2025 Software Engineering Benchmarks Report

Created from a study of 6.1M+ PRs from 3,000 engineering teams across 32 countries.

DATA REPORT

LinearB

Table of Contents

Introduction

We're thrilled to be publishing the fourth annual edition of our 2025 Software Engineering Benchmarks Report! This year, we've focused our research on two key areas of Software Engineering Intelligence (SEI): Developer Experience and Developer Productivity, as defined below.

Developer Experience (DevEx)

A development team's overall morale and engagement when interacting with their organization's tools, processes and environments.

Developer Productivity (DevProd)

How effectively and efficiently developers can complete meaningful tasks quickly and with minimal waste.

DATA SOURCED FROM

3,000+	6.1M+	32
TEAMS	PRS	COUNTRIES

Balancing the relationship between these priorities in your engineering organization is crucial but often misunderstood. A lot of software teams feel pressured to prioritize DevProd over DevEx due to tight deadlines and limited resources, thinking they must prioritize short-term output over developer well-being. There's also a misconception that investing in DevEx — such as improving tools or enhancing workflows — will take effort away from new value delivery, when in fact, it often leads to sustained Developer Productivity gains. This mindset overlooks how a better DevEx reduces developer toil and burnout, thereby creating a more resilient, and ultimately, productive team.

At LinearB, we recommend taking an approach that balances both outcomes. To achieve both results, you need visibility into what's working, and what isn't. But visibility without context doesn't solve your problem — you need benchmarks to answer essential questions like: 'Is my team's 7-day Cycle Time good or bad?' and more existential ones like: 'Am I running a healthy engineering organization?' That's why we studied the work of over 3,000 dev teams and 6.1 million pull requests (PRs) to develop industry-standard engineering benchmarks – to help you contextualize your metrics, see how your teams are stacking up against the industry and meet improvement goals.

In this report, you'll find all metrics organized by the following criteria:

01 Organization

02 Size

03 Geography

04 Industry

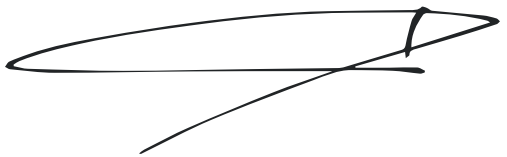
It's important to note that all data has been anonymized and normalized. For aggregation, we used the P75 (75th percentile) calculation. P75 is less sensitive to extreme values or outliers in the data, providing a robust and reliable measure.

TOOLTIP

We've organized the data into the following levels of performance for each metric:

ELITE	Top 10% of the LinearB community
GOOD	Top 30% of the LinearB community
FAIR	Top 60% of the LinearB community
NEEDS FOCUS	Bottom 40% of the LinearB community

We hope that you enjoy these fascinating new insights into how engineering teams work, set goals and achieve success.



Yishai Beeri
CTO, LinearB

2024 vs 2025: What's New this Year?

This year's report includes 7 brand-new metrics, plus our sample size has doubled from last year, with our analysis now spanning 3,000+ teams, 167k+ contributors and 6.1+ million PRs. We're also including new research on bot-generated PRs.

NEW DEVPROD AND DEVEX METRICS

Approve Time

Measures the time from the first comment on a pull request to when the PR is first approved.

Merge Time

Measures the time from first approval to when the PR is merged.

PR Maturity

The ratio between the total changes added to a PR branch after the PR was published and the total changes in the PR. Example: A PR was merged with a total of 100 lines of code. 20 lines were modified since the PR was published. In this case, PR Maturity is 80% (0.8).

Why We're Including Them

- **Approve Time** and **Merge Time** split **Review Time** into two sub segments with distinct dynamics — use this additional level of detail to more accurately diagnose your code review bottlenecks.
- **PR Maturity** gives a unique view into the impact of code reviews, highlighting ineffective review (very high PR Maturity, PRs go nearly unchanged through review), as well as premature review (low PR Maturity, more work should be done by the developers to get the PRs to a better state prior to taking scarce review cycles).

NEW PROJECT MANAGEMENT (PM) HYGIENE METRICS

Issues Linked to Parents

The percentage of issues or tickets with active work that are linked to a parent issue, such as an epic or story. This does not include subtasks.

Branches Linked to Issues

The percentage of code branches that contain a reference to specific PM issues, providing visibility into the alignment of code changes with planned tasks.

In Progress Issues with Estimation

The proportion of ongoing PM tasks that have time or effort estimates assigned.

In Progress Issues with Assignees

The percentage of active PM tasks that have a designated team member responsible for completing them.

Why We're Including Them

- Both **Issues Linked to Parents** and **Branches Linked to Issues** are metrics that engineering teams can use as proxies for traceability. Optimizing for these metrics will make it easier for teams to monitor development progress and ensure that actual dev work is tied to defined, planned work items.
- Tracking both **In Progress Issues with Estimation** & **In Progress Issues with Assignees** helps teams ensure effective planning and ownership for work items, aiding in predictability accountability and effective workload management.

Software Engineering Benchmarks

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

Category	Metric	Elite	Good	Fair	Needs Focus
DevEx	Coding Time (hours)	< 1	1 - 4	5 - 23	> 23
	Pickup Time (mins/hours)	< 75 mins	75 mins - 5 hours	6 - 16	> 16
	Approve Time (hours)	< 10	10 - 22	23 - 41	> 41
	Merge Time (hours)	2	2 - 4	5 - 17	> 17
	Review Time (hours)	< 3	3 - 13	14 - 24	> 24
	Deploy Time (hours)	< 6	6 - 95	96 - 248	> 248
	Merge Frequency (per dev/week)	> 2.25	2.25 - 1.35	1.34 - 0.75	< .75
	PR Maturity (%)	> 91%	91 - 84%	83 - 77%	< 77%
DORA	Cycle Time (hours)	< 26	26 - 80	81 - 167	> 167
	Deploy Frequency (per service)	> 1	1 - 0.5	.4 - 0.15	< .15
	Change Failure Rate (%)	< 1%	1-4%	5-23%	> 23%
	MTTR (hours)	< 6	6 - 11	12 - 30	> 30
Predictability	PR Size (code changes)	< 85	85 - 138	139 - 209	> 209
	Refactor Rate (%)	< 11%	11 - 16%	17 - 20%	> 20%
	Rework Rate (%)	< 3%	3 - 5%	6 - 7%	> 7%
	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%
	PRs Without Review (%)	< 0.7%	.7 - 3%	4 - 13%	> 13%
PM Hygiene	Issues Linked to Parents (%)	> 91%	91 - 72%	71 - 62%	< 62%
	Branches Linked to Issues (%)	> 81%	81 - 64%	63 - 51%	< 51%
	In Progress Issues with Estimation (%)	> 59%	59 - 32%	31 - 20%	< 20%
	In Progress Issues with Assignees (%)	> 97%	97 - 86%	85 - 80%	< 80%

DORA Benchmarks

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

Category	Metric	Elite	Good	Fair	Needs Focus
DORA	Cycle Time (hours)	< 26	26 - 80	81 - 167	> 167
	Deploy Frequency (per service)	> 1	1 - 0.5	4 - 0.15	< 1.5
	MTTR (hours)	< 6	6 - 11	12 - 30	> 30
	Change Failure Rate (%)	< 1%	1 - 4%	5 - 23%	> 23%

Cycle Time

The time it takes for a single engineering task to go through the different phases of the delivery process from ‘code’ to ‘production’.

