How to Misuse & Abuse DORA Metrics

Bryan Finster,
Distinguished Engineer & Value Stream Architect, Defense Unicorns

Author

Bryan Finster,
Distinguished Engineer
& Value Stream Architect,
Defense Unicorns

Bryan Finster has over two decades of experience delivering and supporting mission-critical solutions for very large enterprises. He is the founder and former lead for the Walmart DevOps Dojo with hands-on experience both executing continuous delivery for production systems and helping other organizations find and remove the constraints that prevent a CD workflow. He is co-author of *Modern Cybersecurity: Tales from the Near-Distant Future*, author of the *5-Minute DevOps* blog on Medium, and a frequent speaker on all topics related to improving the flow of software delivery.

My first experience looking at how to improve delivery in a large enterprise was as one of the engineers trying to solve a practical problem. How, with no existing automated tests and a scheduled week of twenty-four hour support after every delivery, do we deliver changes every two weeks with a 25-million-line-entangled monolith that we already struggle to deliver every quarter?

We decided that to make this better, we needed to solve the technical, architectural, and organizational problems to enable continuous delivery. We began by reading *Continuous Delivery* by Jez Humble and Dave Farley. Then we started the methodical process of solving the problem of "Why can't we deliver working changes to production every day?" We quickly realized that we needed to include two important topics in our list of studies: effective testing in the pipeline and methods for measuring improvement.

After solving many of these problems, we began experiencing something we hadn't predicted: the positive impact our work had on the teams. We built closer partnerships with our end users because we were interacting with them daily. We also built teams that had much higher morale and grew skills rapidly by working daily to find ways to do the work better.

In 2017, my teammate and I gave a talk at DevOps Enterprise Summit, "Continuous Delivery: Solving the Talent Problem," where we discussed the impact of this work on our team and how solving this problem grew the team's skills, morale, and "made us love development again."[1] We found that solving the constraints to continuous delivery and focusing on improving throughput and quality simultaneously were more effective tools for improving business goals than the years of failed "Agile transformations" with scaling frameworks that had come before.

In 2018, a groundbreaking book was released: *Accelerate*, by Dr. Nicole Forsgren, Jez Humble, and Gene Kim, based on years of research from DevOps Research and Assessment (or DORA). It was the first attempt at using statistical analysis to make correlations between organizational outcomes and ways of working. It was validating to see that our experience wasn't unique.

> "Continuous delivery improves both delivery performance and quality, and also helps improve culture and reduce burnout and deployment pain."[2]

1 Bryan Finster and Brent Pendergraft, "Continuous Delivery: Solving the Talent Problem - Walmart," San Francisco, presented at the 2017 DevOps Enterprise Summit, November 13–15, 2017, YouTube video, 27:12, https://www.youtube.com/watch?v=MHK16QNVXXU&t=6s.

2 Nicole Forsgren, PhD, Jez Humble, and Gene Kim, *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* (Portland, OR: IT Revolution, 2018), 56.

Using several years of responses from *The State of DevOps Report* survey, they established a set of signals and correlated those signals to organizational outcomes. Chapter 2, "Measuring Performance," contains a set of four key metrics they chose for tracking throughput and stability, because they recognized these as signals that broader improvements existed in the organization.

Since that time, those four metrics have become commonly known as "DORA Metrics," and their usage is spreading across the industry. On one hand, this is encouraging because more companies are starting to talk about measurable improvements to delivery rather than "Agile transformation." However, a different problem was created.

When *Accelerate* was released, many of us saw it as the "easy button" for communicating what we were trying to accomplish by measuring and working to improve continuous delivery across the enterprise. We purchased hundreds of copies of the book, distributed them to middle and upper management, included an insert for why they should read the book, and pointed them to page 19, where the four metrics were shown relative to organizational outcomes. In the process, we unintentionally communicated that measuring human activity was simple instead of complex, and that all we needed to do was set these metrics as goals to measure our way to improvement.

Since then, many of us have been seeing this problem spreading across the industry. More commercial tools are including DORA metrics, but the context is being lost and the promised improvements to delivery, quality, culture, and pain are not being realized. My goal is to help correct the course on DORA metrics usage so that every organization can see the benefits we know to be true.

DORA Metrics Explained

What are the DORA Metrics? They are a set of four metrics focused on throughput (lead time and deploy frequency) and stability (change-fail percentage and mean time to repair).

These metrics were chosen because:

> "We have found a valid, reliable way to measure software delivery performance that satisfies the requirements we laid out. It focuses on global, system-level goals, and measures outcomes that different functions must collaborate in order to improve."[3]

Having experienced what changes must happen in a value stream to improve throughput and stability, I concur. Everything must improve to improve throughput and stability. However, only focusing on these metrics ignores something important. These are only leading indicators of improved business outcomes *if* we deliver the right thing and do it sustainably. There are also some problems with using these specific metrics directly in our improvement efforts. Let's look at each in detail.

