volatility. Over time, teams will normalize to an expected output. And if your volatility is generally flat, then your team is probably very efficient. Once you start to see spikes up and down and volatility, then you can start to ascertain what's going on in the team, because you have newer team members that are still learning what's going on in the application. And if you have a more mature team, then you might have additional things that you need to take a closer look at. I like metrics like volatility as opposed to velocity. For that specific use, there's a whole other bunch of different metrics that are not apples to apples; you have to figure out what the right metrics for tracking individual teams look like.

Michael: Volatility versus velocity as a way of adding in exactness into a metric, right? To drive towards what you're really interested in. Something that you might be really interested in could be, "is this team functioning well — nevermind the speed or the exact number?" Something like volatility gives you an indication of that. And it reminds me of way back in the beginning of agile, how there was a lot of thought into also putting in exactness and metrics, like using story points instead of feature points, and all these things that were made up to emphasize more of the qualitative nature of things rather than quantitative. Again, so that you can have a metric that isn't scary to some extent, or can't be used against you.

Paul: Yeah, I've seen, Coté, where they can be used to manipulate metrics for the sake of reporting. If you extrapolate out, if your team tends to sandbag on stories and points, you'll see the numbers will never necessarily lie. If you sandbag three-point stories, you will see that every three-point story should have a certain amount of time that it goes from into the backlog to being completed. And all of these things will normalize over time. It's all how you use the metrics and if you're using them in the right way: the right metrics for the right teams.



04

The (im)possible Task of Measuring Customer Experience

- Customer service (ticketing)
- Applying agile methodology to metrics themselves
- Qualitative results like “happiness” in favor of numbers

Michael: We haven’t necessarily covered “business value,” or “was the software useful? Did it actually achieve some goal that helps our business out? Did we achieve business value, or outcomes?”

Unless you’re just measuring revenue, the above can be hard to measure. To that end, there are some other stories we have about being careful on how you’re measuring this outcome. Even though you might have really good metrics, they might not be as useful as you think.

Jonathan, you had a story about chat ops that wraps a lot of this together, about the support that you’re providing and how to measure the business value of it. What happened in your experience with trying to figure out value?

Jonathan: Yeah, so being on a platform team, I’m very customer service focused. My customers are basically every software development team at Garmin. And they’ll have questions or need things from the platform team. We run a Slack channel that is very active, and we respond really quickly in that slack channel. Our customers are really happy with that because they get really fast responses on what they need. But our management asks, “Oh,

what do you guys do?" We answer, "have you looked in the Slack channel?" And they answer, "Well, we were looking for your ticket velocity."

There's a tension there, right? I could create a support ticket for everything that I do in the Slack channel. But from my perspective, it defeats the value of the Slack channel. I'm trying to keep my customers out of "please make a ticket, and then I'll answer you." I don't want to slow my customers down like that. I don't want to slow myself down like that.

There needs to be some automation, and we're talking about building a Slack bot for this, so that we can get the best of both worlds. But if you have to, if somebody has to make a ticket in order for something to happen, what you really end up doing is just putting lead weights on people's feet. And that just slows everybody way, way down.

Our experience with chat ops has been obviously great for our customers, but not so great for when we go to our management. We say, "Hey, we're overworked. We need another person." And they say, "We're looking at your tickets — do you really?"

Michael: Some business gurus say that if the goal of metrics becomes to improve metrics, things are going to go terribly wrong. So it's always important to focus on why you're using these metrics in the first place. This is something that in all walks of life, but especially in software, we easily lose track of. It's important to make sure you're evaluating if the metrics that you have are helping, and looking at the feedback on them and evolving them. It can be very difficult for people to retire anything, right? Then this metric, which was so awesome, is no longer awesome or useful. Perhaps we should come up with a new one.

Jim, I think you had an interesting point that it's also easy when you're obsessed with metrics to basically lose track of the "unmetric things," the qualitative value and comments that people have. Can you share your thoughts on that topic?

Jim: Yes, when we adopted the whole Tanzu methodology and the platform, we did a few test runs of apps through it. But then when we started to add real commercial ones, our internal customers were delighted. In fact, they started wanting to pop more and more products through this whole system. They were delighted, and that was a very important metric. And our executive team, they love it. That's the goal right there.

