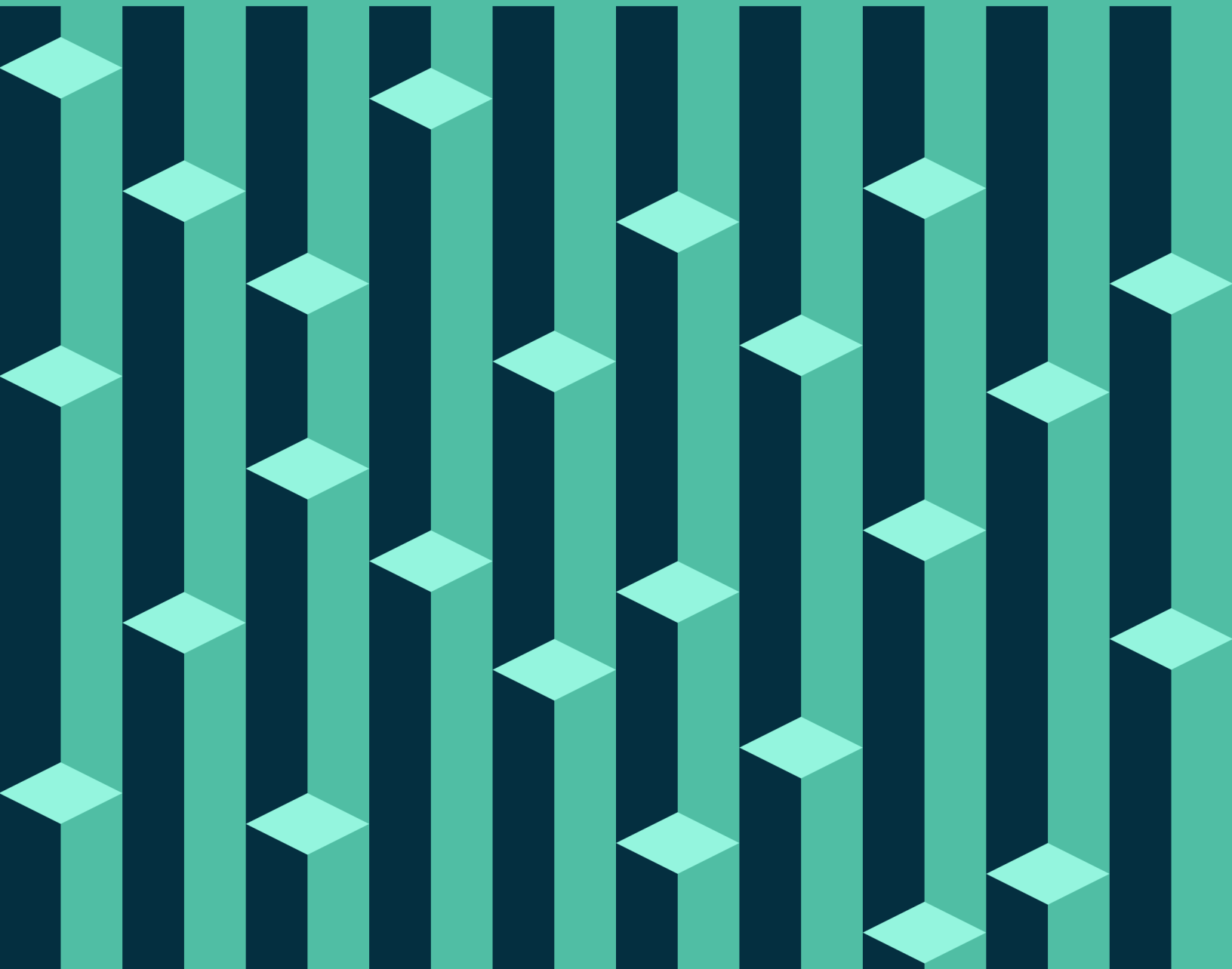
GitPrime

A Data Driven Approach to Leading Software Teams



Introduction

Whether you're the VP of Engineering at a large enterprise or a startup CEO, one thing reigns supreme: Shipping product is king. And typically the stress of leading a team toward that goal boils down to a few key questions:

- **What is my engineering team doing right now?**
- **Why is the release running late?**
- **How can we do better?**

Just like you, we've been haunted by those same nagging questions. Our mission is to build a tool that finally answers those seemingly impossible questions, to help software teams reach their highest potential.

