17 Metrics for Modern Dev Leaders

Measuring the right indicators will help you accelerate delivery, maintain positive culture, and translate dev activity to business value

LINEARB

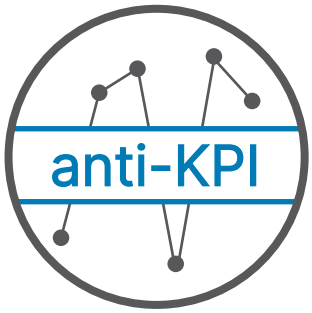
Motivation to be data-driven

As dev leaders, we spend most of our time translating between two groups of people in two parallel universes. Most CEOs and board members come from the business side (sales, marketing, finance) and while they enjoy the outcomes of engineering, they don't fully understand how we work. At the same time, many engineers don't fully understand the business side of the organization. This is the background of most dev leaders.

Driven to help our teams succeed and bridge the gap between engineering and the business, many of us have turned to data and metrics. But what metrics truly help us accelerate delivery and correlate to metrics the business cares about (revenue, leads, retention)?

Velocity is the most dangerous metric for dev teams. Click here to read more about why this "anti-KPI" detroys dev culture.

We would all agree that measuring the right things is important. But some software development teams still use legacy measurements that actually stop them from getting better. We call these metrics anti-KPIs. Some of the most common software development anti-KPIs include velocity (story points completed), lines of code, code commits, Jira tickets completed.



What do these anti-KPIs all have in common? First, they may on the surface indicate that your contributors are busy but they don't actually help you figure out if you're providing value to customers. Second, they encourage the wrong behavior. Is writing more code good? Maybe. Or maybe it is a sign of inefficiency. Do completed Jira tasks show productivity? Most developers would argue that what happens in Jira has little to do with the "real work" of writing code. Third, and worst of all, these anti-KPIs can easily be gamed. Your contributors that are struggling might be tempted to inflate these anti-KPI numbers (which they can do easily) which actually blocks you from giving them the help they need.

So what should modern dev leaders measure? Keep reading to discover 17 metrics that matter for dev leaders and how to use them.

What to Measure?

At LinearB we think of operational excellence in software engineering as the pursuit of predictably delivering projects, with high quality, maintaining efficient working hours, with happy contributors and teams, continuously improving. That's a lot :-) But we think it's possible.

It can seem overwhelming - especially if you're not sure how you're doing against all of those dimensions and what steps can be taken to improve. We worked backwards from the goal of delighting our customers and being operationally excellent and came up with 12 key performance indicators. Some you're probably already measuring. Some may be new to you.

We look at three categories of KPIs:

- Work quality
- Delivery pipeline
- Investment profile

And we measure those categories across two dimensions:

- Iterations
- Teams

At LinearB we think of operational excellence in software engineering as the pursuit of predictably delivering projects, with high quality, maintaining efficient working hours, with happy contributors and teams, continuously improving.

DELIVERY PIPELINE

Delivery matters! An end to end view of your pipeline with metrics like time to merge, time to deploy, deploy frequency, and cycle time uncovers bottlenecks and accelerates your delivery rate.

INVESTMENT PROFILE

Are your teams working on bugs or new value? LinearB is the first platform to combine git and project insights to provide unprecedented visibility, even if your record tracking isn't pristine.

QUALITY

See what holds your teams back. Understand where you suffer from high code churn and large amounts of rework. Use metrics and set goals to adopt a continuous refactoring culture and shorten lead times for changes.

ITERATION

Control and predictably complete your iteration commitments. Reduce story churn, get alerted on stuck PRs, and get an automated data-driven retrospective report delivered right to your inbox.

TEAM

Happy and engaged teams deliver more. Efficient teams avoid burnout. Empower your teams with access to the metrics they need to operate successfully and autonomously.