Now we've got people wanting to come in and do more projects here. But the problem is, they don't necessarily follow all the practices that we do, they just want to use say, Tanzu Application Service and not follow the practices. And with that in mind, it's time maybe to measure a little bit more and see where they are compared to the others. That's where we enter into the metrics.

Michael: There's an idea that y'all have been growing in my head throughout this, and the topic has come up several times that comparing different teams based on metrics is dangerous, is fraught. It makes me think of a very direct reason, which is that metrics misused will be used to reward but more importantly punish people, and the punishment often in the corporate context is withholding things. It's almost as if people in the mindset of using metrics to compare teams are a little too competitive for how those being measured by the metrics would like. But really, it's probably a good idea to figure out what you don't want to do, like pitting each team against each other.

Jonathan: That competitive nature, it really can change the dynamics between teams, right? Where, if I do this, I might lose my advantage. Or if I do this, I might make them look good. I'm going to try this other thing that makes my team look good. And suddenly, you're not necessarily doing the best thing, but you're doing the best thing for a small group of people. If you can somehow combine all of that with a servant leadership mentality where everyone is focused on driving the whole train forward, you're going to have to be more successful.

Paul: Yeah, I wanted to add on that too, because I shared an anecdote earlier about an organization that went through a digital transformation. We ended up with several product teams, and I want to comment that you're so spot on, Jonathan. Those things can become points of pride between teams, but they can also become very envious for other teams, if they have certain difficulties that they have to work through that other teams don't have. We were lucky enough to have leadership in our area and all the way to the top to our CIO, who was taking the lead on making sure that everyone was collaborating together and feeling like they were a bigger part of just their small team. That was super important to building that trust and not feeling like metrics were going to be used for ill will.

Jonathan: You can create massive positive buzz in your office, and not the high stakes positive buzz that we talked about earlier. But genuine positive buzz through this kind of thing. If somebody knows that, hey, I had this idea, and it blew up, and we did this amazing thing with it. Like, you're gonna get so many more of those. Or you could go with I had this idea that didn't work real well, and everybody blamed me, so I just don't have ideas anymore.

Michael: Well, we have a lot of great audience questions to answer. Let's share those now!

Audience Q and A

Q: What is your recommendation on how to track these elements in-house, as the source data could reside in multiple places? How does one gather these metrics? Is it even possible?

Paul: You can use tools to do this, and you can build applications to do this, too. We built an enablement team in my previous job: a team that tried to make things easier for everybody. And it could be anything from building a shared library that can be used across teams to working on security that everyone can use, or a metrics library that you can just plug into your application and make it easy to pull stuff out.

We had deployments for each team where you could use as a configurable variable, a Pivotal Tracker ID, and then you could pull out story IDs. From the time that the story was started to the time that it was closed, you could pull those numbers out, and we built dashboards around them. So you can use APIs and applications to do this. It does not have to be manual, because that does not sound like any job that anyone should have to do - to manually pull numbers out.

Jonathan: Yeah, I can think of three ways. And the problem is that your organization may be using all of them at the same time. That's where the real issue comes: how is one bigger than the other? Like, how do you gauge size?

We're a JIRA shop. So you might have JIRA tickets for each story that you're working on. And then you might have a release ticket that ties all of those

JIRAs together. And that release ticket is what goes out the door. So if you're tracking those releases, you can track okay, it's been 30 JIRA tickets. And that was 60 points that we just released. So that could be one way to do it. If everybody's doing that, you could have something somewhat consistent. But going back to story points, you might have a team doing story points on the Fibonacci scale, you might have another team using t-shirt. So it's hard to quantify that.

Another method would be to just count CF pushes, or count the number of times a new app is deployed. That could be a way to do it, or count the number of times that you're Maven versions, in whatever your repo is, whether you're using Artifactory, or Nexus or whatever that is. Look at, look at how many times you see the minor version bump in there, or the major version bump in there, and track releases that way.

So those are all different ways you can do it. The problem is, some teams might be using releases, some might only be counting the times they push. So I may not do a release at all, we got away from that for a little while. If you're constantly rolling forward, there's not really a need for a Maven release, because you don't really care. It's not "well, we're on 15.2.3, and we've got a rollback to 15.2.2." If you're always rolling forward, this was the last commit that was good. Let's fix it.

