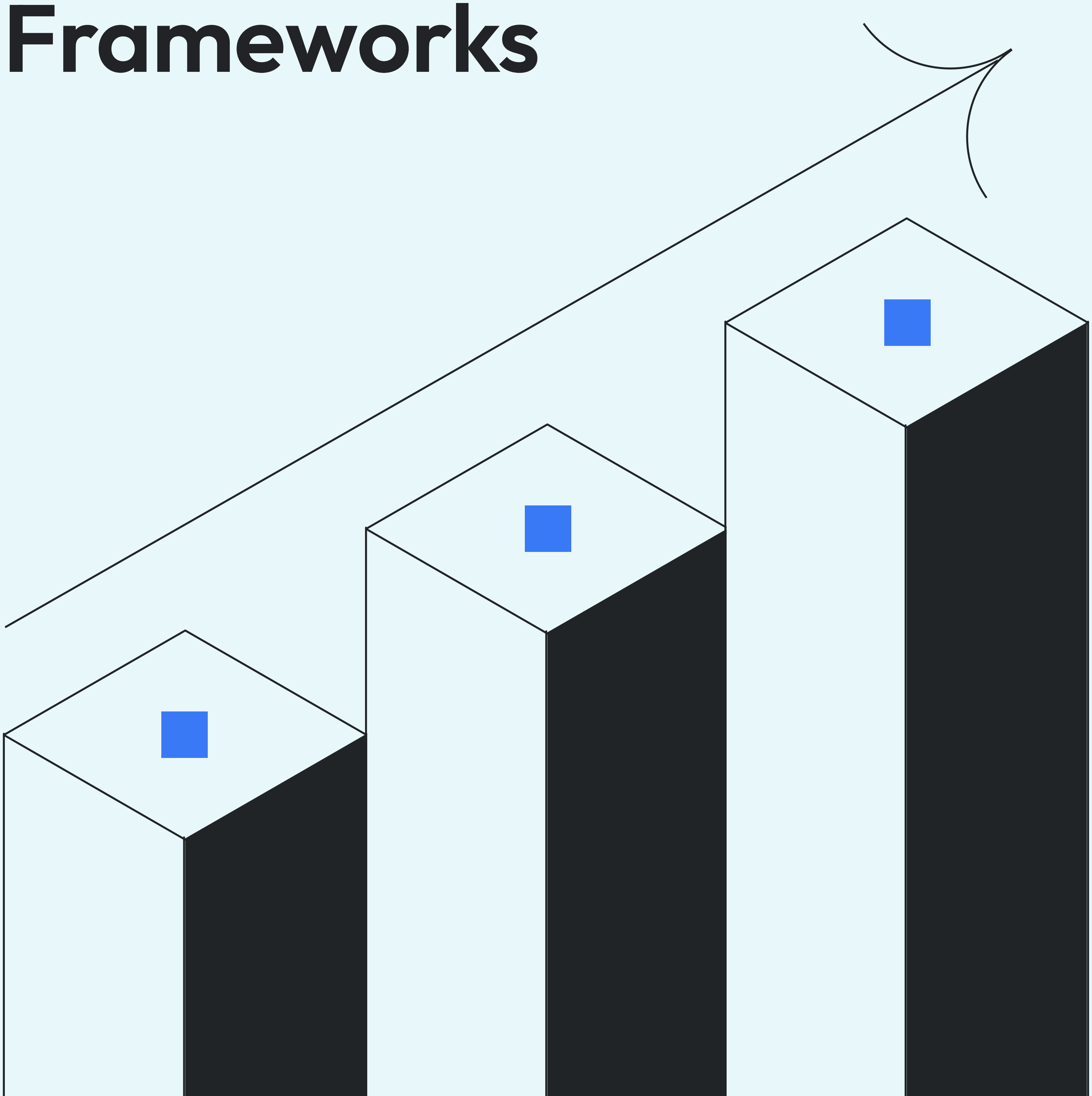
LinearB | Refactoring

Guide

Beyond the DORA Frameworks

A Practical Guide to Data-Driven Engineering Maturity



Another Day, Another DX Framework ...

In recent years, we've seen the rise of research and frameworks to help measure (and improve) engineering productivity.

For the most part, we are talking about fantastic works: DORA & SPACE, among many others, have driven healthy conversations about what "productivity" means, what a good Developer Experience looks like, and how elite teams should aspire to operate.

But as much as we believe this space has done wonders to create awareness about these topics, and is fighting the good fight of making engineering processes more data-driven, I have also found that it suffers from two main issues:

- **Lack of Cohesion**
  By now, there are many frameworks out there, each proposing their own set of KPIs and only covering a small slice of your team's work. E.g. DORA metrics are great, but they're only about delivery. You can't measure the success of your engineering org with a simple set of metrics.
- **Lack of Implementation Details**
  For the most part, these works lack a certain real-world touch. They bring in good theory but are light on implementation: How do you work on these numbers? Who should be involved? What are the ceremonies? What's the cadence?

In other words, many tech leaders find there is a gap between the metrics' world and their teams' reality, and don't know how to bridge it.

In this report, LinearB and the Refactoring community have partnered to create an **in-depth industry survey** to help bridge this gap and uncover two things:

- **The Big Picture**
  Existing frameworks are awesome, we don't need to create new ones. But we need to put these practices on a bigger map, figure out how they relate to each other, and determine their scope and boundaries.
- **Playbooks**
  How teams use data to improve. In real life. With details.

So we asked 30 questions to 350+ engineers and managers, and spent two months working through the answers.

Today we are finally presenting our findings! Here is the agenda:

Scope
Topics, audience size, and goals of this report.

Methodology
How we worked through quantitative and qualitative data.

What Is A Successful Engineering Team?
Working backwards from people's answers.

Successful Practices
Teams that do these things consistently do better than others.

The Role of Metrics
How teams use data to improve.

The Pyramid of Engineering Maturity
Weaving our findings into a simple top-down framework.

Let's dive in!