We've been writing about our lessons and experiences on the GitPrime blog, and wanted to now share with you: a data-driven approach to leading software teams.

Here's to making your software engineering team's work visible, quantifiable, and actionable.

—The GitPrime team

First, a few foundational points

Engineers are valuable

The reality of our era is that software engineers are in such high demand that negative-unemployment is the norm. We're all lucky to have every engineer that we do! The real challenge is providing guidance, timely feedback and learning where team members are most engaged.

The data is there, ready to speak

Software engineering can be one of the most quantifiable activities in any organization, yet this powerful asset sits largely ignored. Mining version control data offers realtime insights to make teams stronger and reach their highest potential.

Forward progress is not the same as effort

Using flat lines of code as proxy for productivity is a deeply flawed approach — engineering is just more subtle than that. What matters are more subtle metrics: how long that code sticks around, the manner in which it's introduced into the codebase, and why it's being written in the first place.

Peter Drucker was Right

"What gets measured gets managed," and we consistently see that the mere act of paying attention to what's happening in engineering leads to tangible improvement. Knowing which metrics matter for your team and consistently measuring them is key.

Agile is great, but it's lacks hard data

Agile & Scrum methodologies are perfect for establishing targets and setting goals. While burndown charts give some visibility here, agile is still limited to a narrative view of the team's success, as story points & tickets have no inherent connection to the actual engineering work.

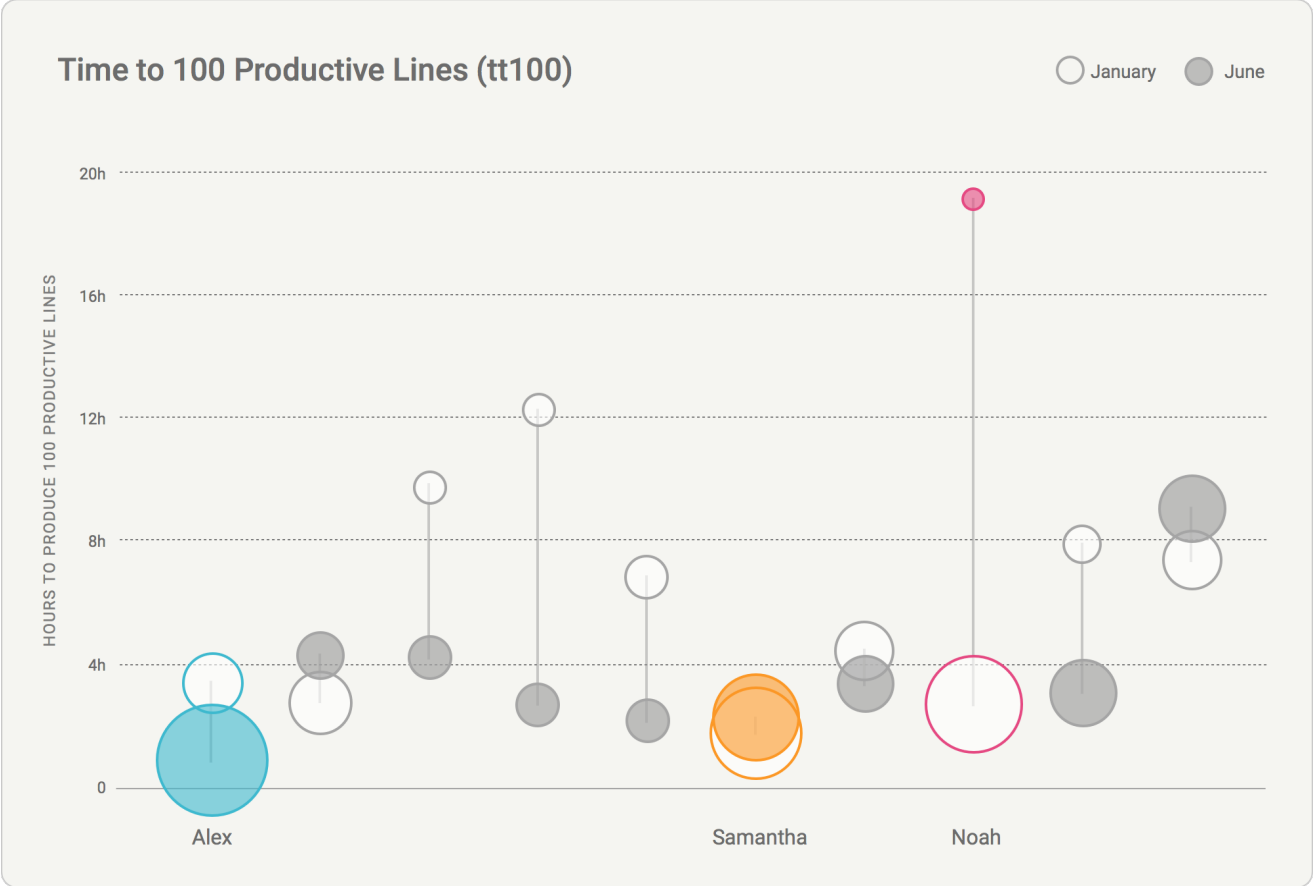
Data-driven insights are launching points for conversation