Q: Have any of you encountered working with external development teams that may occasionally interact with multiple applications within your organization? If so, how has it impacted your deployments and metrics?

Paul: Great question and the answer depends largely on what levels of rigor your teams place in testing those integrations, and the contracts you build between consumers/producers.

Jim: We have our lab, and have also added other teams from other corners that don't necessarily buy into all the hype. As a consequence, we start to see drifting configuration, which creates more tech debt.

Q: Good discussion! Can you all talk about the importance of driving a 'Continuous Improvement Mindset' and value of 'feedback' to mitigate the fear of being measured for punishment versus understanding the need for measurement to improve. Also, trust is HUGE! We've spent the last 3 years on our DevX journey focused on building trust. How have you all tried to establish trust?

Paul: This is such a big question to try and answer. It warrants its own discussion honestly, but I'll give it a few words and try to capture the major themes. Trust is definitely earned. In organizations that are making organizational changes, there will always be naysayers and individuals that don't necessarily want to see you succeed. I know that's hard to fathom, but what I want to point out is that those early wins seem to be more valuable. If you can build, test, and deploy even the smallest piece of business value more effectively than you've done it in the past, you've already won.

Jim: Yes, to me, trust is a two way street. Management buys the platform and practices, with perhaps some initial distrust on the part of devs. If dev teams deliver faster, management begins to trust the people and process. In return,

management trusts devs with new business and more such projects. Devs keep delivering, and trust goes up even more. Both management and devs then want to ensure that there are few things that could wreck this partnership, including very careful consideration of using metrics. Management trusts the teams to do their work, and devs trust that management won't do things that could be perceived as pitting teams against each other or looking for ways to 'help' apparent underperformers.

We focus on true trouble areas like security vulnerabilities or production outages. It's something we can all agree is important. As engineers, we are naturally curious as to how we can improve. If this topic is placed in our hands as a tool we can use, then chances are much better that it will be used for improvement.

Q: Do you all think it's more important to index on capabilities of the teams and use metrics to help measure the capabilities, but tailored to the given types of software systems they manage or manufacture? (i.e. legacy, monolith, micro-services, functions, etc.)

Jim: Regarding the topic of software systems, I like this idea a lot. It certainly would be interesting to see if the expected characteristics for each type of project line up with actual delivery data. There are so many other factors that might be hidden in such an evaluation such as what is the level of TDD being used: unit testing? Acceptance testing? How rigorous are these? What is the team's position on resolving tech debt (security issues, config issues, etc)? Proactive or lazy loaders?

Q: Thoughts on how management shouldn't have access to team metrics because they should be focused on outcomes not outputs? (i.e. metrics are for teams to self-evaluate)

Paul: I don't believe this to be true. I think that the more data (evidence) that you can provide to your stakeholders around quality, efficiency, and speed to business value can help you build trust from the people that can help foster a great environment for building incredible software.

Jim: I somewhat disagree. I think that repeated faster delivery is quickly noticed without any metrics at all (aside from using a calendar). While I like the idea that metrics data should be shared between management and devs (unless there is a profound culture shift), there is an instinctual inclination to compare, desire to fix/optimize, and share upwards. I'm jaded, but this is why I think metrics should be for the team only, and potentially other teams that want to learn from (more) successful teams. If teams agree on sharing metrics with management, then that's great. Self-managed team culture is strong in our shop.

Jonathan: I think leadership should have access to team metrics... but "with great power comes great responsibility." Leaders need to understand more than just the numbers. Productivity can be influenced by a number of actors. Managing by numbers can engender massive distrust, and therefore poor employee morale. And yes, outcomes are more important than outcomes. One useful way to use metrics would be to look at a team and figure out why their numbers aren't meeting expectations. Likely it has more to do with some issue the team is dealing with; solving the issue is more likely to produce positive outcomes than using metrics as a stick.

Q: What do you think about a metric that tracks the number of GIT commits for each engineer? I have a strong opinion on this, but am interested in a perspective outside my company.

Paul: I hate using metrics for this purpose. It seems to be ripe for fostering distrust among engineers and management. It doesn't begin to tell the story of what happened in a given period of time. I could be pairing with someone and one name gets attached to the commit. I could be spending time doing merge request reviews. I could be working ahead of the team researching solutions for the next tracks of work for the team.