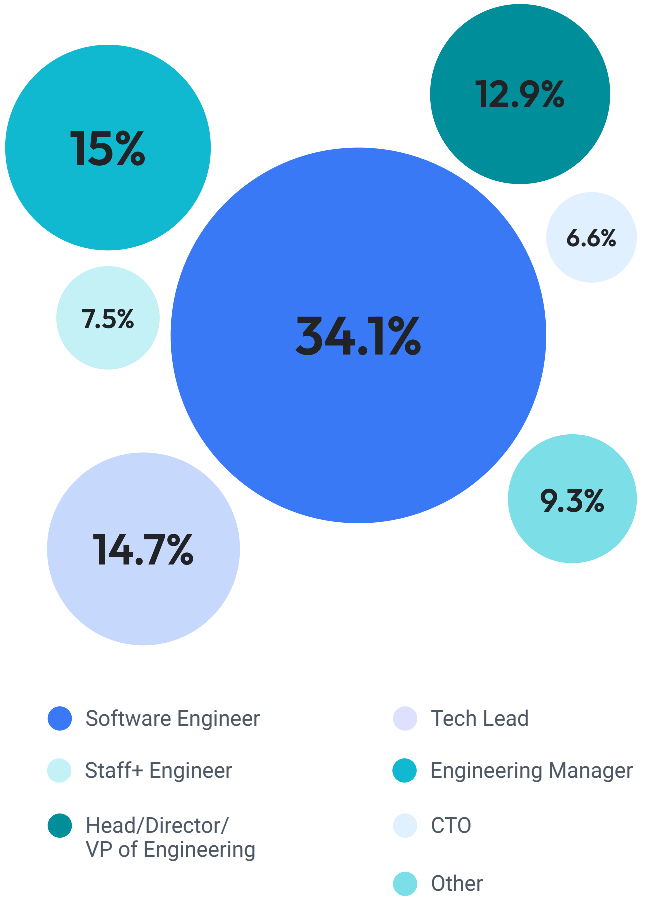
Luca Rossi
Founder of Refactoring

Scope

We got **334 full responses**, from people with a variety of roles in tech:

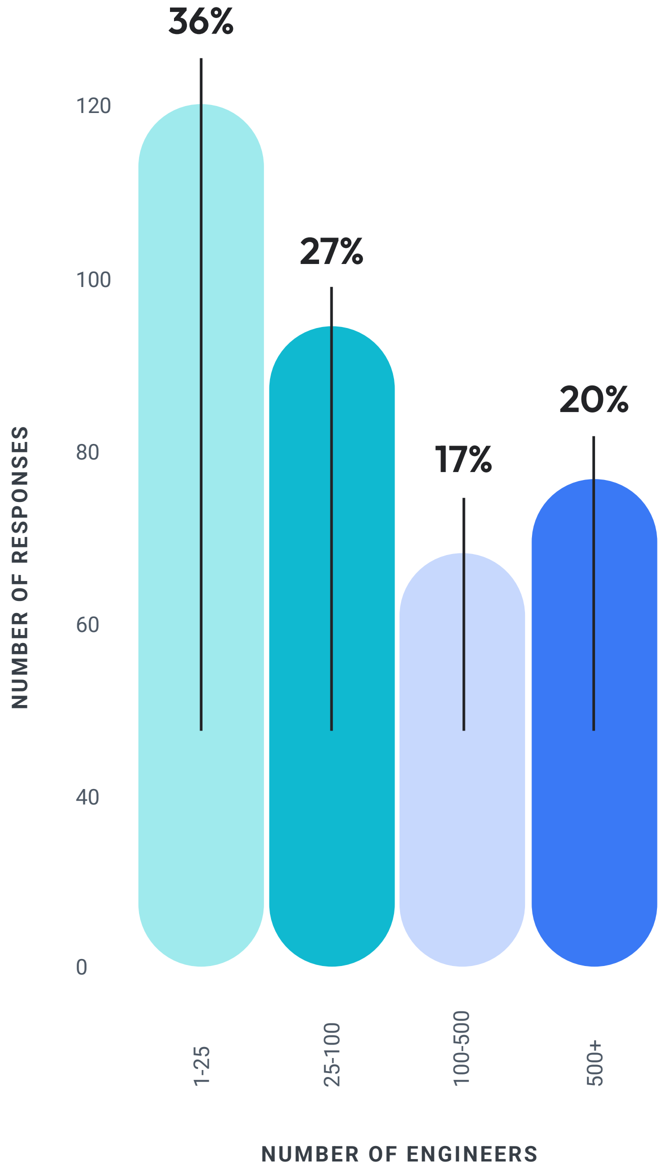
Role

After removing “other” roles outside of software engineering (hello to the four Marketing Managers who answered this survey!), we have **almost perfect parity between IC and Manager roles** (187 vs. 184). That’s by counting Tech Leads in both camps, as the role varies significantly depending on the company.



Company Size

Company size distribution is similar to that of other surveys we have run in the past. About one third of participants work at companies with fewer than 25 engineers:

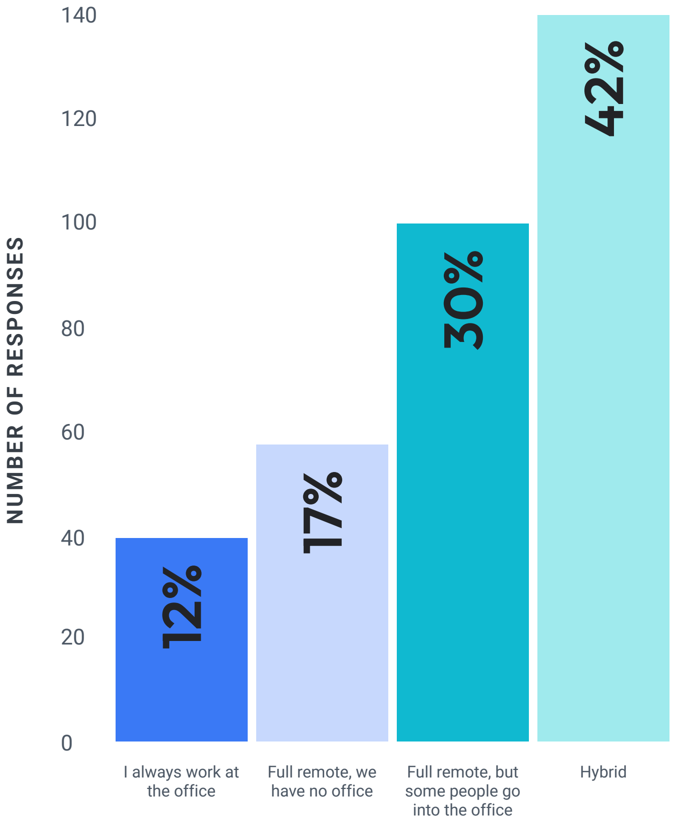


Work Setup

The vast majority of respondents work from home in some capacity, with only 11.7% working full-time in an office.