---

3 Forsgren, Humble, and Kim, *Accelerate*, 23.

Deployment Frequency

> "…we settled on deployment frequency as a proxy for batch size since it is easy to measure and typically has low variability. By 'deployment' we mean a software deployment to production or to an app store."[4]

This is usually the first metric people focus on. It is the easiest to measure because it can be tied directly to delivery artifacts. If we don't understand the intent of this metric and the reasons that smaller batches are more desirable, then we will fail to measure and interpret the results correctly. Focusing on this metric without quality backpressure yields results that have caused some organizations to abandon the journey.

Lead Time

> "Lead time is the time it takes to go from a customer making a request to the request being satisfied."[5]

Lead time is an important indicator of how quickly we can satisfy customers and our ability to get feedback on the value hypothesis we are making. What we are delivering is probably wrong in some way, so we need to find out quickly, fail small, and adjust.

> "…there are two parts to lead time: the time it takes to design and validate a product or feature, and the time to deliver the feature to customers."[6]

For the sake of extracting some meaningful data, the part of lead time during which we decide if something should go on the backlog or gather more user data was not measured. They focused on the delivery portion to limit their variables in the data. They give two different definitions for the delivery portion. First:

> "…the delivery part of the lead time—the time it takes for work to be implemented, tested, and delivered…"[7]

This is the development cycle time—the time from when work begins until it is delivered. This portion gives us some good insight into where we should look for improvement opportunities. This isn't what was measured though. Remember that their goal was finding correlations, not improving them.

---

4 Forsgren, Humble, and Kim, *Accelerate*, 16.
5 Forsgren, Humble, and Kim, *Accelerate*, 14.
6 Forsgren, Humble, and Kim, *Accelerate*, 14.
7 Forsgren, Humble, and Kim, *Accelerate*, 14.

The "lead time" values they show are based on the definition they used for the survey question.

> "We measured product delivery lead time as the time it takes to go from code committed to code successfully running in production…"[8]

The delivery cycle time is only the portion of the development cycle time that covers the CD pipeline. The reason they looked at this cycle time is that it isn't dependent on the size of the task. High-performing teams will focus on reducing the delivery cycle time to improve the efficiency of their quality gates to improve their confidence that they can deliver. A short cycle time here indicates a mature delivery process. However, the reason they measured this is different from what we need the metrics to tell us. We are working on reducing the batch size of work, not statistical analysis. We will need to keep an eye on this as part of improving our development cycle time, but it's only one of the problems to look at.

Change-Fail Percentage

> "We asked respondents what percentage of changes for the primary application or service they work on either result in degraded service or subsequently require remediation (e.g., lead to service impairment or outage, require a hotfix, a rollback, a fix-forward, or a patch)."[9]

Why measure this?

> "In the context of Lean, this is the same as percent complete and accurate for the product delivery process, and is a key quality metric."[10]

Smaller batches and shorter lead times are pointless if we are delivering broken solutions. Asking respondents to estimate how many changes require remediation makes sense in the context of a survey. "About 20% of the time, we'll deliver something and it breaks in some way." For our purposes, which change caused a defect does not meet our improvement goal. Our goal is to improve the "percent complete and accurate" by reducing the number of defects we create, not identifying which deliveries cause defects.

Mean Time To Restore

In a world of complex cloud environments, failure is not just an option, it's a promise. Improving uptime is important, but improving how we react to failure is more important. To measure MTTR (mean time to restore), they did the following:

---

8 Forsgren, Humble, and Kim, *Accelerate*, 15.
9 Forsgren, Humble, and Kim, *Accelerate*, 17.
10 Forsgren, Humble, and Kim, *Accelerate*, 17.

> "We asked respondents how long it generally takes to restore service for the primary application or service they work on when a service incident (e.g., unplanned outage, service impairment) occurs."[11]

As a survey question where people are estimating their outcomes, this works fine. The challenge for using this as part of an improvement plan is how to measure it in reality.

Metrics Run Amok

We've covered the DORA metrics, their intent, and how they do and do not apply as tools for measuring improvement. The *Accelerate* authors' goal was a statistical analysis of behaviors that high- and low-performing organizations exhibit, not to use these metrics to improve those results. We need to make sure we define our goals to ensure the metrics we choose align with the goals. What happens when we don't?

Making the right metrics visible in the right way for the right reasons is a core part of any improvement process. Measuring the performance of software delivery has gone through many iterations over the years with generally poor results. The typical pattern has been to find metrics that appear to make sense and then to use the metrics inappropriately. An example of this is velocity.

Velocity is the number of story points completed in a defined timeframe. When velocity is properly used, a team can plan workflow based on capacity. However, it is frequently misused and abused as a performance metric with goals. "We need your team to increase its velocity!" No problem. Increase the story points every sprint. We lose the ability to track our capacity and plan, but the person demanding we make the metric "better" is happy. We need to understand the intent of a metric and the possible consequences of measuring it to prevent misuse and perverse incentives.

I mentioned the problem we created when we accidentally implied that measuring behavior was simple and we could just measure our way to improvement when we pointed people to Table 2.3 in *Accelerate*. The outcome is a set of common misuses of these metrics spreading across the industry.