Jim: Yes, squashed versus individual committers, plus forked repo commit data, can also skew things. While we do accumulate this data — it's for a binary query — is this dev breathing in Github or not?

Jonathan: I wholeheartedly agree with Jim and Paul. This idea reminds me too much of a Dilbert comic. As with the previous question, this metric would be too easily abused, by both leadership and individual contributors.

Q: When you say "deploy once a day" are you referring to production deployments?

Paul: You don't realize the business value until it's in production right? If you build the proper testing framework around your application, you should feel confident that the new changes you make will not cause issues to your customers (internal or external).

Jim: I agree and I'd expand it to not think of it as once a day, but rather, be able to deploy anything in a moment up to prod at any time. The TDD foundation helps make this assumption hold as long as there are no broken tests and if there is a solid working copy in blue/green to utilize.

Jonathan: I agree with both of these.

Q: How do you compare cycle time across different "ideas," as some ideas are big (and have longer cycle times) and some are small with shorter cycle times?

Jim: I'd say we do it by story pointing, but then each team may use a different system on pointing (complexity versus time-based), so it's somewhat dangerous to look at things like points and velocity. And pointing is a front-loaded estimate allegedly from experience, but the fight often differs from the battle plan. That said, this early t-shirt sizing is the closest thing we have about the thing. I could add metadata, but the more you add, the less likely teams will keep it up.

Q: This is for Jonathan because I've been a lifelong Garmin customer, but actually this is also for everyone. Can you talk about how:

- Roll Forward — does it play a role in your software manufacturing process?
- Versioning Drift — have you observed the impact of versioning drift on your dev team's ability to push changes due to dependency management and the pace by which teams upgrade. Have you all established Versioning Drift Policies via the Semantic Versioning Standards?

Jonathan: Thanks for the Garmin shout-out. I love our products too. Regarding the first question about Roll Forward — yes it does. When we first adopted Tanzu Application Service (back in the day) we talked a lot about deployment pipelines being fast enough that regular releases were as fast as emergency releases. We convinced both dev teams and management of the value of small releases and how they enabled roll-forward instead of rolling back. As for versioning, we don't currently enforce versioning on our developers. We encourage teams to grab newer versions of things like Spring Boot and Java as they develop.

Jim: Versioning Drift - yes from buildpacks, the choice of language version to individual dependencies. We added detection of deviance from bp and language and for individual dependencies that have known sec risk scores >= 7, we create issues on teams tracker boards where they plan their work. We also add these issues to an overall public dashboard which tends to drive action. We place the responsibility on teams to address these items as they can alongside features (most teams are diligent about resolving these). At the beginning it was an adjustment to normal op procedure. In regards to using the Semantic Versioning standards, we are not driving teams generally to a specific major or minor or patch as a course of habit - but more in reaction to security posture and deprecated versions.

Q: Using metrics to suggest possible root cause (suggesting someone did something) may be better received if trust is established. Teamwork, trust with teams can be improved by establishing credibility which includes building upon integrity, intent, capabilities and results, correct?

Jim: Agreed, as trust is 2-way street. Focus on hardcore data about outtages and vulnerabilities getting resolved. Anything more than that (you should be doing this faster might ignore some of the important things 'slower' teams are doing that are really important).

What is a huge release and what is a long cycle time? What are the key building blocks to reduce cycle time from quarterly into weekly and then daily? Do you all use external consulting services to review your code and call out tech debt? The challenge with UX is that revenue is king over anything else and it's a steep climb to get execs to appreciate the UX value long term?

Jim: We strive to have smaller releases and shorter cycle times (hours, days). While some in our company are using SAFE, which tend to be bigger releases and delivery is every 2 weeks, minimum. Some crossover into months or quarters. Building blocks? We adopted the Tanzu practices of TDD, smallest stories possible, and CI/CD. It has paid off with satisfied business units.

Regarding external consulting services for reviewing code/calling out tech debt, my answer is "no," but we do have integration with Sonarqube and Checkmarx for detecting issues and putting findings into workflow. It's paying off!

Regarding UX, some execs think it is magic and that no code solutions will solve all the problems. I don't even know what to say to that.

Q: What is your recommendation on how to track these in-house as the source data could reside in multiple places?

Jim: We've built our own tooling around Github, which for us is the right fit.