However, the office population almost doubled from last year (11.6% vs 6.3%), when we asked the same question in a similar survey. The full remote share, on the other hand, stayed the same, with most of the migration happening from hybrid to office setups.

This suggests—as is to be expected—that a fully remote model is significantly harder to revert than hybrid.



Supplementary LinearB Labs Data

In November, LinearB published the fourth annual edition of their 2025 Software Engineering Benchmarks Report to help engineering teams baseline their performance against data-backed industry standards. Throughout this guide, you'll find supplemental insights from their analysis of 6.1+ million pull requests from 3,000+ organizations worldwide.

Data Sourced From

3,000+

Teams

6.1M+

PRs

32

Countries

Questions

Below you'll find a comprehensive list of all the questions we asked our survey participants:

- How much you agree with the following (1-5):
    - I am happy with the dev practices of my team
    - I am happy with the dev tools my team uses
    - My team's processes are efficient
    - I have enough focus time
    - I regularly need to wait for others to do work
    - It is easy to deploy new software
    - Our code review process is efficient
    - Software requirements are clear
- How often does your team ship to production?
- How long does it take to get a PR approved?
- How long does it take to deploy software to production after a PR is closed?
- How much time does your team spend on KTLO / maintenance / tech debt work?
- How often does your team deliver projects on time?
- How often do you actually spend your time the way it was planned?
- Does your team intentionally allocate engineering investment to different areas?
- What percentage of your work (or PRs) are linked to a project issue?
- Do you feel engineering work is well-regarded among the leadership and non-technical stakeholders?
- Does your team track and review any team productivity or efficiency metrics?
- Which engineering metrics do you track?
- Do you use metrics at team or individual level? Or both?
- How useful do you find engineering metrics?
- How does your team use metrics and data to improve? (Open question)
- What is the #1 benefit you get from using metrics?
- What is the #1 issue you have in using metrics?
- Is there something else you would like to tell us?

It's a lot to cover!
Here is how we worked through their answers.

Methodology

Finding good, reliable insights is hard, for a myriad of reasons:

- **Correlation ≠ causation** — Even if some things are clearly correlated, it can be difficult to figure out what's upstream and what's downstream.
- **Statistical significance** — Given the (relatively) small sample size, you can only trust big deviations.
- **Human bias** — We are human, and thus come to the party with our own set of opinions and biases. If I expect some result, I may more or less intentionally look for it in the data.

So, to find clear patterns, we put all the data in a giant Airtable, where we exploded results by several dimensions.

Then, for each candidate insight, we looked for confirmation from multiple angles, including similar questions, related topics, and open answers where respondents wrote about their team's practices.

	Count	Happy with dev practices	No need to wait for others	Spend time as planned	Time spent on KTLO	How often deliver on time
100% office	37	2.95	2.89	47%	25%	47%
Hybrid	137	3.44	3.11	48%	25%	57%
Full remote with office	97	3.46	3.09	57%	23%	58%
Full remote no office	57	3.42	3.18	48%	24%	55%

Above, you can see a slice of our response data, broken down by work setup. We look for interesting + statistically significant deviations, look for additional confirmation, and try to interpret findings.

For the open answers, we went through all of them manually to extract individual stories and ideas. We then tagged them into various categories, plus used AI to uncover gems we might have missed.

The result is a report that, while still subject to our own expert interpretation, aims to be a profound, reliable analysis that blends thousands of quantitative and qualitative data points.

Key Takeaways

Below you'll find a high-level view of the key takeaways from the report, highlighting the most significant findings and actionable insights.

Successful teams have three clear traits

- Engineering is well-regarded among leadership and non-tech stakeholders
- Engineers are happy about their dev practices
- Projects consistently ship on time

Metrics work on two levels

- **Tactical** — Used in sprints, retros, and daily ceremonies
- **Strategic** — Used for planning, resource allocation and performance reviews

Engineering maturity follows a pyramid

- **Transparency** (foundation) — Being aware and creating feedback loops
- **Intent** (middle) — Intentional allocation and clear direction
- **Speed** (top) — Only valuable when built on the other two layers

Engineering thrives with autonomy & mastery

Teams perform best when developers:

- Have enough focus time
- Spend time as planned
- Don't need to wait for others

Six practices correlate strongly with success

- Some remote flexibility (hybrid or full remote)
- Intentional allocation of engineering time
- Clear requirements
- Frequent code shipping
- Quick PR reviews (minutes, not days)
- Tracking engineering metrics

What Is A Successful Engineering Team?

To figure out what practices lead to successful engineering teams, first we need to understand what a successful team looks like.

Out of all the questions we asked,
there are three we want to focus on:

- Engineers are happy about dev practices
- Engineering is well-regarded among leadership / non-tech stakeholders
- Projects ship on time

These are the elements that correlate the strongest with the highest number of positive behaviors. That is: the gradient of answers for these questions is most likely to be matched by a similar gradient in other behaviors.

		Enough focus time	Spend time as planned	No need to wait for others	Happy with dev practices
Engineering is well regarded among non-tech stakeholders	Absolutely yes	↑ 3.40	↑ 51%	↑ 3.35	↑ 3.81
	Yes	↑ 3.24	↑ 56%	↑ 3.20	↑ 3.39
	Meh	↑ 2.90	↑ 48%	↑ 3.02	↑ 3.34
	No	↑ 2.79	↑ 40%	↑ 2.59	↑ 3.00
	Absolutely not	↑ 2.45	↑ 35%	↑ 2.45	↑ 3.00

		Spend time as planned	Enough focus time	No need to wait for others
Engineers are happy about dev practices	Strongly agree	↑ 59%	↑ 3.80	↑ 3.47
	Agree	↑ 54%	↑ 3.28	↑ 3.12
	Neither disagee nor agree	↑ 47%	↑ 2.88	↑ 2.99
	Disagree	↑ 44%	↑ 2.78	↑ 2.87
	Strongly disagree	↑ 41%	↑ 2.25	↑ 4.00

		Spend time as planned	Enough focus time	No need to wait for others
Projects ship on time	Strongly agree	↑ 59%	↑ 3.80	↑ 3.47
	Agree	↑ 54%	↑ 3.28	↑ 3.12
	Neither disagee nor agree	↑ 47%	↑ 2.88	↑ 2.99
	Disagree	↑ 44%	↑ 2.78	↑ 2.87
	Strongly disagree	↑ 41%	↑ 2.25	↑ 4.00

Engineering being well-regarded, engineers being happy about their practices, and projects shipping on time correlate very strongly with positive engineering behavior.