Focusing on Speed

"We need to deliver features faster. Deliver more frequently so we can be high-performing." Deployment frequency is about batch size, not speed. Smaller batches of work correlate with higher quality and improved value delivery with less waste. If we focus on speed, quality will inevitably suffer. If we focus on smaller batches and improved quality processes, speed will increase. No matter what, measuring deployment frequency without using quality metrics as guardrails will result in poor outcomes. We see this happen, and it causes organizations to abandon CD and return to delivery flows that will never be agile.

---

11 Forsgren, Humble, and Kim, *Accelerate*, 17.

Moving the Goalpost

"We produced ten releasable changes this week. We are high-performing!" Were they delivered as ten separate artifacts? One big release? Were they delivered at all? This isn't a vanity metric. The point is to reduce the size of the batches we deliver as a method for improving how we work, not what we could have delivered. Undelivered change is only waste.

Goal Setting

"Here are your DORA metric OKRs for the quarter." The goal isn't better DORA metrics. The goal is to improve our ability to deliver the right thing sustainably with lower cost, higher throughput, and more stability. OKRs (objectives and key results) should be focused on our desired business outcomes. If we are only focusing on DORA metrics, it's similar to driving a car with the goal of acceleration. Accelerating to where? Do we care if the engine explodes?

Measuring Our Way To Enterprise Transformation

Leaving aside the fact that we should be continuously improving instead of running "transformations," DORA metrics won't get us there. They inform us about one aspect of improvement, they don't fix things. If we simply get a dashboard and do not buy into using it to identify improvement items, then nothing will get better. Also, if we lack the right culture, the metrics will only cause fear, not improvement. We need to communicate the intent frequently and openly and build the level of trust required to improve our organization. If we have the right culture, there are some leading indicators we can gamify to help spread information. This point will be elaborated on later.

Vanity Metrics

Many tools are now including DORA metrics dashboards because customers are demanding observability. However, these dashboards can become problems. Aside from the fact that each is interpreting the meaning of the metrics differently, they are often used as "vanity radiators" instead of information we can use to help us improve. The following example, is a mockup of a DORA dashboard I've seen used.

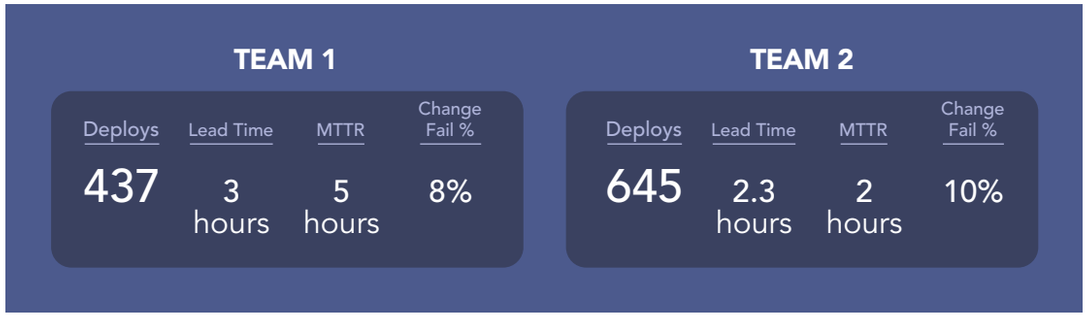

Figure 1: DORA Metrics Dashboard Example

This exemplifies the growing problem. First, it focuses most attention on deployments while giving no useful indication of whether things are getting better. Are we improving at reducing batch size and quality? The message "Look how often we deliver!" is not useful either for the team or for those wanting to help teams. It also encourages comparing teams against each other instead of teams comparing themselves against their past selves. The most useful information we get from this is that Team 2 may be using a hotfix process outside of their normal flow to resolve incidents—a big red flag. These may make the organization feel good, but they don't help our goals.

If our goals are misaligned or we do not communicate intent clearly, we can create unintended consequences that actually make things worse. We need to continuously compare the behaviors we are seeing with our intended behaviors and not just look at dashboards to understand if the organization is improving. We need a plan to keep things on track.

Tools for Improvement

So, how can we avoid using metrics destructively? We need to make sure we understand and clearly communicate our goals. We want to find and remove technical, process, and organizational constraints to continuous delivery because solving these problems "...improves both delivery performance and quality, and also helps improve culture and reduce burnout and deployment pain."[12] We also need to make sure that everyone understands that these metrics are tools for the teams to use to help self-improve, not indications of performance or badges of achievement. This needs to be clearly and frequently communicated. Using metrics correctly requires education.

Reduce Batch Size

We need to reduce batch size because smaller batches of work are easier to verify, they tend to fail small, we are less likely to suffer from sunk-cost fallacy, we amplify feedback loops, etc. How small should they be? As small as we can make them so we can get production feedback on what we are trying to learn. Working to reduce batch size acts as a forcing function for exposing and removing hidden waste in upstream processes. There are several batch sizes we are trying to reduce.

Deployment Frequency

There are two common arguments I hear against increasing deployment frequency.