Deploy Frequency

Measures how often code is released. Elite Deploy Frequency represents a stable and healthy continuous delivery pipeline.

Mean Time to Recovery (MTTR)

The average time it takes to restore from a failure of the system or one of its components.

Change Failure Rate (CFR)

The percentage of deploys causing a failure in production.

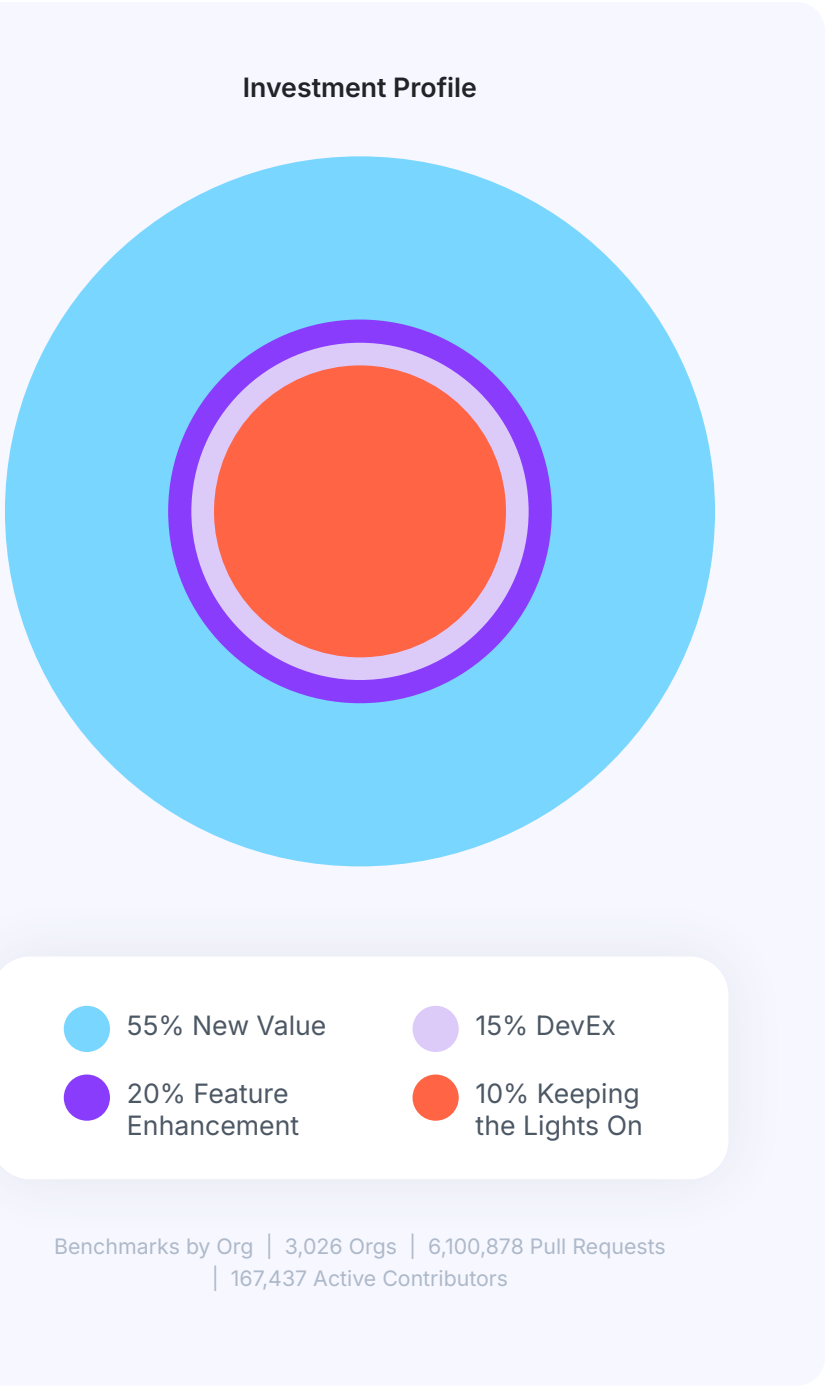
Engineering Investment Benchmarks

The Engineering Investment Benchmarks provide a high-level view into where and how engineering teams are investing their resources. Unlike our Metrics Benchmarks, you'll see the Investment Benchmarks do not include proficiency levels. Instead, these benchmarks represent the average investment split across many organizations.

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

Engineering teams can use the Investment Benchmarks to help answer questions like:

- Is our Investment Profile very different from 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?



CATEGORY BREAKDOWN

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:

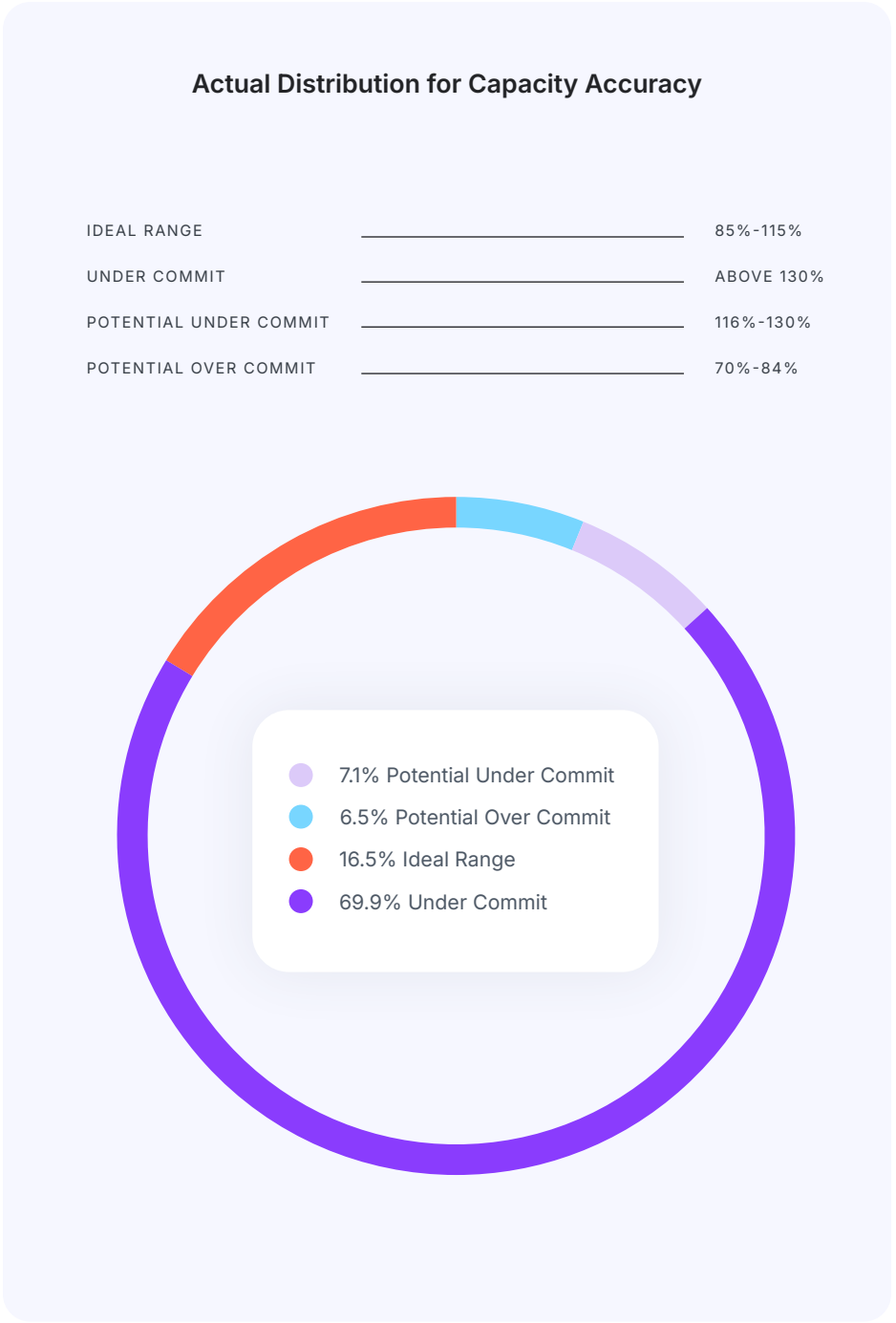
- Maintaining current security posture
- Service and ticket monitoring & troubleshooting
- Maintaining current levels of service uptime, etc.

Predictability Benchmarks

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.

Capacity Accuracy is a unique benchmark that is based on project data. It helps answer these questions:

- "Are teams taking on an amount of work that they can reasonably accomplish in an iteration?" Low Capacity Accuracy may point to over-commitment during planning.
- Are teams "playing it safe" and under-committing to always hit the mark? This would lead to high Capacity Accuracy.
- Do we have a good grasp of the team's work capacity and can predictably deliver that capacity sprint over sprint? Unstable Capacity Accuracy that swings from low to high shows that the team hasn't found its pace yet and cannot predict the amount of work they can deliver.



Predictability Benchmarks

> **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.

In a single “accuracy score” you’ll know if your teams are scoping iterations well, whether they’re completing their tasks and how unplanned work is affecting execution.

The calculation is based on the following types of work:

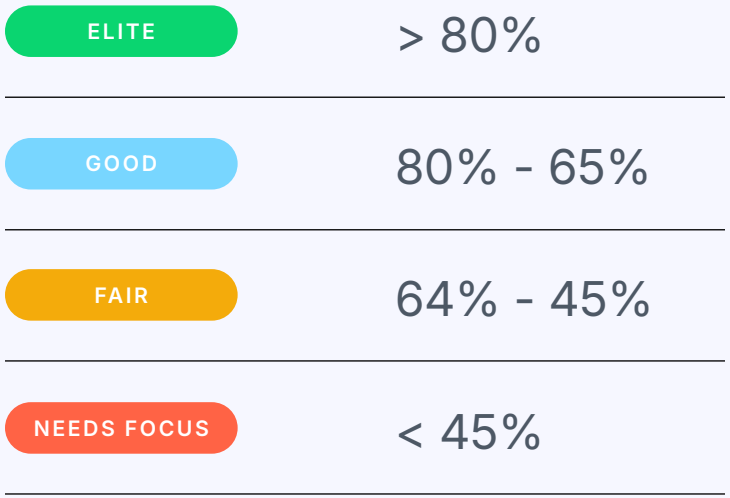
- **Planned:** Story points or issues added before or within 24 hours of a sprint beginning.
- **Added:** Story points or issues added and completed after the sprint begins.
- **Completed:** Planned story points or issues completed in a sprint.
- **Carryover:** Planned but not completed story points or issues.

In rare cases, teams may have a Planning Accuracy that is too high (>95%), and a Capacity Accuracy that is too high (>130%). That means they aren’t taking on enough planned work – which should be a priority for improvement initiatives.

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.

In our research, we found that over half (69.9%) of engineering projects under-commit 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.



Since we started tracking predictability metrics, we're getting things done faster. It's easy to see where our projects are at, who's doing what, and what needs attention. No more endless meetings and confusing email chains. Everything's right there in one place."

Leif Asmund M.

VP of Engineering, Seven Peaks Software

SEVEN PEAKS

Software Engineering Benchmarks By Org Size

ENTERPRISE
1000+ Employees

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

Category	Metric	Elite	Good	Fair	Needs Focus
DevEx	Coding Time (hours)	< 2	2 - 9	10 - 28	> 28
	Pickup Time (mins/hours)	< 78 mins	78 mins - 4 hours	5 - 15	> 15
	Approve Time (hours)	< 10	10 - 22	23 - 31	> 32
	Merge Time (mins/hours)	< 65 mins	65 mins - 4 hours	5 - 14	> 14
	Review Time (hours)	< 3	3 - 10	11 - 22	> 22
	Deploy Time (hours)	< 23	23 - 165	166 - 300	> 300
	Merge Frequency (per dev/week)	> 1.70	1.70 - 0.95	0.94 - 0.5	< 0.5
	PR Maturity (%)	> 92%	92 - 87%	86 - 80%	< 80%
DORA	Cycle Time (hours)	< 27	27 - 97	98 - 174	> 174
	Deploy Frequency (per service)	> 0.6	0.6 - 0.25	0.24 - 0.1	< 0.1
	Change Failure Rate (%)	< 1%	1 - 5%	6 - 30%	> 30%
	MTTR (hours)	< 6	6 - 12	13 - 50	> 50
Predictability	PR Size (code changes)	< 91	91 - 138	139 - 208	> 208
	Refactor Rate (%)	< 11%	1 - 15%	16 - 20%	> 20%
	Rework Rate (%)	< 4%	4 - 5%	6 - 7%	> 7%