Below are some examples we want to point out (see data above for more):

1. In teams where engineers are very happy about dev practices vs. those where they are neutral, **focus time is 30% more satisfactory**.

2. In teams where engineering is not well regarded by non-technical stakeholders vs. those where it is, **people need to wait for others 29% more of the time**.

3. In teams that ship 75% of projects on time vs. those who ship only 30% on time, people spend **28% more time as planned**.

So, the first two questions are strictly qualitative and collect the opinions of both technical and non-technical stakeholders. We believe these are extremely important: in fact, most quantitative practices (e.g. how often you ship, how much time is spent on KTLO, …) depend on the team's context and do not decisively sort good teams from bad ones.

Conversely, we would be hard-pressed to find low performing teams where all stakeholders—engineers, managers, and leadership—are happy.

The last question is about predictability. We found that shipping on time correlates strongly with most good behaviors we polled for. By most measures, predictable shipping is even more important than frequent shipping.

So, what kind of traits are exhibited by teams that ship on time and where stakeholders are happy? As you can see from the pictures, three traits are the most common:

Engineers have enough focus time

Engineers spend time as it was planned

Engineers don't need to wait for others

This is not surprising, but it is always nice to see it backed by numbers.

LinearB Labs

Pro-Tip: To begin accurately forecasting project delivery and determine if timelines can be moved up, we recommend tracking quality and predictability metrics like **Planning Accuracy** and **Capacity Accuracy** (as defined below):

Capacity Accuracy

Measures how many issues (or story points) a team completed in an iteration (planned or unplanned) compared to the amount planned for that iteration.

Planning Accuracy

The ratio of planned work vs. what is actually delivered during a sprint or iteration. High planning accuracy signals a high level of predictability and stable execution.

Metric	Elite	Good	Fair	Needs Focus
Capacity Accuracy (%)	85 - 115%	75 - 85% or 115 - 125%	70 - 75% or 115 - 130%	< 70% or > 130%
Planning Accuracy (%)	> 80%	80% - 65%	64% - 45%	< 45%

3,026 Orgs, 6,100,878 Pull Requests, 167,437 Active Contributors, Time metrics in minutes or hours, as noted

In rare cases teams may have too high Planning Accuracy (>95%) and too High Capacity Accuracy (>115%). That means they aren't taking on enough planned work–which should be a priority for improvement initiatives.

LinearB Labs' research revealed that over half (69.9%) of engineering projects under-commit to their iteration plans, and that less than a fifth (only 16.5%) were in the ideal range.

DevEx and DevProd metrics can be very useful early indicators for predictable delivery. Tracking and improving metrics such as PR Size, Code Rework and Cycle Time can help teams improve how they make and keep accurate promises to the business about delivery timelines.

Engineering thrives when developers can cultivate **autonomy** and **mastery**. Conversely, it suffers when developers are not in control of their time, need to constantly wait for others, and spend time in meetings and putting out fires instead of doing creative work.

But what enables focus time, no waiting, and predictable work? Let's get into the core of the report, which breaks down the successful practices of elite engineering teams.

> Software projects can be unpredictable due to a multitude of reasons — from unforeseen technical challenges to scope changes. Engineering metrics, such as Planning Accuracy, and workflow automation tools have helped us increase predictability in release schedules and timelines."
>
> **Marko T.**
> CTO, Assignar

Successful Practices

A good portion of our survey was comprised of questions about team or engineering practices. Once we identified what successful teams look like, we tried to figure out what practices correlate the most closely with success indicators.

Here are the six where such correlation is the strongest:

	Happy with dev practices	No need to wait for others	Spend time as planned	Time spent on KTLO	How often deliver on time	Enough focus time
100% office	2.95	2.89	47%	25%	47%	3.19
Hybrid	3.44	3.11	48%	25%	57%	3.03
Full remote with office	3.46	3.09	57%	23%	58%	3.16
Full remote no office	3.42	3.18	48%	24%	55%	3.18

Work Setup

The various work environments don't say much about how people work except for one case: 100% office-based teams.

Those teams fare worse than others on several measures:

- Engineers need to wait for others 10% more
- Projects are delivered on time 21% less frequently
- Engineers are, on average, 16.6% less happy about dev practices

This could be for a variety of reasons. First of all, in-person teams often face more interruptions, reducing focus time for deep work. Plus, the lack of autonomy in traditional office settings can also lead to dissatisfaction with development practices, as engineers may feel constrained by outdated processes or micromanagement, reducing overall happiness and productivity.

Other setups, conversely, are pretty comparable: fully remote is not decisively better or worse than hybrid.

These results suggest that some remote flexibility significantly helps developers, but fully remote is not necessarily a step up from a good hybrid setup.

Intentionally Allocating Engineering Time

Intentionally allocating engineering time across various categories (e.g. KTLO vs Improvements vs New Features) has strong positive correlation with several traits:

+6.7%

Focus time (3.16 vs 2.96)

+22%

Time spent as planned (55% vs 45%)

+24%

Projects delivered on time (62% vs 50%)

+14.7%

Happiness about dev practices (3.57 vs 3.11)

	Time spent on KTLO	Spend time as planned	How often deliver on time	Happy with dev practices	Enough focus time	No need to wait for others
Yes	= 24%	↑ 55%	↑ 62%	↑ 3.57	↑ 3.16	↑ 3.12
No	24%	↑ 45%	↑ 50%	↑ 3.11	↑ 2.96	↑ 3.03
I have no idea	26%	↑ 49%	↑ 49%	↑ 3.42	↑ 3.25	↑ 3.15

This is something we often discussed on the newsletter and we were already big fans of, but we were surprised by how strong the correlation is.

LinearB Labs

Pro-Tip: LinearB's Engineering Investment Benchmarks (below) provide a high level view into where and how engineering teams are investing their resources. These benchmarks represent the average investment split across many organizations.