The first is a misunderstanding of "valuable." "We don't want to deliver incomplete features, because the customer can't use them, so we aren't delivering any value." There are more stakeholders expecting value than just the end user. One of those is the product team. We are reducing the level of inventory waste in our flow and getting rapid feedback that we haven't broken existing behaviors with

12 Forsgren, Humble, and Kim, *Accelerate*, 56.

the new change. This gives us feedback on our quality gates and also lowers the risks of delivering a production fix.

The second objection is, "Our customers don't want changes that frequently." This comes from a misunderstanding of what CD is for. Yes, we can deliver features with continuous delivery. Notice that one of the four key metrics is MTTR and not the number of features delivered. A primary purpose of CD is production support. When production has an incident or we have a new zero-day vulnerability, they want changes that frequently to resolve those problems. Can we resolve them? By improving delivery frequency, we are continuously verifying that we can still deliver those fixes safely.

To measure this, we can start by using our workflow management tool. However, we need a common definition of "done" that includes "delivered to the end user" and not the measurement of some intermediate step like "ready for someone else to do something with it." However, if workflow metrics haven't been visible before, tool usage is usually inconsistent. (We'll explore this problem more with lead time.) A better solution is to measure delivery directly by instrumenting the tools we deliver with. There are a growing number of commercial and open-source tools for this.

After we make this visible, we can monitor our trends over time. Note that relentless improvement is the goal, not becoming an elite team. We want to reduce the batch size, but we also don't want to over-optimize this at the expense of the rest of the flow. We should note that only measuring the rate of deploying change is insufficient for decreasing the inventory of work in progress. Incomplete work can be deployed but hidden by configuration. This metric can give us insights, but we still need to educate everyone on the intent to keep batch sizes low.

Again, this is not a performance metric, and attempting to set a standard target value for all teams is a mistake. It is also not an indicator of how awesome the team is. This is like a speedometer that we can use to decide if the current value makes sense in our local context rather than a lap-timer to see if we are the winners. Every team will have its own delivery context and will eventually level out at a cadence that makes sense in that context. That being said, we shouldn't pat ourselves on the back for being high-performing because our organization of a hundred teams deploys to production once a day. That's only one delivery per quarter per team. Those are not small batches. Can we deliver today's changes today? If not, we have more work to do.

Lead Time

> "Shorter product delivery lead times are better since they enable faster feedback on what we are building and allow us to course-correct more rapidly."[13]

The term "lead time" can cause confusion when it's used to describe a subset of the total lead time. In Lean, the lead time is the time from request to delivery. There are many steps with nested cycle times

13 Forsgren, Humble, and Kim, *Accelerate*, 15.

within the total lead time. Ubiquitous language is important for communicating clearly, after all. I recommend being more specific when defining terms for the organization. We can see these cycle time relationships in Figure 2, and all of them impact batch size.

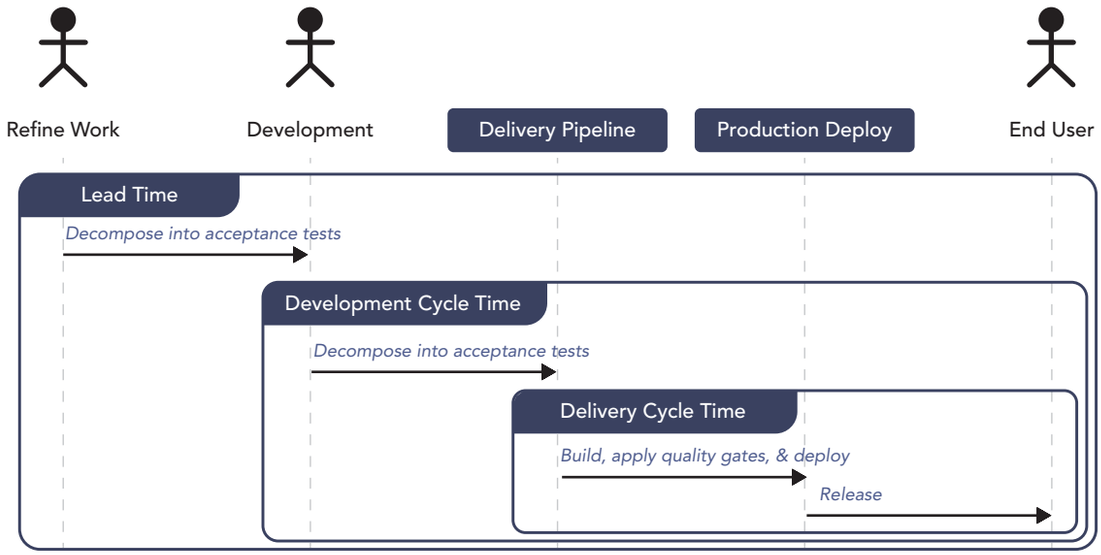

Figure 2: Cycle Time Relationaships

(This image is provided under a creative commons license: CC BY 4.0 - Bryan Finster - Commercial use with attribution.)