Software Engineering Benchmarks By Org Size

SCALE-UP
200-1000 Employees

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

Category	Metric	Elite	Good	Fair	Needs Focus
DevEx	Coding Time (hours)	< 3	3 - 6	7 - 28	> 28
	Pickup Time (mins/hours)	< 65 mins	65 mins - 5 hours	6 - 17	> 17
	Approve Time (hours)	< 15	15 - 23	24 - 44	> 44
	Merge Time (mins/hours)	< 75 mins	75 mins - 4 hours	5 - 18	> 18
	Review Time (mins/hours)	< 21 mins	21 - 56 mins	57 mins - 3 hours	> 3
	Deploy Time (hours)	< 16	16 - 103	104 - 265	> 265
	Merge Frequency (per dev/week)	> 1.85	1.85 - 1.05	1.04 - 0.6	< 0.6
	PR Maturity (%)	> 90%	90 - 83%	82 - 76%	< 76%
DORA	Cycle Time (hours)	< 47	47 - 93	94 - 172	> 172
	Deploy Frequency (per service)	> 0.8	0.8 - 0.4	0.39 - 0.15	< 0.15
	Change Failure Rate (%)	< 1%	1 - 3%	4 - 15%	> 15%
	MTTR (hours)	< 8	8 - 12	13 - 35	> 35
Predictability	PR Size (code changes)	< 89	89 - 135	136 - 197	> 197
	Refactor Rate (%)	< 12%	12 - 16%	17 - 21%	> 21%
	Rework Rate (%)	< 4%	4 - 5%	6 - 7%	> 7%

Software Engineering Benchmarks By Org Size

STARTUP
0-200 Employees

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

Category	Metric	Elite	Good	Fair	Needs Focus
DevEx	Coding Time (hours)	< 1	1 - 4	5 - 22	> 22
	Pickup Time (mins/hours)	< 88 mins	88 mins - 5 hours	6 - 17	> 17
	Approve Time (hours)	< 12	12 - 21	22 - 39	> 39
	Merge Time (mins/hours)	< 77 mins	77 mins - 3 hours	4 - 17	> 17
	Review Time (hours)	< 3	3 - 13	14 - 24	> 24
	Deploy Time (hours)	< 5	5 - 95	96 - 245	> 245
	Merge Frequency (per dev/week)	> 2.30	2.30 - 1.4	1.39 - 0.85	< 0.85
	PR Maturity (%)	> 91%	91 - 84%	83 - 77%	< 77%
DORA	Cycle Time (hours)	< 26	26 - 78	79 - 171	> 171
	Deploy Frequency (per service)	> 1.1	1.1 - 0.5	0.49 - 0.15	< 0.15
	Change Failure Rate (%)	< 1%	1 - 5%	6 - 25%	> 25%
	MTTR (hours)	< 6	6 - 11	12 - 26	> 26
Predictability	PR Size (code changes)	< 90	90 - 142	143 - 214	> 214
	Refactor Rate (%)	< 11%	11 - 16%	17 - 21%	> 21%
	Rework Rate (%)	< 3%	3 5%	6 - 7%	> 7%

Metrics Definitions

DEVEX METRICS

PR Size

The number of code lines modified in a pull request. Smaller pull requests are easier to review, safer to merge and correlate to a lower Cycle Time.

Coding Time

The time it takes from the first commit until a pull request is published. Short Coding Time correlates to low WIP, small PR Size and clear requirements.

Pickup Time

The time a pull request waits for someone to start reviewing it. Low Pickup Time represents strong teamwork and a healthy review process.

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.

Merge Time

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

Review Time

The time it takes to complete a code review and get a pull request merged. Low Review Time represents strong teamwork and a healthy review process.

Deploy Time

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

Merge Frequency

The total number of pull requests or merge requests merged by a team over a period of time.

PR Maturity

The ratio between the total changes added to a PR branch after the PR was published and the total changes in the PR.

Rework Rate

The amount of changes made to code that is less than 21 days old. High Rework rates signal code churn and is a leading indicator of quality issues.

Refactor Rate

Refactored work represents changes to legacy code. LinearB considers code “legacy” if it has been in your codebase for over 21 days.

Metrics Definitions

DORA METRICS

Cycle Time

The time it takes for a single engineering task to go through the different phases of the delivery process from ‘code’ to ‘production’.

Deploy Frequency

A measurement of how often code is released. Elite Deploy Frequency represents a stable and healthy continuous delivery pipeline.

Mean Time to Recovery (MTTR)

The average time it takes to restore from a failure of the system or one of its components.

Change Failure Rate (CFR)

The percentage of deploys causing a failure in production.

PREDICTABILITY METRICS

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.

Capacity Accuracy

Capacity Accuracy measures all completed (planned and unplanned) work as a ratio of planned work.

PROJECT MANAGEMENT (PM) HYGIENE METRICS

Issues Linked to Parents

The percentage of issues or tickets within your PM instance that are linked to a parent issue, such as an epic or story. This does not include subtasks.

Branches Linked to Issues

The percentage of code branches that contain a reference to specific PM issues, providing visibility into the alignment of code changes with planned tasks.

In Progress Issues with Estimation

The proportion of ongoing PM tasks that have time or effort estimates assigned.

In Progress Issues with Assignees

The percentage of active PM tasks that have a designated team member responsible for completing them.

Insights

DISCLAIMER

It’s important to note that correlation does not indicate causation. However, the insights this data exposes align closely with the qualitative and anecdotal research we’ve gathered from LinearB users over the past year.

SUMMARY

PR Lifecycle Insights

KEY TAKEAWAY

PR Size is the most significant driver of velocity across the PR lifecycle.

Insight No. 1

Larger PRs wait longer to get picked up for review.

Insight No. 2

Larger PRs have longer Cycle Times.

Insight No. 3

Larger PRs take longer to approve.

Insight No. 4

PRs that wait longer for the review to start also take longer from approval to merge.

Insight No. 5

Larger PRs are modified more heavily during review.