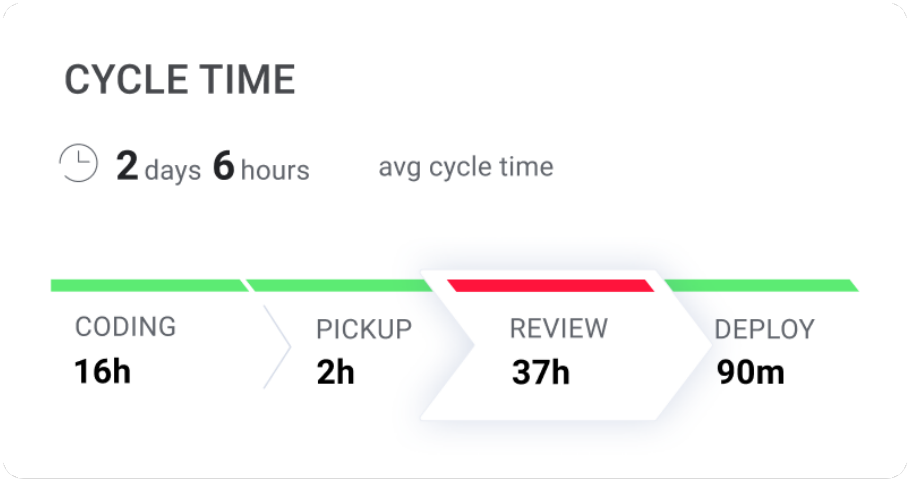
Measure Your Software Delivery Pipeline

Efficient delivery pipelines lead to predictable value delivery, happy developers, happy product owners, and happy customers. Frustrations arise with inefficient pipelines.

Your software delivery pipeline is what enables your team to deliver code to production. It includes all of the phases from "work requested" all the way through release to production and validation. Some of the common phases include development work beginning, pull request open, pull request merge, and release to prod.

Efficient delivery pipelines lead to predictable value delivery, happy developers, happy product owners, and happy customers. Frustrations arise with inefficient pipelines. These situations can happen when code is merged and ready to be released but "the release is not until next week" or when a developer has opened a pull request but it takes days to receive a review or when a story has been sitting in the backlog for weeks.

Measuring the stages of your delivery pipeline allows for bottleneck detection. This provides a high leverage point to increase your delivery performance because it impacts all teams and contributors



ESSENTIAL KPIs TO MEASURE:

Cycle Time: The amount of time from work started until work finished.

Why it matters: Cycle time is the #1 indicator of your speed to value and efficiency ratio.

Note: Many of you are already measuring (or trying to measure) cycle time. But what do you do when you think your cycle time is too high? The next four KPIs help you diagnose where to spend your time to drive cycle time down.

Cycle time was among the most impacted metrics when dev teams shifted remote. Click here to download the report to learn why and get 10 other insights.

Deployment Frequency: The number of releases per day.

Why it matters: This is a strong indicator of how much value your team is capable of getting into the hands of customers. Even if you have an efficient pipeline, if your deployment frequency is low, you may not be delivering enough value.

Lead time: The amount of time from work requested until release.

Why it matters: Start time + Cycle time = Lead time. Once you understand your cycle time, looking at start time can tell you how long it takes on average to get through your product management process and backlog. If you have high start times it could be an indication you need to hire more contributors or change the way you set expectations with customers and your sales team.

Time to release: The amount of time from Pull Request Merged to Production Release

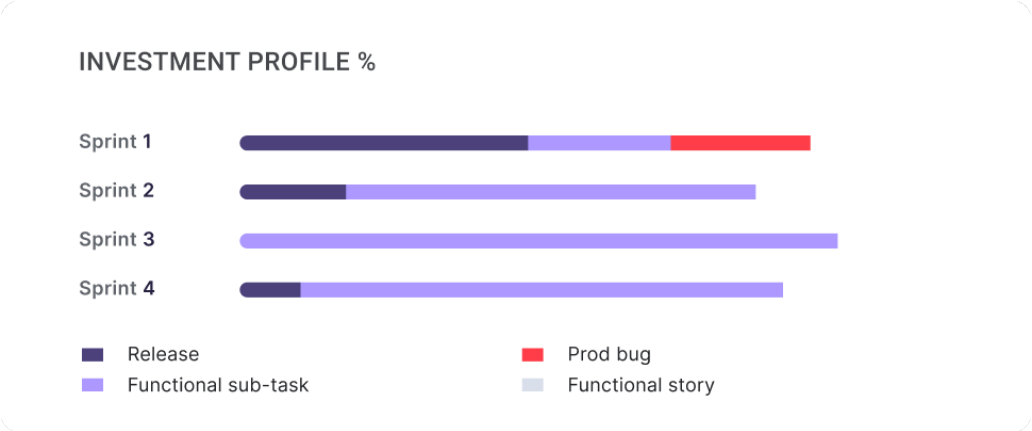
Why it matters: Your developers may have finished their job but your customers may not be getting the value. If time to release is high, it could mean you need to invest more in continuous deployment (CD) or that you have an opportunity to move to a micro-service architecture.