New Value

The actions performed to invest in new features that increase revenue and growth by new customer acquisition or expansion. This might include activities such as:

- Adding a new feature
- Implementing roadmap work, etc.
- Supporting a new platform or partner application

Feature Enhancements

The actions taken to enhance features or deliver a product that ensures customer satisfaction. This might include activities such as:

- Customer requested improvements
- Improved performance & utilization
- Improved product reliability or security, etc.
- Iterations to improve adoption/ retention/quality

Developer Experience

The actions performed to improve the productivity of development teams and their overall experience. This might include activities such as:

- Code restructuring
- Testing automation
- Better developer tooling
- Working to reduce the size of the KTLO bucket in the future

KTLO (Keeping the Lights On)

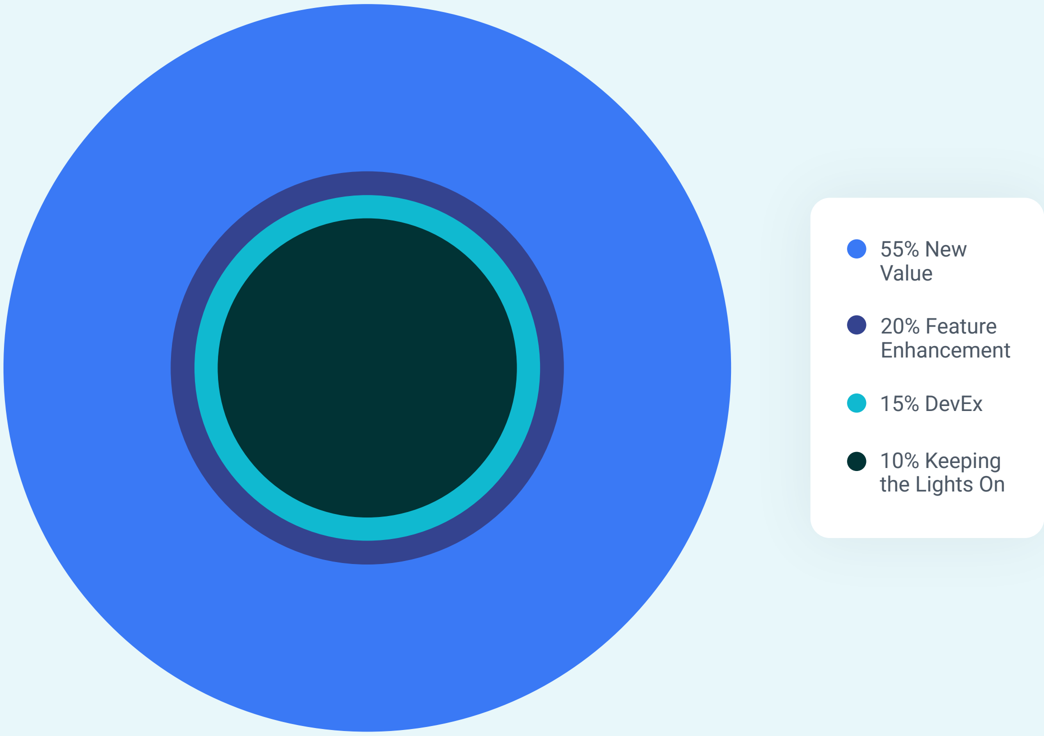
The minimum tasks a company is required to perform in order to stay operational on a daily level, while maintaining a stable level of service. This might include activities such as:

- Maintaining current security posture
- Service and ticket monitoring & troubleshooting
- Maintaining current levels of service uptime, etc.

We recommend using these categories and investment percentages as a starting point when aligning R&D resource investment with the board and executive team.

Investment Profile

3,026 Orgs
6,100,878 Pull Requests
167,437 Active Contributors
Time metrics in minutes
or hours, as noted



Engineering teams can use the investment benchmarks to help answer questions like:

- Is our Investment Profile very different rom what's typical - and if so is that difference serving us?
- Are we investing below average in New Value? Or above average in Keeping the Lights On?
- Are we balancing our investment in New Value with our investment in the tools and processes that allow new value to build more effectively?

Clear Requirements

Writing clear requirements is another big winner.

Most traits improve linearly and in tandem with how clear the requirements are perceived to be. For example, engineers who strongly agree about requirements being clear vs. those who disagree with it:

39%

Have more focus time (3.85 vs 2.76)

34%

Spend more time as planned (63% vs 47%)

34%

More projects shipped on time (66% vs 49%)

32%

Are happier about dev practices (4.15 vs 3.14)

	Spend time as planned	How often deliver on time	Happy with dev practices	Enough focus time
Strongly agree	↑ 63%	↑ 66%	↑ 4.15	↑ 3.85
Agree	↑ 56%	↑ 64%	↑ 3.68	↑ 3.53
Neither disagree nor agree	↑ 48%	↑ 56%	↑ 3.43	↑ 3.02
Disagree	↑ 47%	↑ 49%	↑ 3.14	↑ 2.76
Strongly disagree	↑ 43%	↑ 44%	↑ 2.57	↑ 2.63

Ship Code Often

Shipping frequently displays solid correlations with time spent on maintenance and engineers' overall happiness.

As seen in the data, teams who ship multiple times a day vs. teams who ship a few times per month:

- Wait for others 7.3% less of the time
- Spend 12% less time on maintenance (KTLO)
- Are 19.4% happier about their own dev practices

The correlation between shipping frequency and low maintenance hints at what we discussed often, that more speed counterintuitively leads to more stability, as it enables teams to recover faster from mistakes.

At the same time, there is no significant correlation between shipping fast and people's focus time, nor delivering projects on time.

	Time spent on KTLO	Happy with dev practices	No need to wait for others	Spend time as planned	How often deliver on time	Enough focus time
Multiple times a day	↑ 22%	↑ 3.67	↑ 3.21	49%	57%	3.09
A few times a week	↑ 22%	↑ 3.39	↑ 3.13	= 52%	= 54%	= 3.27
A few times a month	↑ 25%	↑ 3.09	↑ 2.99	50%	55%	3.02

> “We deploy frequently, but versioning can be tricky. We now have completely seamless automatic deployments thanks to a custom gitStream checker that enforces semantic commits for automatic versioning during deployments.”
>
> **Jeff Williams**
> CTO, Contrast Security

LinearB Labs

Key-Takeaway: The longer the **Deploy Time**, the higher the **Change Failure Rate** (CFR).