Delivery Cycle Time

Delivery cycle time was surveyed for *Accelerate*—the time from code commit until delivery to production. We need a shared understanding of the delivery pipeline to measure this correctly. The delivery pipeline consists of every manual or automated quality process that is executed after a change is committed to version control until the change is delivered to production. The purpose is to make every change prove it meets the organization's definition of production-worthy. Tracking pipeline cycle time acts as a forcing function to improve how efficient our quality process is and to automate manual steps. This helps us reduce the cost of delivery to enable us to deliver the small batches we want.

If we are using any method to deliver changes other than our defined pipeline, then we need to prioritize improvements to fix this anti-pattern. One of the reasons CD reduces burnout is that when we need something fixed immediately, we can confidently do that because we are not bypassing all of our quality and security checks. Our delivery pipeline must be the safest, most secure, and fastest method for delivering production fixes.

Continuous Integration

"That's not a DORA metric!" No? It's not listed in the chapter on measuring performance, but the goal of that chapter wasn't to track our performance efforts. However, in Chapter 4 on recommended technical practices, the authors define CI behavior and how to measure it.

> "Following our principle of working in small batches and building quality in, high-performing teams keep branches short-lived (less than one day's work) and integrate them into trunk/master frequently. Each change triggers a build process that includes running unit tests. If any part of this process fails, developers fix it immediately."[14]

Continuous integration metrics are leading indicators for DORA's four key metrics. They are also easier to automate and come with recommended time boxes. During the first experience I had working to solve the CD problem, we posted the rules for CI in our team area and used them to drive our retrospectives until we found solutions for them.

- Work integrates to the trunk at a minimum daily.
- Work has automated testing before merging to trunk.
- Work is tested with other work automatically after merging to the trunk.
- All feature work stops when the build is red.
- New work does not break delivered work.
- Trunk is always deployable.

Solving these problems to execute CI is the foundation of the efforts to improve the organization. It is very effective at exposing gaps in testing, evolutionary coding practices, trunk-based development, work decomposition, and teamwork. It's also effective at shining a light on upstream issues. We'll see this relationship later.

We have two metrics to track: frequency of changes integrated into the trunk and duration of branches from the trunk. It is important that we measure using the trunk of version control; the place where production releases come from.

Because the definition of CI contains a hard timebox (branches removed in less than a day) with a smaller timebox (trunk integration frequency) nested within, and because CI is not contextual to an application but is simply a high-performing workflow, we can provide direct feedback to help teams identify issues by gamifying some of the signals and providing tips that work in every context.

Development Cycle Time

This is the time from when work starts on something that changes the behavior of the application and the time it is delivered to the end user. "That's not a DORA metric!" I'd again refer you to Chapter 4, "Work in Small Batches."

How small is small? It's typical for teams who have only been taught Scrum to refine work until it can fit in the sprint. Therefore, five- to ten-day stories are common. It's also common for those to take ten to fifteen days to actually finish development due to the lack of clarity in the stories. Because the desired outcomes are unclear, exploratory coding is also common. This leads to creating tests after

14 Forsgren, Humble, and Kim, *Accelerate*, 44.

development is complete. These problems all prevent CI from functioning. To resolve this, we shrink the timebox for a story then fix everything that prevents us from staying within that time box.

In 2012, Paul Hammant, author of *Trunk-Based Development and Branch by Abstraction*, made the following suggestion:

> "Story sizes should average as close to one day as possible. If they don't, your Agile project is going to be harder for nearly everyone involved. If your average is significantly greater than that one day, then change something until you get there."[15]

This may sound unachievable, but we have seen how effective this is in the enterprise Dojos. Dojos work with teams to help them discover better ways of delivering in their context. A primary tool for doing this is the hyper-sprint. A hyper-sprint lasts for 2.5 days and includes every activity required to deliver small changes, get feedback, and adjust.

This short time box forces a mind shift to change perspective on what defines "small." This also acts as a forcing function for uncovering and removing upstream impediments with missing product information, external hard dependencies with other teams, Change Advisory Board compliance theater, or other organizational issues.

Continuous integration is hard to achieve if teams do not have the skills to slice stories into thinner value increments with testable acceptance criteria and deliver them as a team. This skill set is our highest priority.

Build Quality In

Reducing our batch size is one aspect of increasing efficiency. However, if all we do is break things down into small pieces and shove them out as rapidly as possible, quality will suffer. We need guardrails.

Defect Rate

DORA asked respondents to report their change-fail percentage. As we discussed earlier, this is a good question for comparing organizations based on a survey, but it isn't the best for trying to measure as part of our improvement process. It requires identifying which release caused a specific defect. However, defects are more complicated than that, and spending the effort to find which release caused which defect generates complexity without adding offsetting value to our goal.

We need to track the rate at which defects are created. If we are improving our quality process, we should expect the number of defects created over time to decrease even as we maintain or improve our delivery frequency. It is important that we do not focus on this in isolation. It needs to be tracked concurrently with delivery frequency, CI metrics, and the other batch-size metrics. Grouping these metrics

15 Paul Hammant, "Call to Arms: Average Story Sizes of One Day," Paul Hammant's Blog, April 24, 2012, https://paulhammant.com/2012/04/24/call-to-arms-average-story-sizes-of-one-day/.

prevents us from using quality processes that increase batch size or reduce batch-size at the expense of quality. We are forced to find solutions to both that make us better as an organization.