PR Lifecycle Insights Analysis

KEY TAKEAWAY

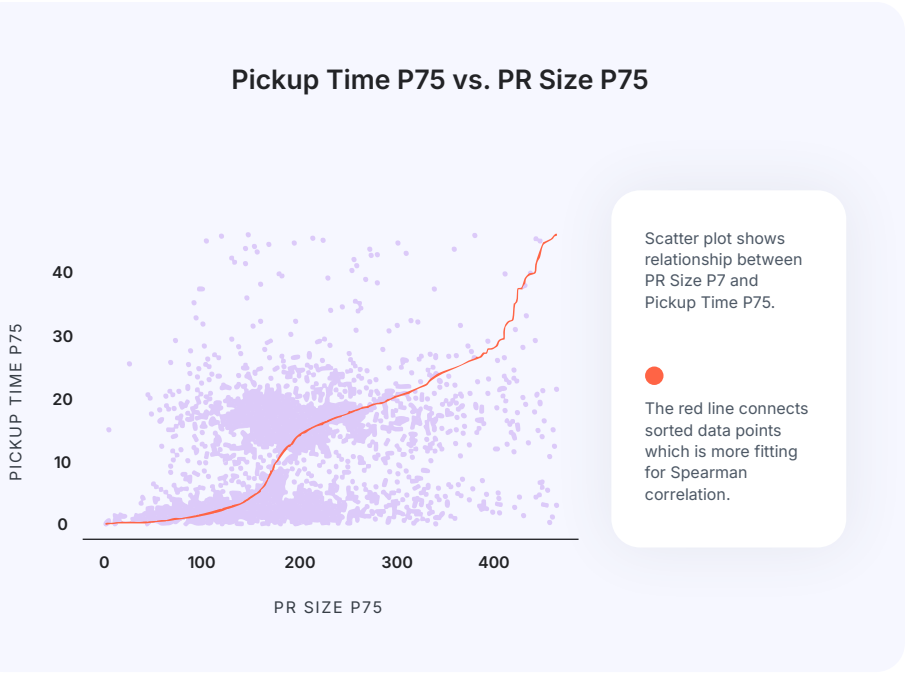
PR Size is the most significant driver of velocity across the PR lifecycle.

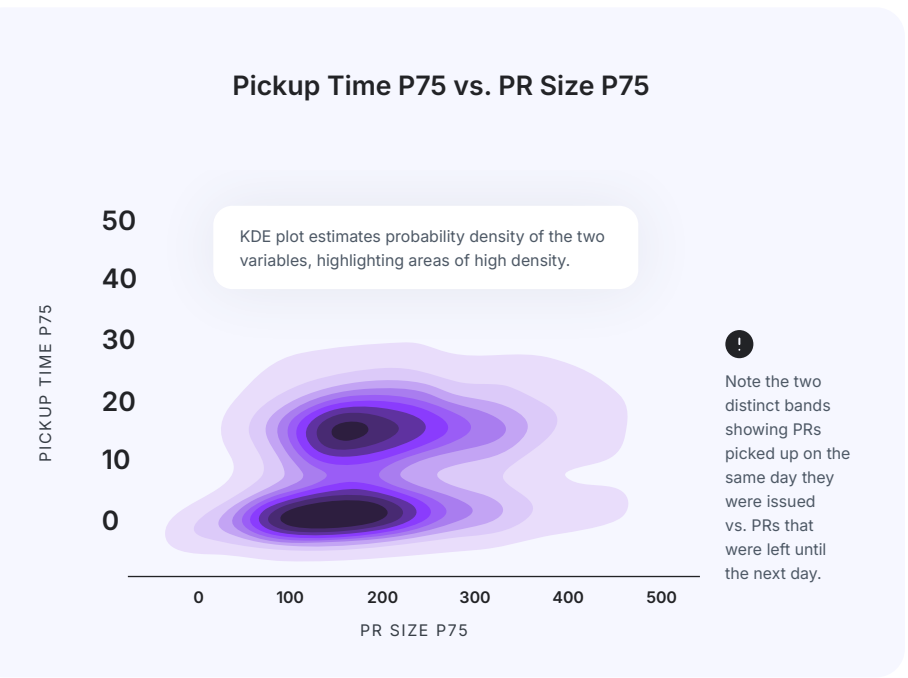
PR Size plays a critical role in how quickly code is shipped, since this metric directly impacts each subsequent phase of the development and review process, from PR pickup to merge. When pull requests are small, they tend to be less complex and lower risk, allowing reviewers to quickly understand, approve and merge changes. Conversely, larger PRs often require more in-depth reviews due to the complexity and higher likelihood of bugs, leading to lengthy feedback loops that have a compounding effect on velocity.

INSIGHT NO. 1

Larger PRs wait longer to get picked up for review.

When PRs contain many lines of code, they present a more daunting task for the reviewer, and are thus more likely to wait longer to get picked up. Additionally, larger PRs often involve multiple files, modules or systems, raising the potential for unintended side effects, which increases the probability of error. This complexity can also increase cognitive load, as reviewers need to take more time to understand how these changes integrate with existing code and detect possible bugs. Large PRs increase the chance of needing reviews from someone outside of the author's immediate team — another possible cause for delays.





INSIGHT NO. 2

Larger PRs have longer Cycle Times.

> **Cycle Time**
> The time it takes for a single engineering task to go through the different phases of the delivery process from 'code' to 'production'.

Larger PRs are the bane of fast moving teams. Harder to review, difficult to merge, and riskier to deploy, developers actively shy away from reviewing these, and every step in the delivery chain becomes longer and more drawn out.

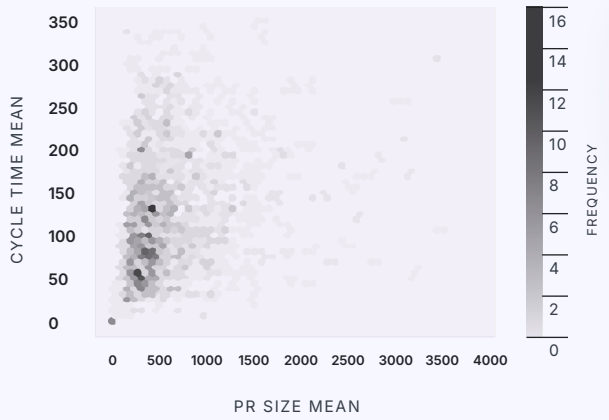
INSIGHT NO. 3

Larger PRs take longer to approve.

> **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.