Deploy Time

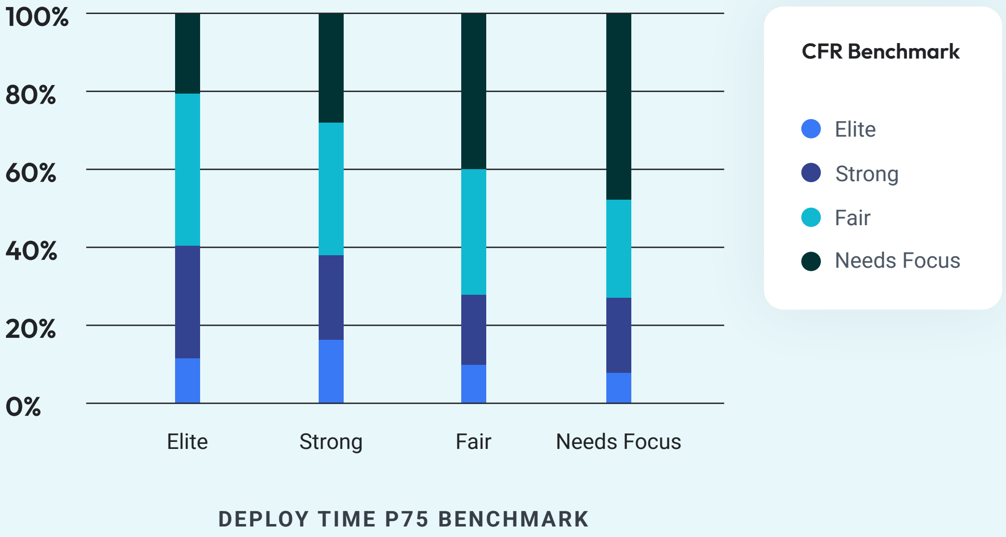
The time from when a branch is merged to when the code is released. Low Deploy Time correlates to high Deploy Frequency.

Change Failure Rate (CFR)

The percentage of deploys causing a failure in production.

CFR vs. Deploy Time P75 Benchmark Distribution

3,026 Orgs
6,100,878 Pull Requests
167,437 Active Contributors
Time metrics in minutes
or hours, as noted



When deployments take a significant amount of time, it can be for a variety of different reasons, namely:

- Larger deploy batches increasing the risk of failure.
- The more time that passes after code is merged, the higher the risk of drift and conflict with other changes.
- Longer deploy cycles often mean the developers writing the code are not directly responsible for deploying it, a detachment often correlated with lower sense of ownership and resulting quality.

Short-Lived PRs

Data about how long it takes for a PR to get approved is also interesting: there are no stark differences when the time ranges between 1 hour and 1 day, but performance is 1) significantly better when PRs take *minutes* to get approved, and 2) significantly worse when they take days.

	Time spent on KTLO	Spend time as planned	How often deliver on time	Happy with dev practices	Enough focus time	No need to wait for others
Few minutes	21%	55%	64%	3.59	3.30	3.41
One hour	27%	47%	52%	3.50	3.26	3.07
Half day	24%	51%	58%	3.44	3.03	3.01
One day	26%	54%	56%	3.47	3.44	3.21
Days	22%	44%	49%	3.02	2.67	2.92

Why is that? Our interpretation is the following:

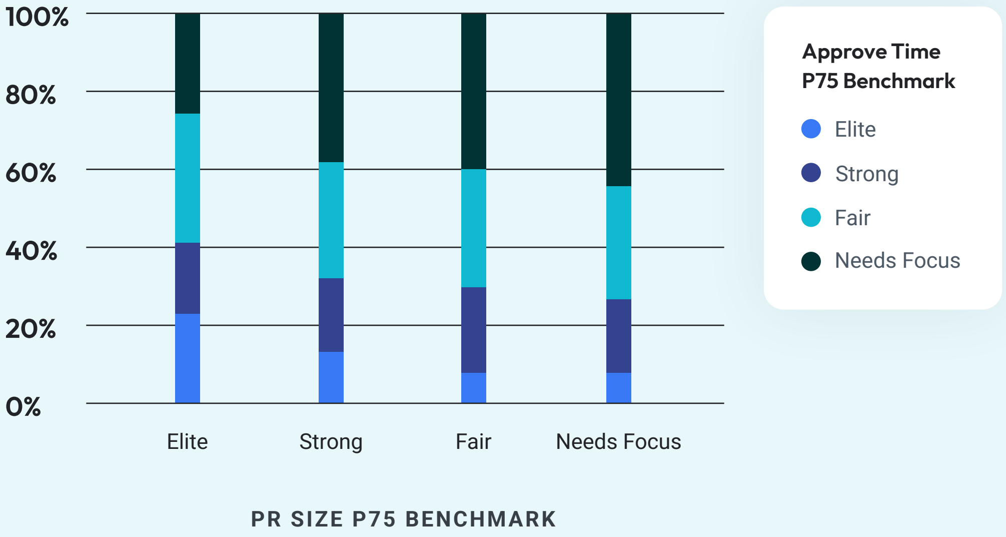
- **When a PR takes minutes** — Whether it's because PRs are small, people pair on reviews, or everyone stops in their tracks to review code — there is little to no context switch for the submitter, which leads to less work-in-progress, tasks getting resolved faster, and all kinds of benefits downstream.
- **When a PR takes between 1 hour and one day** — The submitter needs to switch to other tasks. At that point, whether the review takes 1 hour or, say, 4, there isn't a lot of difference in terms of workflow.
- **When a PR takes days** — The workflow degrades considerably as multiple changes need to get batched together, which creates more risk, more outages, rework, and a worse feedback loop.

Approve Time

The time from first comment to the first approval. This metric, along with Merge Time, is a more granular segment of Review Time.

Approve Time P75 vs. PR Size P75 Benchmark Distribution

3,026 Orgs
6,100,878 Pull Requests
167,437 Active Contributors
Time metrics in minutes
or hours, as noted



In the 2025 Software Engineering Benchmarks Report, LinearB Labs found that larger PRs take longer to approve. This is likely because larger PRs require reviewers to dedicate more time to comprehend the intent, scope, and potential impact of each change. This broader scope often leads to more detailed scrutiny and a higher likelihood of finding issues or raising questions, which in turn requires additional discussions, clarifications, and, in some cases, iterations.

Larger PRs can also lead to a perception of risk, as more extensive code changes carry a greater chance of introducing bugs or conflicts, prompting reviewers to approach with added caution and slower, more deliberate consideration. Plus, the involvement of multiple reviewers for cross-functional or cross-team input can slow down the approval process as more stakeholders need time to thoroughly review and align on the changes.

LinearB Labs

4 Proven Methods for Reducing PR Approval Time

When it comes to unblocking your teams' review processes, here are some best practices we recommend:

Assign the right reviewer

Leverage workflow automation to route PRs to developers with the most relevant recent activity and knowledge on the code being modified.

Set team goals to author smaller PRs

When PRs contain fewer lines of code, they present a less daunting undertaking for the reviewer, and are far more likely to get picked up quickly.

Get real-time alerts on PR activity

Setting up real-time notifications will provide you with immediate context about your PRs, including review assignments, approvals, comments and change requests.

Reduce cognitive load

Provide vital context through labels for estimated review time, sensitive code, and deprecated components.