MTTR

MTTR is the most problematic of the four DORA metrics to measure at the team level. Tracking at the product level is relatively easy if there is a disciplined approach to tracking customer-facing incidents and capturing the time from detection until customer impact is resolved. However, this is usually done by humans updating an incident tracking tool, so accuracy depends on standardizing practices and executing with discipline. To get more finely-grained data that specific teams can review and improve at the service level requires that each team instrument their components to track failure. This will take time, but operational observability is always worth the investment.

This is another lagging indicator and an important forcing function for improving our standard delivery process so that we can use it to respond quickly to incidents, security issues, or any other emergency in production. If we are celebrating a low MTTR while also using nonstandard, hacky, hot-fix processes when an emergency happens, then we are misunderstanding the purpose of this metric.

Not Just Robots

Continuous delivery is a holistic process of taking product ideas, decomposing them into small increments of value hypotheses, delivering them to the end user with a heavily automated standard process, and getting feedback to inform future plans. We need a view into every aspect of the flow to improve it. (See Figure 3.)

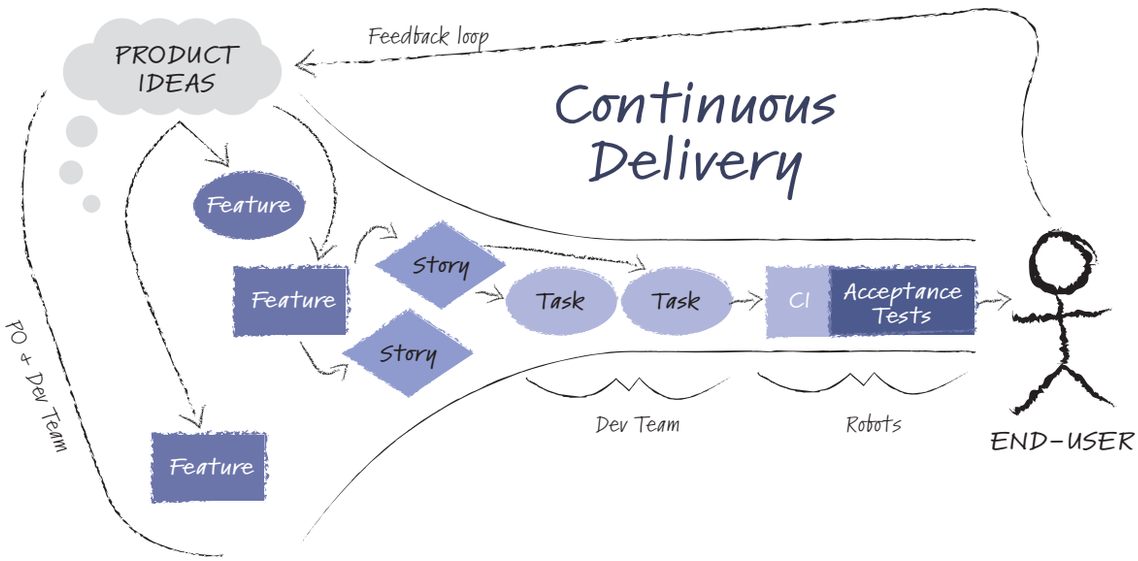

Figure 3: CD Flow Example

(This image is provided under a creative commons license:CC BY 4.0 - Bryan Finster - Commercial use with attribution)

Radiate the Information

If a metric is collected and no one looks at it, does it exist? We need information radiators that display the information in a way that causes an impact. Looking at the dashboard, we should have some idea about what action to take based on the current state. Also, we need to make some decisions about the insights we want. Ultimately we are trying to help our teams identify and remove constraints.

While tracking metrics at the product level may give us planning insights, to improve flow we need to help teams remove friction. So, we need to measure at the team level whenever possible. The team level is also the lowest common denominator. Avoid falling into the trap that development metrics can be used to track individual productivity. It's destructive to measure individuals; they will be incentivized to focus on personal output rather than team outcomes.

Gamifying Improvement

If we have a culture of high trust, learning, and continuous improvement, then deliberate gamification can be a powerful force multiplier for improvement efforts. Metrics, like deployment frequency, will be contextual to the application. For example, if we are delivering to the cloud, we can deliver much more quickly than if we are delivering to a submarine that won't surface until a month from now. The cadence will also be dependent on how much control teams have. Some metrics are not contextual. Code integration frequency, branch duration, and development cycle time are examples. The product team has full control over these, and working to improve these is effective at uncovering impediments. For these, we can visualize trends in ways that encourage improvement (see Figure 4).