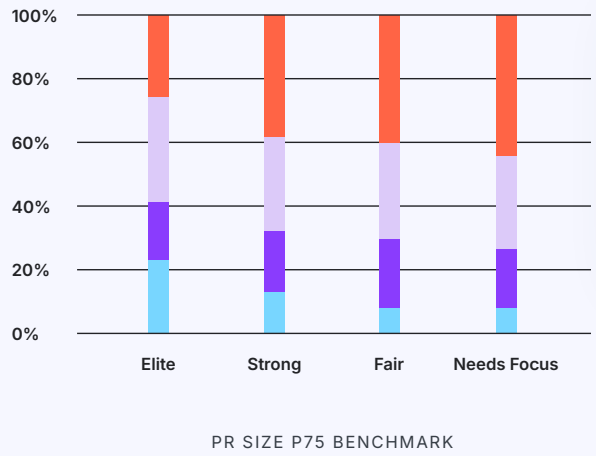
When it comes to larger PRs, reviewers must dedicate more time to comprehend the intent, scope and potential impact of each change. This broader scope often leads to more detailed scrutiny and a higher likelihood of finding issues or raising questions, which in turn requires additional discussions, clarifications, 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.

Cycle Time Mean vs. PR Size Mean



Hexbin plots display the density of points in hexagonal bins to visualize the concentration of data.

Approve Time P75 vs. PR Size P75 Benchmark Distribution



INSIGHT NO. 4

PRs that wait longer for the review to start also take longer from approval to merge.

Merge Time

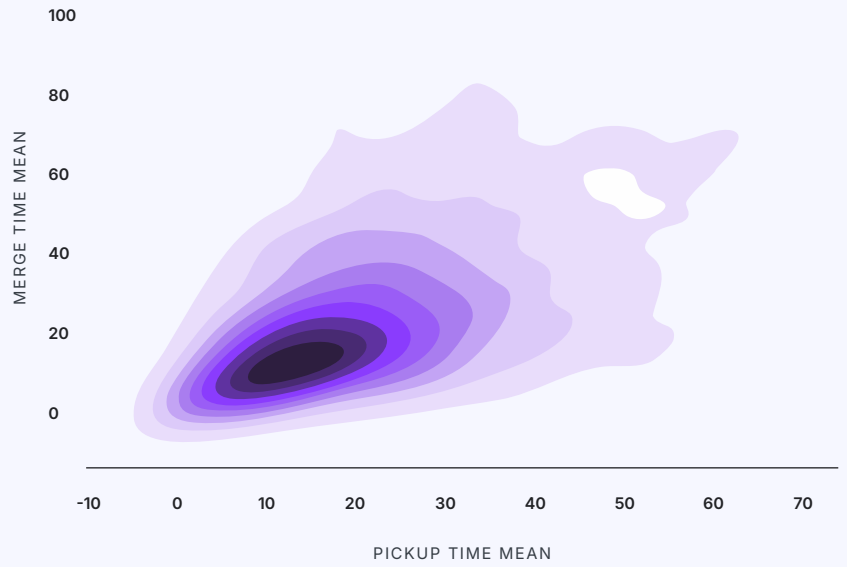
The time from the first approval to merge. This metric, along with Approve Time, is a more granular segment of Review Time.

Pickup Time

The time a pull request waits for someone to start reviewing it. Low Pickup Time represents strong teamwork and a healthy review process.

Longer Pickup Times often lead to extended Merge Times because the initial delay in reviewing a PR can disrupt workflow momentum. While this may seem like a foregone conclusion, it's important to understand the impact of longer PR lifecycle and context switches. When a PR sits idle waiting to be reviewed, it becomes increasingly likely that other changes in the codebase will create conflicts or alter dependencies, requiring the developer to revisit and rework the code before merging. This delay can result in a cascade of further reviews and adjustments, especially if updates from other branches must be incorporated to ensure compatibility. Additionally, the psychological effect of a delayed pickup can reduce urgency around completing the PR, which further slows down the review and merge stages. A lack of early feedback also limits opportunities to catch issues quickly, often resulting in last-minute adjustments that increase overall Merge Time.

2KDE Plot of Merge Time Mean vs. Pickup Time Mean



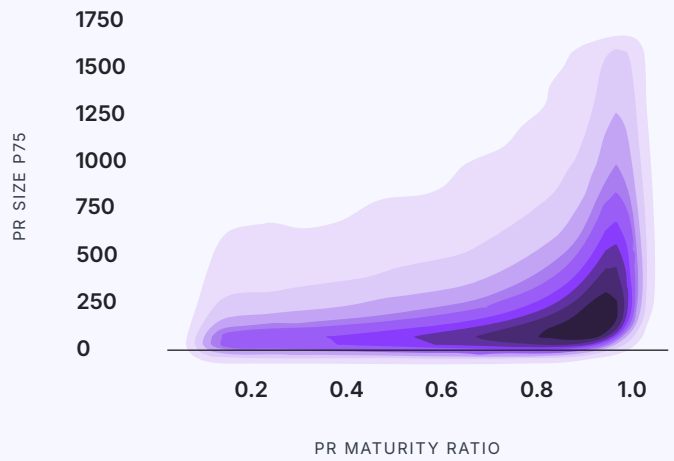
This 2KDE plot estimates probability density of the two variables, highlighting areas of high density.

INSIGHT NO. 5

Larger PRs are modified more heavily during review.

In small PRs, even a minor change added during the code review can result in a low PR Maturity Ratio (recall PR Maturity measures the ratio of code line modified after the PR was published). However, we see that small PRs tend to have a higher PR Maturity and require fewer modifications.

Some of this can be chalked up to skipped or rubber-stamped reviews for small PRs. Further, splitting large PRs into separate small ones is a hallmark of upfront planning and strong developers — and these will also tend to create more mature PRs.



TOOLTIP — 4 PROVEN METHODS FOR ELIMINATING BOTTLENECKS IN YOUR PR LIFECYCLE

When it comes to unblocking your team, here are some best practices we recommend:

1. 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.

2. 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.

3. 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.

4. Reduce cognitive load

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

SUMMARY

PM Hygiene Insights

KEY TAKEAWAY

Poor PM hygiene is directly correlated with shorter cycles.

Insight No. 1

When the percentage of Branches not Linked to Issues is higher, the shorter the Coding Time.

Insight No. 2

When the percentage of Branches not Linked to Issues is higher, the shorter the Review Time.

Insight No. 3

When the percentage of Branches not Linked to Issues is higher, the shorter the Merge Time.

PM Hygiene Insights Analysis

KEY TAKEAWAY

Poor PM hygiene is directly correlated with shorter cycles.