Track Engineering Metrics

Finally, tracking engineering productivity metrics is positively correlated to most measures.

	Time spent on KTLO	Spend time as planned	How often deliver on time	Happy with dev practices	Enough focus time	No need to wait for others
Yes	= 25%	↑ 55%	↑ 64%	↑ 3.55	↑ 3.16	↓ 3.06
No	24%	↑ 47%	↑ 49%	↑ 3.25	↑ 3.04	↓ 3.15

Teams who track and use engineering metrics report:

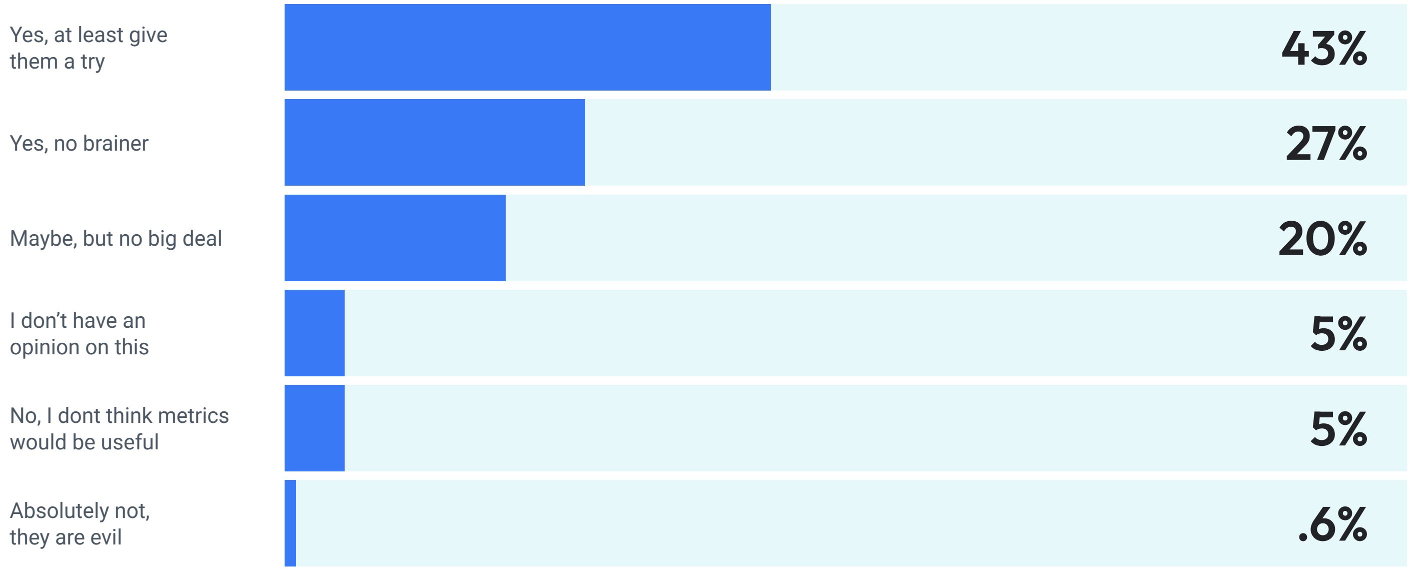
- +17% time spent as planned (55% vs. 47%)
- +30% of projects delivered on time (64% vs. 49%)
- +9.2% happiness about dev practices (3.55 vs. 3.25)

Also, the percentage of teams who track engineering metrics increased by 23% since last year (where we asked a similar question in a different survey), from 36.3% to 44.7%.

For teams that *do not* track metrics, engineers are also decisively in favor of trying, with 69.7% voting positively.

This leads us to the second part of this report.

Do you think your team should track some of those metrics, instead?



The Role of Metrics

One of the core topics we wanted to explore in this survey is how teams use data to improve. As we mentioned in the intro, there is a lot of talk around metrics frameworks, but very little info about how to use them in practice.

We dedicated a good portion of the survey to exactly this: asking qualitative questions about how people use metrics in their daily work, what benefits they report, and what challenges.

Here is what we learned:

Two levels of usage

Metrics are integrated in team processes to drive improvement and supplement decision making. You can divide processes into two main categories: tactical ones, on a weekly / bi-weekly basis, and strategic ones, on a monthly / quarterly basis:

1 Tactical

Here are some responses from our survey participants on how they use metrics in sprint planning, retrospectives, and other Agile ceremonies:

“Weekly review for planning and target setting. Code reviews and improving test coverage in daily scrum meet”

“We will monitor the commits and if it is too less we will go in to each commit to see what is going on. Also we enforced merge requests mandatory before deploying to stage. So we will know how the code is written”

“We set targets on certain team metrics. We use individual metrics only in 1:1 conversations and to back up/defend promo cases. We use sprint metrics to monitor and adapt sprint processes, and they’re looked at in every sprint review”

2 Strategic

Below you’ll find some quotes from our survey participants on how they use metrics in planning sessions, resource allocation discussions, and performance reviews.

“Metrics mainly helped us communicate expectations, challenges, successes and stumbling grounds with stakeholders. Overall satisfaction on both team and stakeholder side has improved as a result. We are now able to expose and address challenging areas in a transparent and structured way.”

“Optimize staffing, increase team efficiency, plan roadmap release dates”

Benefits

Teams report a variety of benefits that can be grouped into two major categories:

Transparency

Metrics enable better communication with all stakeholders, better collaboration and easier alignment. Two of our survey respondents wrote:

Each team has the same cycle time target, if not meet the manager will bring the topic at the retrospective so we dig in the why.

"We have a quarterly retrospective to review and adjust, and finally a yearly review to also set up next challenges to solve."

Enablement

Metrics allow engineering teams to surface data about things that are hard to measure, so stakeholders can identify their top-performing teams and make data-driven decisions.

"[They helped us] find bottleneck in the development flow. (eg: time it take before a review start)"

"Spot trends over time, which we try to understand to build improvements on"

Challenges

For many teams, adopting engineering metrics is a journey not without its challenges.

Here are the most commonly addressed issues:

Interpretation

Numbers often come without enough context, making it difficult to understand the big picture behind the data. It can also take time to understand how to translate metric data into actionable improvements.

Unintended consequences

There are instances where metric usage resulted in counterproductive actions or "gaming the system"

"One issue with using metrics like DORA and commit/day is that they can lead to unintended behaviors, such as focusing on improving numbers rather than genuinely enhancing processes, potentially encouraging superficial changes or excessive pressure on team members, which can undermine the true goal [...]"