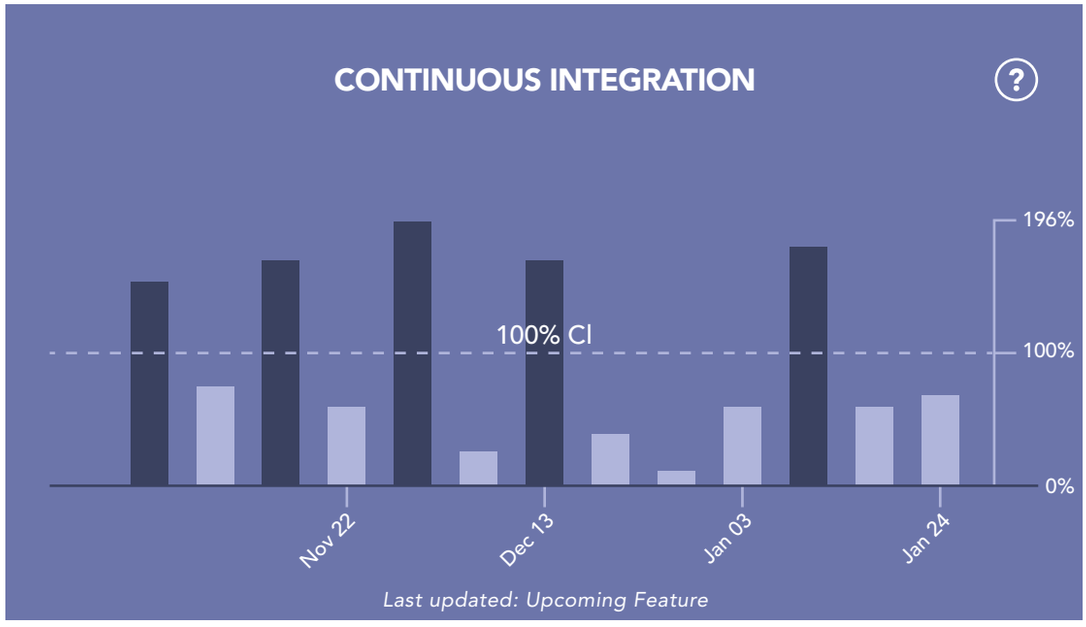

Figure 4: Example of Platform One's Holocron Delivery Dashboard

In Figure 4, we have an example from Platform One's Holocron delivery dashboard. We can see week-to-week trends of how frequently code is integrated into the trunk compared to the team's size and the definition of continuous integration above.

The "100%" line shows the minimum level of change to qualify for CI and tells us if we are improving or performing worse as a team over time. A word of caution though: This is a team-focused metric. Only the team understands the context and why the numbers are the way they are. Was there a holiday or people on vacation or sick? Using team-focused metrics outside the team will harm the value of the metric.

Showing where we are and where we want to go is only helpful if we have some idea of how to get there. Providing information co-located to the tool means we can implement on-demand help. Figure 5 is an example of help content for the CI metric to help teams self-discover effective tips.

GOAL OF MEASURING
Reduce the size of change: Smaller changes improve the effectiveness of code review, give faster quality feedback, and reduce chances of defects introduced during change conflict resolution.
Improve work decomposition: Solving the problem of how to deliver smaller changes uncovers issues with work decomposition. Solving the problem of work decomposition improves the quality of the requirements.
TIPS FOR IMPROVEMENT
Use a Continuous Integration workflow
Use Behavior Driven Development to decompose and better understand the work.
Use Test Driven Development to guide smaller changes and better application architecture.
Use feature flags, branch by abstraction, or other techniques to separate deployment from release.

Figure 5: Help Content for the CI Metric

Here the tool can guide teams to know good content and suggestions rather than forcing each team on a path of discovery and hoping each finds good information.

Gamification can be an effective tool to help jump-start improvement. It cannot be the only strategy but should be seen as one more tool in the kit. It helps the early and middle adopters as long as there is no sense that the metrics are being used to judge team performance. Over communicating the intent and walking the talk is required.

Guardrail Metrics

Deliberate gamification can be an effective aid to help us scale improvement. However, as soon as we measure any human activity and make those results visible, we've gamified the system—we just may not

be doing it deliberately. We need to be aware of that. We shouldn't measure any human activity without a tester's mindset. "What could possibly go wrong? How can we inspect that anything has?"

For example, if we measure defect rates, the natural behavior is to aim for zero defects and to add layers of verification steps to achieve this. Something fails, add more verification steps. This is good until it isn't. When the verification steps begin adding too much time or require too much effort, then other metrics will suffer. It will take us longer to deliver fixes using our standard process, and we cannot deliver change as frequently. We need to combine the defect rate with delivery frequency and delivery cycle time. If we are working to improve one, the others will tell us when we are headed in the wrong direction. Metrics are only signals about our goals, after all. They do not stand alone.

Investment Is Required

I should be clear—teams are delivering the best they can in their current environment. We cannot expect teams to improve flow unless their environment improves, and part of that improvement is investing in their success. Tools help, but more important is training teams on the teamwork and practices required to become more effective in their context. Additionally, are teams incentivized to behave as teams? Are we focusing on individual achievement or team outcomes? Growing stable teams that are focused on how to best deliver the capabilities they own is a critical investment in our future success.