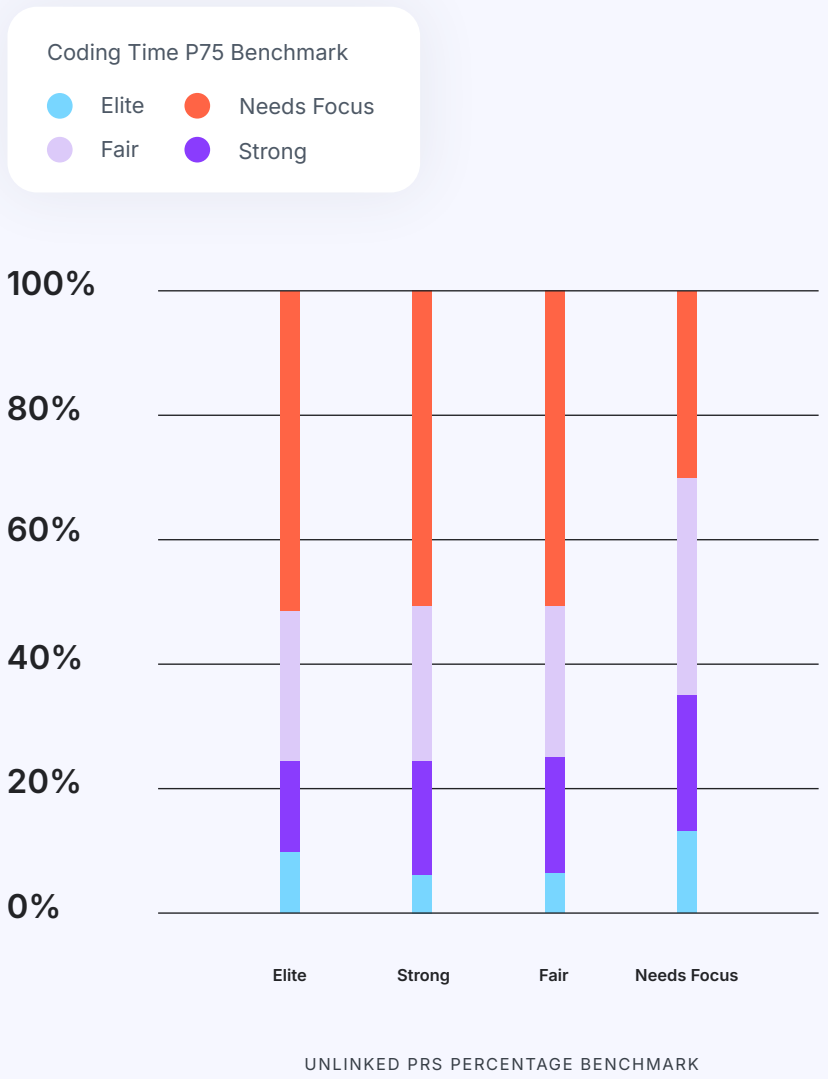
Paradoxically, poor PM hygiene may correlate with shorter cycles. For many teams, moving fast is synonymous to forgoing process and formal overhead. While this may be the right mode for some, care must be taken to track whether this approach also entails superficial reviews and reduced quality. Further, the reduced of visibility and tracking in the Project Management systems often leads to impaired predictability — a trait businesses often value more than raw speed.

INSIGHT NO. 1

When the percentage of Branches not Linked to Issues is higher, the shorter the Coding Time.

When a high percentage of branches are not linked to issues in your PM instance, Coding Time may initially be faster because developers can proceed with less formal overhead. Without having to create, link or update PM tickets, developers have fewer administrative tasks, allowing them to focus solely on coding and rapidly pushing changes. This lack of structure can enable more immediate responses to changes or requests, fostering an environment of quick iterations and minimal delays. However, this perceived efficiency often comes at a cost; without proper issue linkage, the work may lack essential context and alignment with broader project goals, potentially leading to miscommunication and technical debt down the line.

Coding Time P75 vs. Unlinked PRs Percentage Benchmark Distribution



INSIGHT NO. 2

When the percentage of Branches not Linked to Issues is higher, the shorter the Review Time.

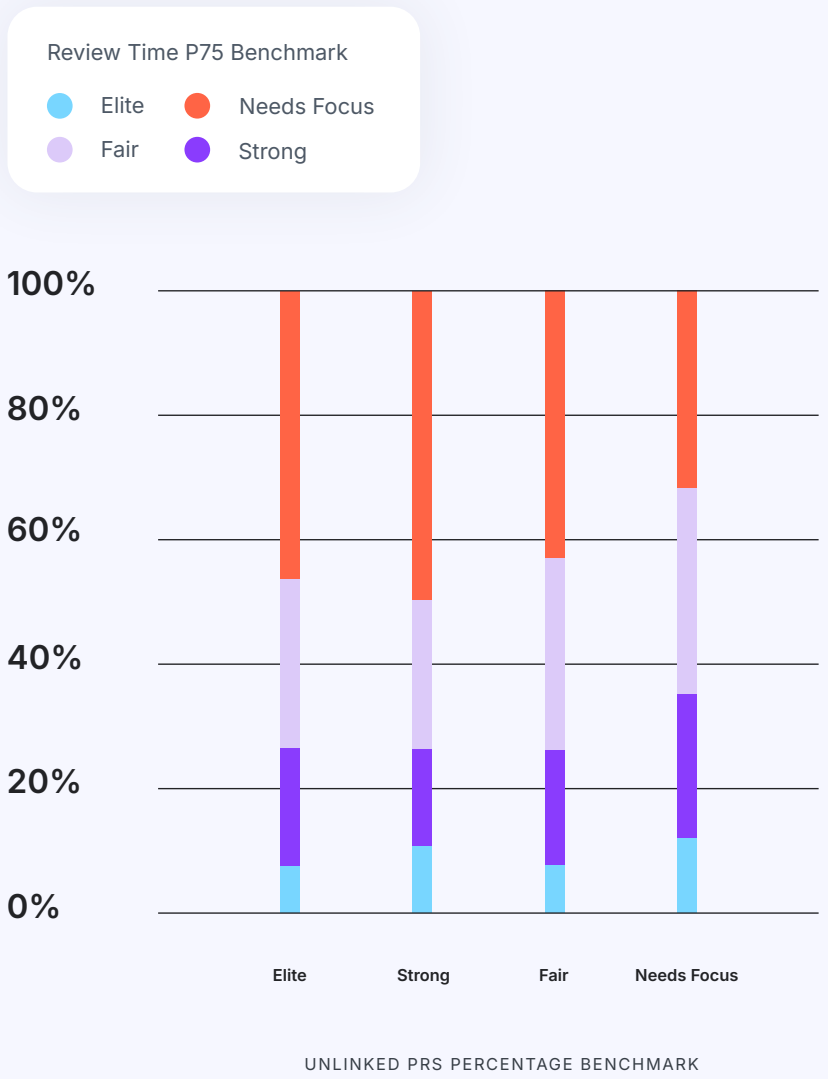
Review Time may initially be faster when the percentage of branches not linked to your PM tool is higher because reviewers might adopt a more cursory approach, given the lack of documented expectations. In this scenario, reviewers may focus solely on the technical aspects of the code without delving into the broader business goals, acceptance criteria or dependencies that typically accompany linked issues. This streamlined, code-only focus can lead to quicker evaluations, as there are fewer formal considerations to navigate and potentially fewer stakeholders involved. Additionally, the absence of a linked issue may create implicit pressure to expedite the review process, as reviewers have limited guidance on how deeply to assess the changes, leading to faster, though potentially less thorough, review cycles.