Engineers also fear being micro-managed / time-tracked through metrics, though this seems a somewhat minor concern.

Alignment with business goals

It is not always trivial to connect technical metrics to broader organizational goals.

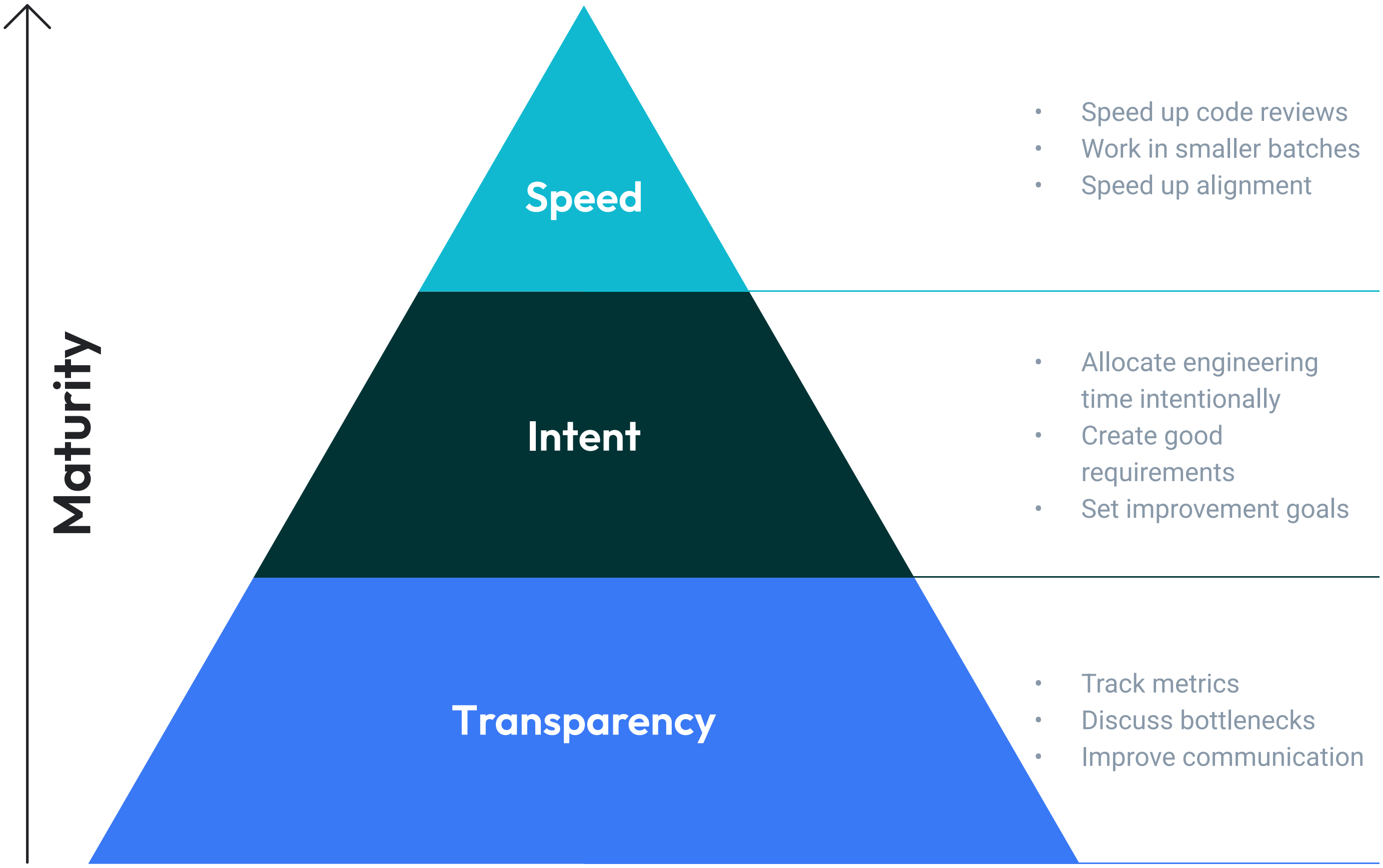
"Balancing engineering and business metrics so that engineering metrics support business goals without becoming the target themselves"

The Pyramid of Engineering Maturity

After we collected the insights above, we tried to figure out how to turn these into practical recommendations for engineering teams. How can you improve your engineering maturity through data? Where do you start?

After combing through the quantitative and qualitative answers, we put together a pyramid of maturity, which is comprised of three levels:

- Transparency
- Intent
- Speed



- Speed up code reviews
- Work in smaller batches
- Speed up alignment

- Allocate engineering time intentionally
- Create good requirements
- Set improvement goals

- Track metrics
- Discuss bottlenecks
- Improve communication

Transparency

Transparency is about being *aware* of what's going on, and creating the correct feedback loop to discuss and design improvements.

Good communication, good retrospectives and continuous improvement are the #1 benefit that teams report from embedding data into their processes — and are foundational to everything the team does.

Even without setting specific targets, data helps you figure out if you are trending in the right direction, and if your initiatives are making things better or worse.

Intent

Transparency enables *intent*.

The second clear trend that emerges out from both our quantitative and qualitative results is that teams that are intentional about parts of their engineering process are rewarded.

Some of the clear wins they see are:

- Setting improvement goals for parts of the engineering process.
- Creating better, participated requirements for features and projects.
- Allocating engineering time across a balanced portfolio of initiatives.

Speed

Speed is the tip of the pyramid.

Going fast is only helpful and sustainable when you are going in the right direction (intent) and you have a good feedback loop to steer your practices if needed (transparency).

This is clear from looking both at the correlations we found, and those we didn't find: good teams are fast, but fast teams are not necessarily good.

Shipping every day doesn't magically turn you into an elite engineering team — but if you are an elite engineering team chances are you are shipping every day.

What Now?

See where your team stacks up with a free forever account and begin building your engineering metrics program today!

Schedule a demo to discuss setting up your own engineering metrics program.

Additional Data Reports & Guides for R&D Leaders

2025 Software Engineering Benchmarks Report

The 2025 Software Engineering Benchmarks Report was created from a study of 6.1+ M PRs from 3,000 engineering teams across 32 countries.

Download Report

Engineering Leader's Guide to Developer Productivity

Discover how to quantify developer productivity, common blockers, strategies to improve it, and how and when to present dev productivity data.

Download Guide

Measuring Impact: The GenAI Code Report

This LinearB Research Report breaks down how to measure the impact of Generative AI code across the software delivery lifecycle.

Download Report