Only the Tip of the Iceberg

The four key metrics DORA used to correlate behaviors of high-performing organizations are a small subset of the metrics recommended in the book *Accelerate*. They also represent only one aspect of the health of the system: the outcomes of improving our quality processes and removing waste. This improvement of efficiency is important, because we cannot be high-performing otherwise. It's relatively easy to measure efficiency, and things that are easy to measure are where most people will focus their attention. Too much focus on one signal will hurt our goals. We need balanced signals to guard against that.

Effectiveness

Are our customers happy? Are we improving our bottom line? Are we achieving the broader strategy of our organization? If we drive efficiency and these aren't true, then we will efficiently go bankrupt.

Customer Outcomes

Integrating code frequently, delivering it frequently, and verifying that it matches our requirements doesn't matter if it's not actually useful to the end user. Do they like what we are delivering? Are they using it? Is it stable and available? The four key metrics track how efficiently we can deliver and the effectiveness of some of our quality processes but tell us nothing about the effectiveness of the ideas we are delivering. Only customer-focused metrics can do that.

Objectives and Key Results (OKRs)

OKRs track our progress toward business goals for long-, medium-, and short-term objectives. If we have no goals and no objective to achieve them, why bother becoming more efficient?

Sustainability

Is the cost of keeping customers happy higher than we can afford? What about our organization's health? Can we sustainably deliver at the pace we've set without burning people out? It is relatively easy to become more efficient, do it effectively, and have our signals hide the fact that people are working long hours, after hours, and on weekends to do it. Measuring and preventing this acts to make sure our efficiency metrics are telling the truth.

Flow

If it takes us too long to convert an idea or a customer request into delivered work, then the work may be irrelevant when we deliver. By tracking total lead time and mapping where that time is spent, we can uncover larger organizational constraints we can improve. By keeping our WIP (work in progress) under control, we can minimize defects and delays caused by context switching and place focus on getting the highest priority thing done.

Culture

We need indicators for our organizational culture. It's easy to get false improvement signals from flow, CD, and CI metrics by simply encouraging people to work in unsustainable ways that burn them out. We need to track feedback from the teams. How many deliveries are made after hours? How many extra hours are being worked? What about turnover? Westrum surveys are also highly recommended. To achieve actual improvement we need to be more efficient, more effective, and do it sustainably with happier coworkers. Establishing the right culture is important before we attempt to use metrics as part of our improvement process.

Trust is the first impediment to overcome. If we lack organizational trust, then any improvement efforts will stall. Trying to improve things means trying to do new things that may not work as we expected. If we lack trust, then trying new things is risky. If we have trust, then we feel free to propose new ideas or "pull the Andon cord" when we are seeing things going wrong. Don't assume trust. We need to intentionally instill it and get feedback on how comfortable people are to try.

All of the metrics rest on a foundation of a shared mission. We discussed how being more efficient is pointless if we are delivering the wrong things. However, how do we know what the right things are? Our organization should have a clear purpose. What is our North Star? Our goals and OKRs should be aligned to that North Star. Every day, everyone should be able to tie their current work to that shared mission and understand the value they are adding.

If we have trust and a shared mission, then we need to make sure we foster a culture of learning and improving. By using our daily work and improving how we do our daily work, we are continuously

learning and coming up with better ideas. This prevents the stagnation that will eventually make our organization irrelevant. We should never be doing it the way we were before.

Figure 6: Measurable Goals to Help Drive Improvement

(This image is provided under a creative commons license:CC BY 4.0 - Bryan Finster - Commercial use with attribution)

Conclusion

DORA's study of how organizations behave relative to their outcomes is an important work. We should not, however, confuse the metrics we need to use to *monitor* our improvement efforts with the purpose behind DORA's four key metrics and attempt to use those four metrics as yet another silver-bullet transformation framework.

Improving throughput and stability is the engine for organizational improvement. We need to measure this with a balanced set of metrics that prevent unintended outcomes, and we must continuously inspect and adapt to ensure we are on track.

We need to keep metrics visible and part of the conversation, but we must also invest in educating everyone on what the metrics are telling us and how to respond. We should not

unleash a dashboard and assume it will be used appropriately. Good intent is insufficient to prevent the damage that will occur. We also need to make sure that we are measuring at the right granularity.

Measuring the behaviors of individuals will lead to people acting as individuals. It requires teamwork to be high-performing.

We should never use delivery metrics to compare teams. Those metrics are the team's metrics, and each team has its own context. We need to establish a culture of improvement as an expected business deliverable along with any other feature and provide teams with the observability and training they need to deliver that feature.

Product development is a complex interaction of people, processes, and products. There are no simple metrics. Measuring any human activity is complex, not simple. The act of measuring changes the outcomes. We need to be mindful when we use these tools. We also need to be aware that we cannot measure our way to improvement. We use these tools to monitor and inform the next improvement experiment.

Measure the process, not people. Invest in providing the help people need to improve the process and outcomes. People are our most valuable asset.