Time to merge: The amount of time from first Commit to PR Merged

Why it matters: This is a key indicator of your cycle time. It can show the efficiency of your pipeline and it affects all members of your team. If time to merge is high it could be an indication that you're lacking automation or your team needs additional coaching or process or an indicator your developers are not getting enough detail from product management.

Review request waiting time or pickup time: The amount of time it takes from the pull request submitted until review begins.

Why it matters: Efficient teams have a low pickup time. The less time PRs spend waiting fo review, the faster they are released. This metric is important for all dev teams, but is even more critical for remote dev teams.



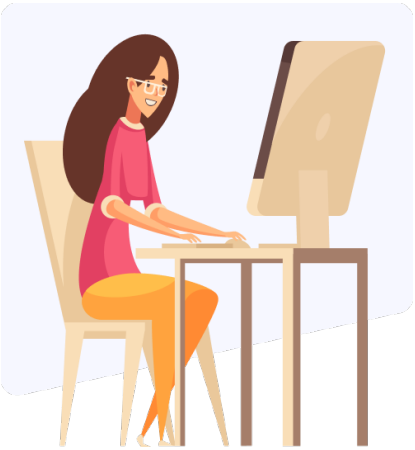
Measuring and tracking your investment profile puts you back in control. It allows you to determine if your actual investment areas match your expected investment areas.

Measure Your Investment Profile

The most valuable asset that your organization possesses is your people's time. It is a scarce and limited resource. There are many forces pulling at your team's time. Your CEO wants to deliver new value to customers, your engineers want non functional investment, the support team wants to fix bugs, and your sales team brings customer commitments. Lacking visibility into where your team is actually spending time makes balancing all of these forces very difficult.

Your investment profile is a data-driven representation of the types of work in which your team is spending effort. The work types typically include, but are not limited to, stories, non-functional tasks, and bugs.

Measuring and tracking your investment profile puts you back in control. It allows you to determine if your actual investment areas match your expected investment areas. It also allows you to be in the driver's seat when interacting with stakeholders like your CEO or Product lead.



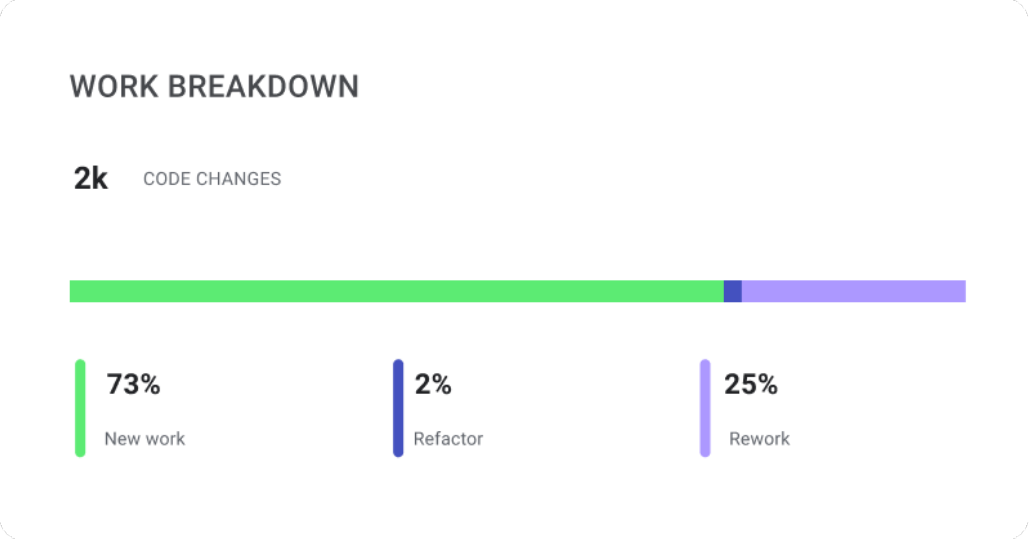
ESSENTIAL KPIs TO MEASURE:

Story to Bug Ratio: The ratio of completed stories to completed bug.