Building a high-performing engineering culture starts with raising the visibility on positive behavior, knowing when to give precise praise for heroic feats, and knowing how to give specific direction when someone is struggling. There's no replacement for a technical manager paying attention to the big picture and having timely conversations with the team.

Let's jump in.

Not all work is created equal

Checking in huge amounts of code is a nice vanity metric, but only the code that sticks around actually moves the project forward. One of the most powerful ways to measure developers' productivity is their "Time to 100 Productive Lines" (***tt100***). Averaged out over time, this metric gives an accurate sense of how well the team is doing.



ON A ROLL

Between January and June, Alex improved from an average of ~3.5 hours to <1 hour for the amount of time required to deliver 100 lines of productive code (***tt100 Productive***). Not only is he delivering productive code more quickly, but the larger diameter circle shows he is taking on more work.

It looks like Alex is learning and thriving. Congrats are in order.

STEADY AS EVER

Samantha is the developer you can count on to give you a bunch of productive code quickly. In the past 6 months, there is little movement in volume or time that it takes her to commit 100 lines of productive code. She's one of the team's cornerstones.

It's probably time to give Samantha some additional critical responsibilities.

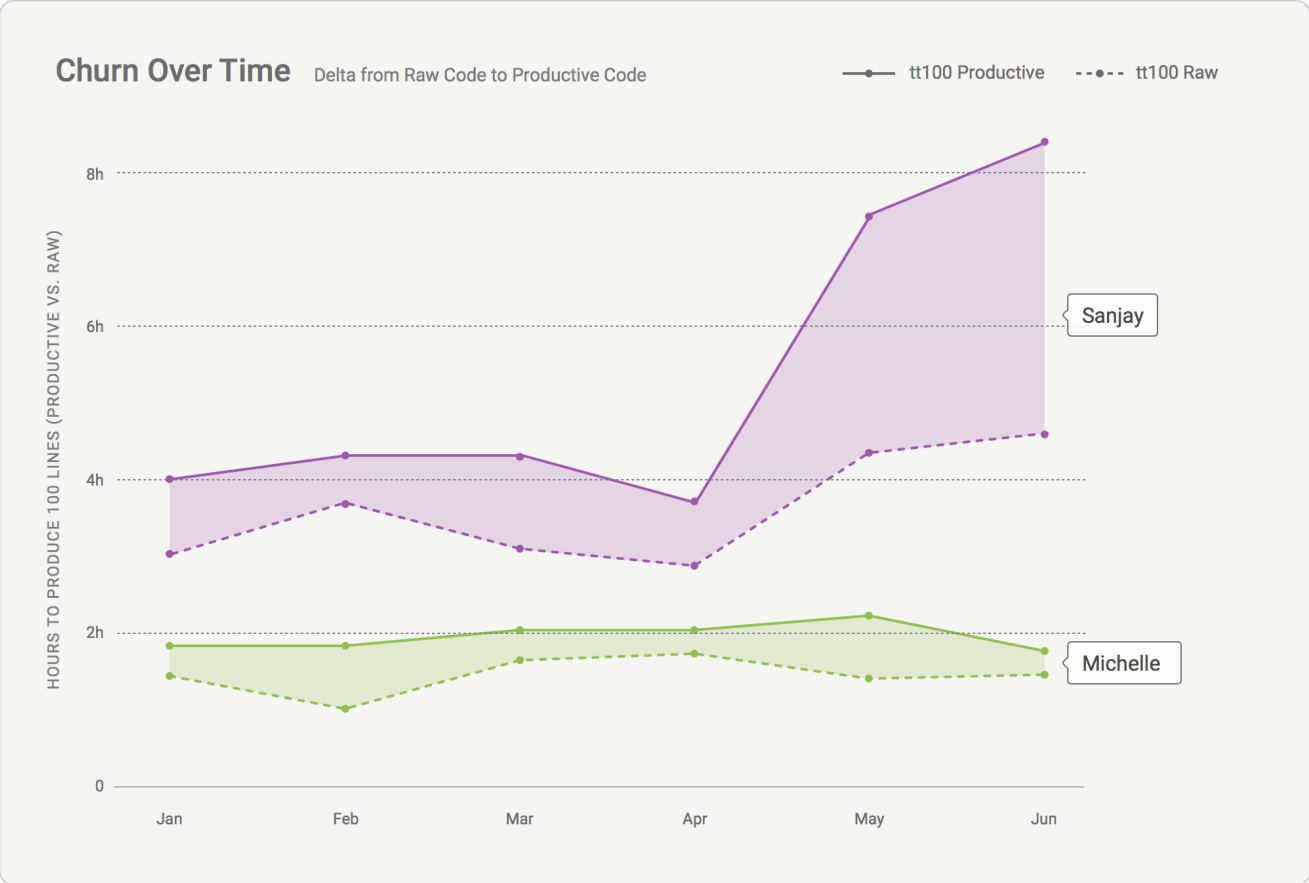
SUDDENLY STRUGGLING

It sure looks like Noah is taking a long time to write productive code (nearly 20 hours!) and that small circle means it's a relatively small volume. While tempting to think he is simply underperforming, consider his pattern in January — he was efficient and prolific.

Noah might be spinning his wheels and in need of a new challenge.

Success is not always “more work”

This is why tracking ***Churn*** is important. Think of churn as the waste that happens when an engineer rewrites their own recent code. The root cause could be any number of factors. Is someone stuck on a tough problem and grinding their gears? Polishing a feature to death at the cost of shipping it? Flying under the radar and working on pet projects? Under-engaged and running in circles because of indecision from the product team? Spotting ***Churn*** is critical to a healthy engineering culture.



WARNING FLAG

In the four months from January to April, Sanjay demonstrated a consistent ~3 hours for 100 lines of raw code, ~4 hours for 100 lines of Productive code, and a stable amount of ***Churn***. In May, his ***tt100 Productive*** doubled and ***Churn*** soared. June looks similar, and is trending the wrong direction, so his work pattern in May is not an isolated event.

It’s time to address the change in behavior with Sanjay and discuss what the root cause may be.

