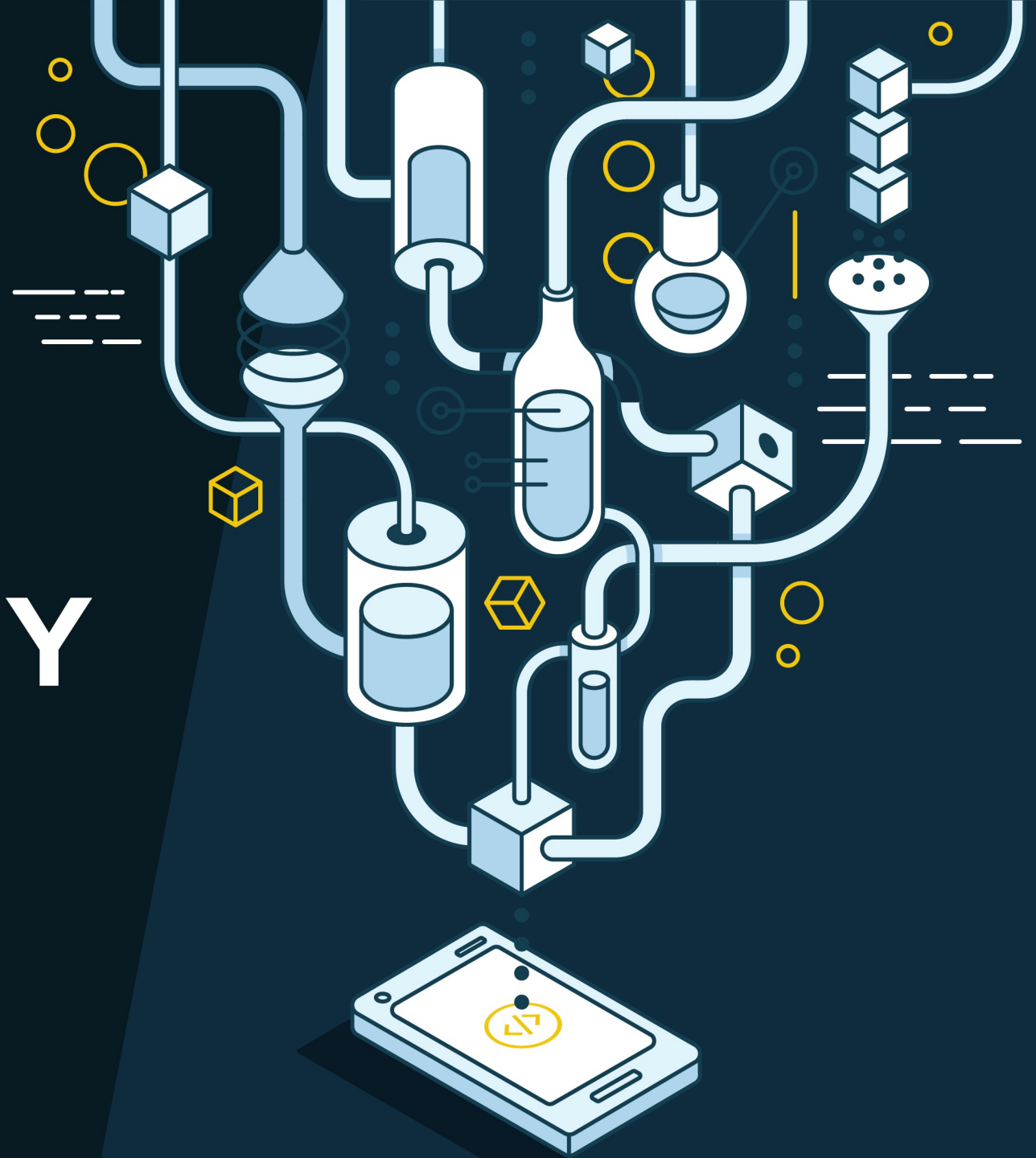




A BRIEF AND PRACTICAL GUIDE TO

DEVELOPER PRODUCTIVITY METRICS



Can developer productivity can be measured?

Yes, but beware:

Backlash from developers 🤔

Developers don't want to go into a performance review to have a conversation that goes like, "your number on this metric is lower than this other person".

Just like any technology, how you use the metrics determines whether they are perceived as "evil" or not.

If you're a manager facing a situation where a developer's productivity has declined, don't focus on using metrics as proof. Chances are, you don't need metrics to find out what's going on in the dev's life that may be affecting their work.

Unhealthy culture ❌

You can't blame developers for believing "when a measure becomes a target, it ceases to become a good measure", if your culture isn't healthy enough.

It's OK to have metrics. It's OK to have targets. It's not OK to weaponize numbers.

For example, if you practice blameless culture, everyone knows the point of a post-mortem is to learn from the incident.

That makes it psychologically safe for anyone to say, "This problem happened and it's because what so-and-so did".

There is such a thing as “bad metrics”.

Example 1: Lines of code

Lines of code written can be misleading. You can spend a couple of days writing only 3 lines of code, but those three lines of code could be terribly important.

Some say that a good developer writes 9 lines of code a day, including future refactoring, bugs, rewrites, etc.