Why it matters: You probably know how many production bugs you have but do you know the effect it has on your customer-facing work? Your contributors certainly know when they feel like they are spending too much time on bugs. Quantifying the impact can help you decide if you have a bigger issue to investigate.

Support & Sales Issues: The percentage of work dedicated to one-off requests coming from the support or sales team.

Why it matters: If you build software for large enterprises, developing based on support and sales team requirements may be a great use of resources. If not, if this ratio of work is high, it could be an indication of planning issues.



The ability of your team leaders to drive delivery through their teams ultimately results in your organization's failure or success.

Measure Your Work Quality

Most teams have experienced the situation where low quality leads to missed delivery dates, iteration interruptions, long hours, unhappy customers, and a frustrated engineering organization. On the flip side, high quality leads to predictable delivery, efficient work hours, happy customers, and a happy engineering organization.

There are many different metrics that you could measure as an engineering leader. Some of the classics range from test coverage to service uptime. While these metrics are great, we have found that there are a few metrics that really help to measure delivery predictability.

ESSENTIAL KPIs TO MEASURE:

High risk work: Branches with large change and high rework or refactored work.

Why it matters: The general rule on most dev teams is that the larger the change, the higher the risk (i.e. branches with 300 lines of code are riskier than small branches). Branches with a high percentange of rework and refactored work are also riskier. High risk work is a leading indicator of quality.

Code Rework: Percentage of recently delivered code your team is already rewriting.

Why it matters: A high rework percentage could mean you have a training issue, you're rushing the process, your review process is lacking or you have a breakdown in communication with product management.

Bugs Found in Prod: The number of bugs found in production per time period.

Why it matters: Measuring bugs found tending across time is a quick way to determine if your quality is improving or decreasing.

PR merge without review: The ratio of pull requests merged without review.

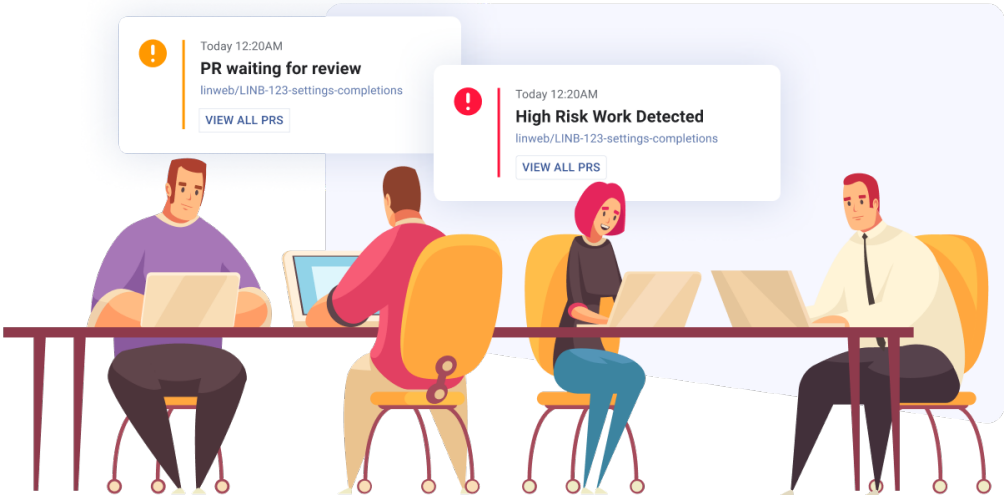
Why it matters: In addition to helping understand potential quality issues, analyzing unreviewed pull requests can sometimes be an indication of a training or culture issue on a given team.

Unreviewed Pull Request Ratio: The ratio of pull requests merged without review.