Resources

Michael: Now you've heard several people mention Tanzu Application Service, which is a modern application platform for enterprises that want to continuously deliver and run microservices across clouds. We also have another platform that's an application toolbox for building a platform as a service on top of Kubernetes: Tanzu Application Platform. It's a modular, application-aware platform that provides a rich set of developer tooling and a prepaved path to production to build and deploy software quickly and securely on any compliant public cloud or on-premises Kubernetes cluster.

The diagram on the next page shows the full end to end view of the software process. There are many metrics — and the philosophy of a lot of the metrics we've talked about — that come in it. It's a very open platform built on open source components, with an open API-driven nature to it. If you're using it, you'll discover things you want to measure, and ways to instrument or measure your process.

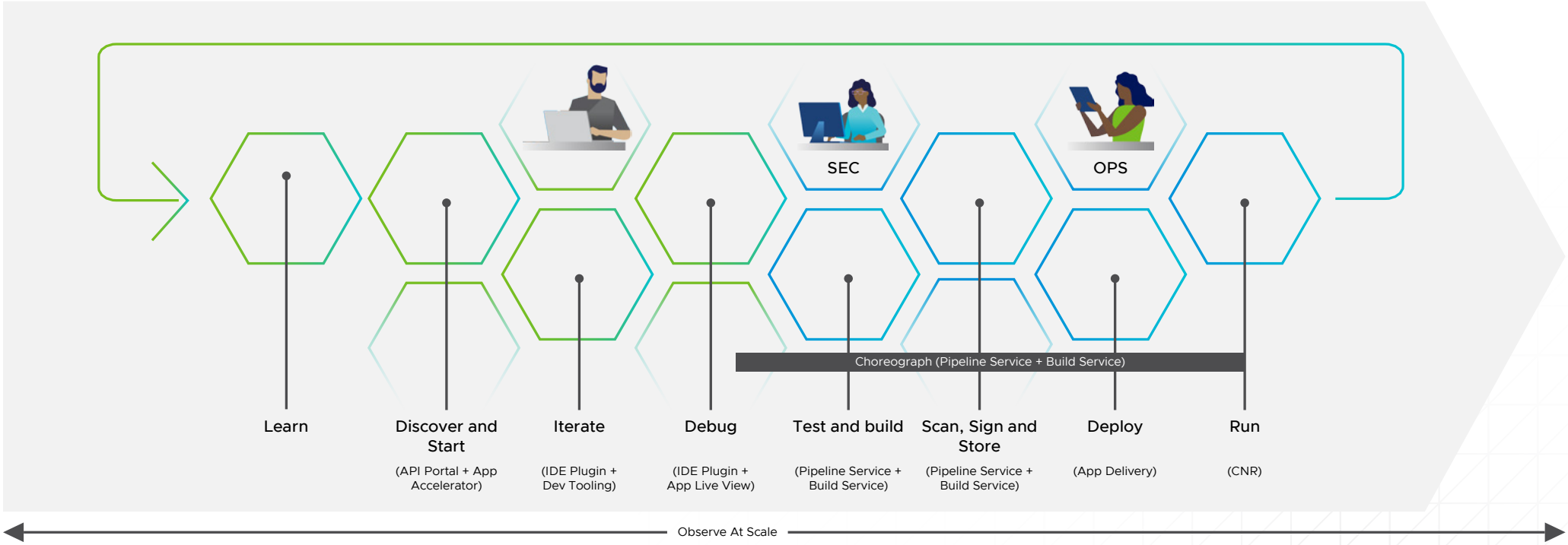
Watch the webinar version of this eBook!

VMware Tanzu whitepaper on Developer Toil: Hidden Tech Debt

VMware Tanzu eBook: The Legacy Trap

How Tanzu Application Platform Can Help

- Automates path to production: every step is automated, because devs just want to work on feature development
- Out of the box security: TAP scans source code, images
- Helps orgs adopt a cultural shift among key stakeholders – across Dev, Sec, Ops



About the Authors

Michael Cote

Senior Member of Technical Staff,
VMware Tanzu

@cote

Jim Kohl

Consulting Software Engineer,
Great American Insurance Group

@jimdkohl

Jonathan Regehr

Architect,
Garmin

@jonathanregehr

Paul Pelafas

Senior Principal Architect,
Kin + Carta

@paulpelafas