Whether 9 is the right number is beside the point:

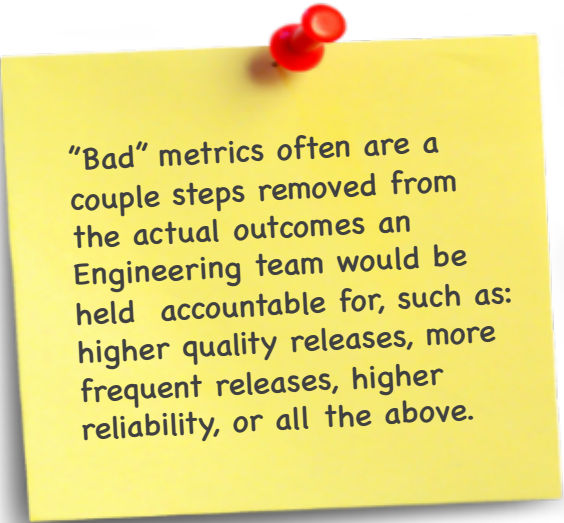
- Lines of code not deployed to production does nothing for end users or business
- Lines of code deployed but of poor quality will actually hurt your team.

Example 2: Story points

Story points can be good in some teams when everything's aligned, but it often can become an exercise of “counting jellybeans”.

The idea behind story points is that the abstraction helps developers stop themselves from equating points to hours. In reality, devs still do this in their heads.

The issue with metrics like story points is it adds a layer of abstraction between the work outputs and the ultimately important outcomes, such as more frequent or higher quality releases.



“Bad” metrics often are a couple steps removed from the actual outcomes an Engineering team would be held accountable for, such as: higher quality releases, more frequent releases, higher reliability, or all the above.

Metrics gone bad: “Business Value Points”.

Annual shenanigans

Don Brown, Sleuth CTO, once worked at a place that used a metric called “business value points” or BVPs.

BVPs are basically like story points, but for annual planning purposes.

The Engineering team would be given a target to deliver more BVPs compared to prior period, e.g., 20% more than last year.

The target brought all kinds of gamesmanships. Devs would pick easier tasks and assign them higher BVPs to make it easier to hit the target.

Short term-isms

The worst part of BVPs is that it disincentivizes the team from working on refactoring and performance tasks.

The metric became yet another way to get the team to ship more and more features, but as they did that, the backlog of tech debt grew, and developing features took longer and longer.

Ultimately stability got worse, resulting in unhappy customers. Unfortunately, Support didn’t have enough BVPs to give – especially compared to revenue-facing functions – so they couldn’t influence the priorities, and things never changed for the better.

Example of good metrics: Accelerate metrics.

Deploy Frequency

How often the team deploys to production.

This metric is good because to improve on the metric, the team will need to adopt best practices, including:

- Keeping codebase in deployable state
- Deploying in small batches
- Having a well-defined deployment process
- Investing in testing and deployment automation

Change Lead Time

How long it takes from first commit to production deploy.

This metric is good because it ensures enough attention is paid to finding and removing potential bottlenecks in the development lifecycle.

It also reinforces a key DevOps tenet: that the definition of “done” is not when the code is written & merged, but when the release has been deployed to production.

Change Failure Rate

How often deploys cause failure in production.

This metric is good because it helps the team focus on release quality.

In fact, the definition of “done” should be when the release has been deployed to production and it is healthy (zero or acceptable negative impact) – however that’s defined for your team.

MTTR

How long it takes to recover from a failure in production.

This metric is good because it requires an entire team effort across Dev and Ops.

Teams would need to have good observability, incident management process, and a blameless culture to do well on MTTR.

Study shows these metrics are highly correlated with software delivery performance. To learn more, check out the [Accelerate State of DevOps 2019 report](#) from DORA.

What makes a metric “good”.

Eyes on the prize 📦

What does a productive Engineering team do? They deliver. Being more productive is about delivering better releases faster.

The Accelerate metrics are good metrics because if your team is deploying more often, making changes faster, causing failures less often, and resolving failures faster when they do occur – there could be no argument that the team is productive.

Lines of code, on the other hand, wouldn't be a good metric because more lines of code don't necessarily mean better releases faster.

No "I" in team

Notice that all four Accelerate metrics are focused not on an individual's work outputs, but rather on how well the entire team works together.

Good metrics promote teamwork, not discourage them.

Offense is the best defense.

Asking “how can we add processes to slow down and make sure our 3-month release has no bugs and is “safe” to ship” is asking how to play better defense.

Good metrics help the team play better offense – as in, “how do we keep the code flowing? What can we do to ship smaller changes and quick fixes if we make a mistake?”

Dylan and Don's fav metric: Deploy Frequency



Dylan

High deploy frequency implies that a lot of things are going well.

It implies the team has a healthy enough culture where developers are empowered and feel safe to make changes.

It also implies the team likely has adopted best practices like small batch sizes.

When you're shipping frequently and you have very small batch sizes, it becomes a non-event to deploy to production.

The surface area you can break things with is smaller with small batches, too.



Don

To deploy frequently, teams would need to employ automation and tracking in their deployment process.

A good deployment process can reduce the stress on developers during deploys.

It means developers don't have to set aside hours after a deploy just in case something goes wrong.

And even if there were so many changes in the deploy, figuring out which one caused the issue – and rolling it back if necessary – should be quick and easy.

Enjoyed this guide? It's from a popular episode of Dev Matters on Sleuth TV. Watch the full 50-minute video 📌

*Dylan Etkin,
Sleuth CEO*



*Don Brown,
Sleuth CTO, and host
of mrdonbrown on
Twitch*

<https://www.youtube.com/watch?v=U4Z0HfRYTHI>