Why it matters: In addition to helping understand potential quality issues, analyzing unreviewed pull requests can sometimes be an indication of a training or culture issue on a given team.

Review Depth: The average number of comments per pull request review.

Why it matters: This metric is an indication regarding the quality of the review and how thorough reviews are done. Reviews are an important factor for improving code quality and finding quality issues in code before it is merged and deployed.



Measurement Dimensions

We think it's critical to measure these KPIs in a way that's aligned with how developers actually work. For us that means looking at Teams and Iterations.

...the first step is to educate your team on what you're doing and why it will help them get better.

Team Dimension

As your engineering organization scales, it is common to find yourself reliant on your individual engineering teams. The ability of your team leaders to drive delivery through their teams ultimately results in your organization's failure or success. A frustrating situation to be in is when you have a team that is struggling but you do not have measurement visibility to understand why and how to help.

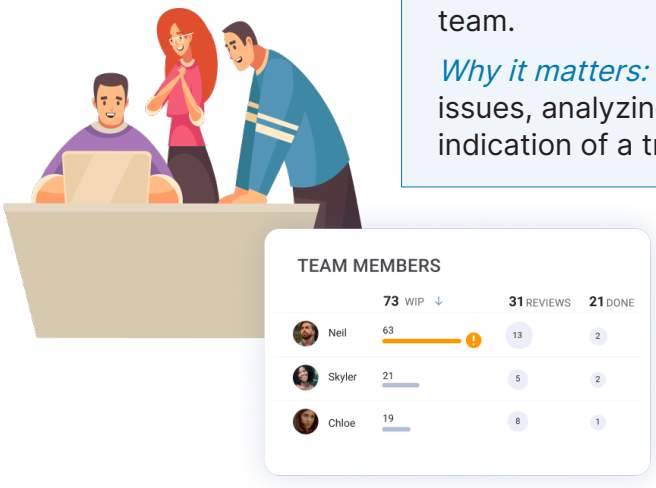
Therefore, it is important that you not only measure all essential KPIs at the organization level, but also at the team level. This allows for three main things:

1. It provides a success framework for your team leader to operate within.
2. It allows you to quickly understand where to focus your time, which teams are operating successfully and which teams are struggling?
3. It allows you to make a positive impact with the struggling team as you will have the measurement visibility to understand what is happening and act.

ESSENTIAL KPIs TO MEASURE:

WIP Amount: How much work in progress assigned to a contributor or team.

Why it matters: In addition to helping understand potential quality issues, analyzing unreviewed pull requests can sometimes be an indication of a training or culture issue on a given team.



Iteration Dimension

The iteration is the delivery pulse of your engineering organization. It represents the construct in which your teams operate. One of the great things about iterations is that they naturally provide proper time periods for measurement.

If you are using the scrum methodology, the sprint length is a great measurement boundary. If you are using Kanban, a one week time frame provides a nice boundary.

ESSENTIAL KPIs TO MEASURE:

Iteration Churn: How many stories come into or out of the iteration once its begun

Why it matters: If you build software for large enterprises, developing based on sales team requirements may be a great use of resources. If not, if your ratio of work coming from sales is high it could be an indication of an issue during the planning phase.

Carryover Work: How many stories carryover from one iteration to the next

Why it matters: If you build software for large enterprises, developing based on sales team requirements may be a great use of resources. If not, if your ratio of work coming from sales is high it could be an indication of an issue during the planning phase.

How do metrics impact your culture?

Many dev leaders worry about how using metrics will impact the culture of their teams. LinearB was founded by dev leaders, for dev leaders. Check out these 2 blog articles to learn the best ways to introduce and use metrics to encourage a positive, productive culture.

How to introduce data-driven culture to your dev team

Can software leaders use metrics without damaging culture?

What do I do with this data?

As you start to measure these KPIs, the first step for you as a leader is to educate your team on what you're doing and why it will help them get better. At LinearB we put these metrics at the center of our business and it helps us deliver value to customers faster with higher quality.

Git, Jira, & Slack coming together to create a new level of context

We connect and analyze data from across your technology stack to unify engineering and product and show you how to improve delivery predictability, speed and quality.