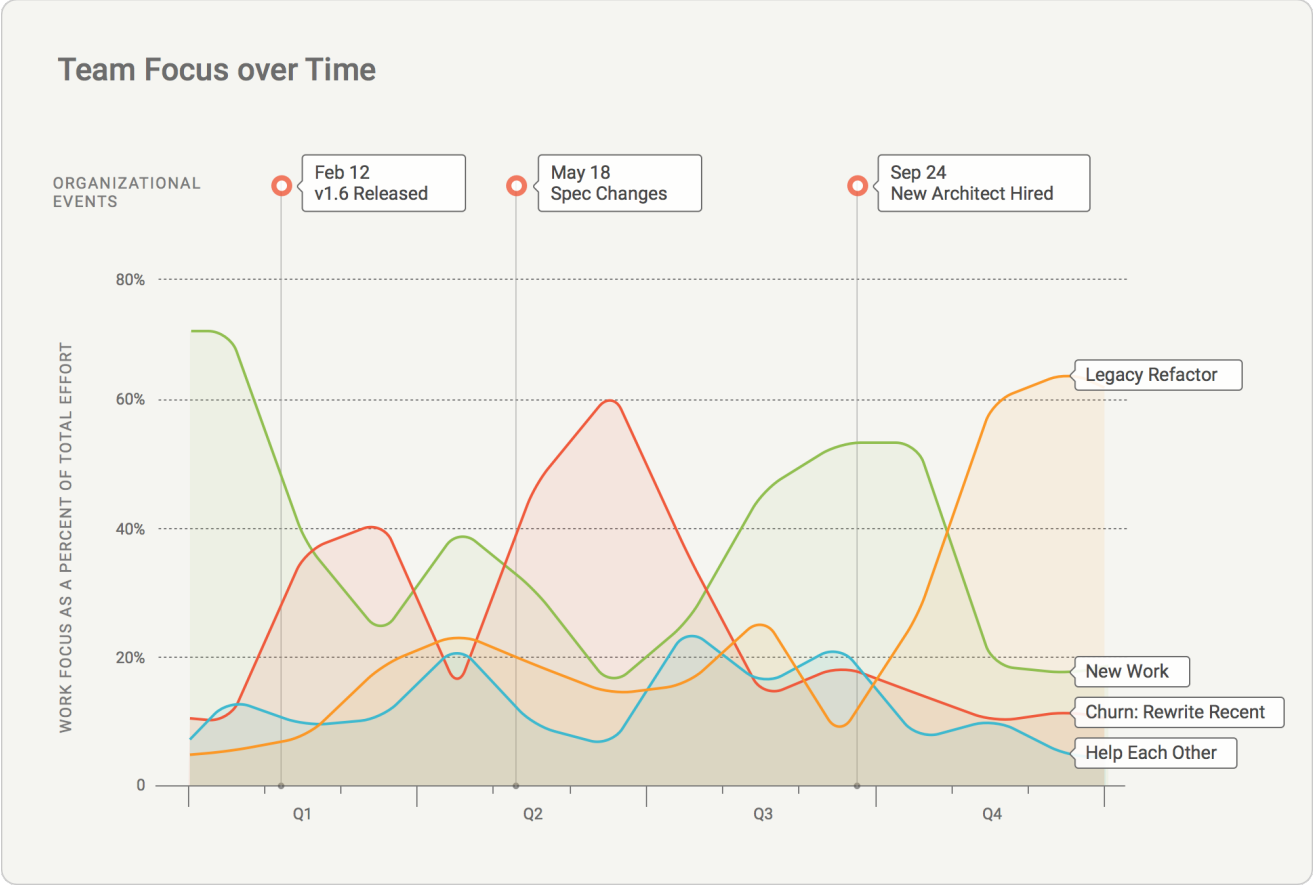
IN THE ZONE

In the same time period, Michelle efficiently wrote 100 lines of Raw Code and her time to 100 Productive lines (***tt100 Productive***) hovered steady around 2 hours. The delta between her Raw and Productive—the area between the two lines—is consistently small. She’s clearly moving the project forward.

Michelle is on a roll. Probably best to stay out of her way and let her continue with what is working.

How do organizational events affect productivity?

Instinctively we know that releases, spec changes and personnel changes play a dramatic role in the productivity of teams. A data-driven approach allows decision makers to access these correlations in post-mortem analysis as an aid for future success.



A In early Q1, we see ~70% of the team's attention went toward ***New Work*** as they made the frantic push toward release v1.6. Shortly after release, there is elevated ***Churn*** as the team scrambled a bit to rewrite issues that surfaced after it went live.

You can demonstrate the impact of rushing through a release without ample QA testing.

B By early Q2 things are settling down for the team with the focus put back on New Work. In May, the distinctive spike in ***Churn*** follows closely on the heels of when the Product Team wavered in direction and handed over a spec that gutted the efforts of the last few months.

Mid-term requirements changes can have dramatic consequences on team performance.

C By late Q3 the team recovers from this hiccup and finds a groove of ***New Work***. In September, a new lead Architect joins the team. After getting settled, she pinpoints the root of some nagging performance issues and much of the team's focus is devoted to ***Legacy Refactor*** work and environmental changes.

A clear & objective answer to the question "What was the team doing in Q4?"

This, and much more

The concepts and metrics in this report hint at the possibilities and power of the data-driven approach to leading software teams. GitPrime is the pioneer in this field, and has launched a SaaS platform to help teams perform at the highest level.

Winning teams are built on results not superstars

In the absence of data, teams are built on personality and skill tests. With statistical insight, software teams can use the same kinds of feedback loops that other industries have been using for years to build high-performing teams.

Measure engagement & activity

Understanding team engagement and the patterns that naturally occur in software engineering is the first step in building a team that promotes individual success and consistently delivers.

Identify work focus

Get a realtime view on how engineering is being deployed. Is it mostly new work? Is the team spending an inordinate amount of time paying down technical debt? Does one member of the team consistently write code that needs to be refactored before it's ready for production?

Target risk patterns

There is alway risk in new code, but with GitPrime, this is visible. The level of risk associated with every commit is clearly called out, making it easy to deploy QA resources or make sure this work has received a serious review.

Pay attention to what matters

Time is our most valuable asset. Knowing how to best spend that time is critical. With GitPrime, team leads know exactly where to focus their effort in code reviews and coaching.