INSIGHT NO. 3

When the percentage of Branches not Linked to Issues is higher, the shorter the Merge Time.

Without the context provided by linked issues in your PM instance, merges may be treated as lower-stakes operations with minimal oversight, leading to faster integration into the main codebase. This unstructured approach can reduce the time spent on more detailed reviews and verification steps that would otherwise align code changes with business goals. However, this speed comes with a trade-off; while Merge Time is reduced, it may result in less stable integrations, potentially introducing unforeseen issues or conflicts that require additional Rework in the future.

Review Time P75 vs. Unlinked PRs Percentage Benchmark Distribution



SUMMARY

DORA Insights

KEY TAKEAWAY

Organizations with longer Cycle Times have a higher rate of failures in production.

Insight No. 1

The longer the Cycle Time, the higher the Change Failure Rate (CFR).

Insight No. 2

The longer the Deploy Time, the higher the Change Failure Rate (CFR).

DORA Insights Analysis

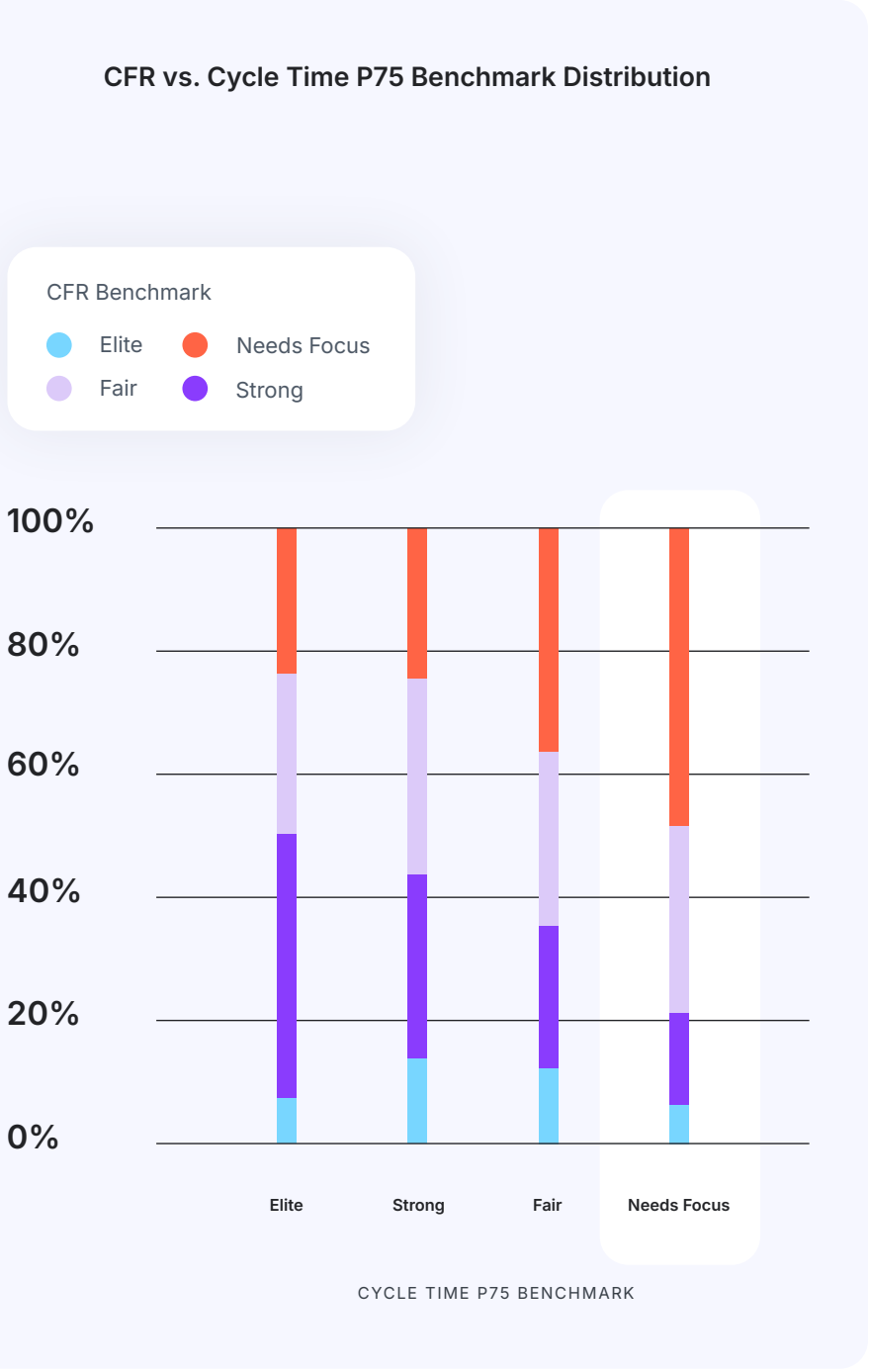
KEY TAKEAWAY Organizations with longer Cycle Times have a higher rate of failures in production.

When delivery cycles are longer, every deployment to production tends to be larger, more complex, and more prone to quality issues. Contrary to common instinct, and much like riding a bicycle, going faster actually helps stability. Teams that ship many small changs in short cycles have lower risk in each deploy and can fix production issues faster. They have typically also developed more robust automated testing capabilities that allow them to move faster in the first place — contributing to overall stability and reducing production failures.

INSIGHT NO. 1

The longer the Cycle Time, the higher the Change Failure Rate (CFR).

Longer Cycle Times often stem from large or intricate code changes that involve multiple revisions, dependencies, or significant Rework.This extended process makes it difficult for developers to keep track of concurrent changes in the codebase, which increases the likelihood of conflicts, outdated code and dependencies that are prone to break. The delay also means that feedback loops are longer, which can result in missed opportunities to catch errors early, leading to more significant issues in production. Consequently, as Cycle Time increases, so too does the complexity of maintaining a stable, conflict-free codebase, contributing to a higher Rate of Change Failures.



INSIGHT NO. 2

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.

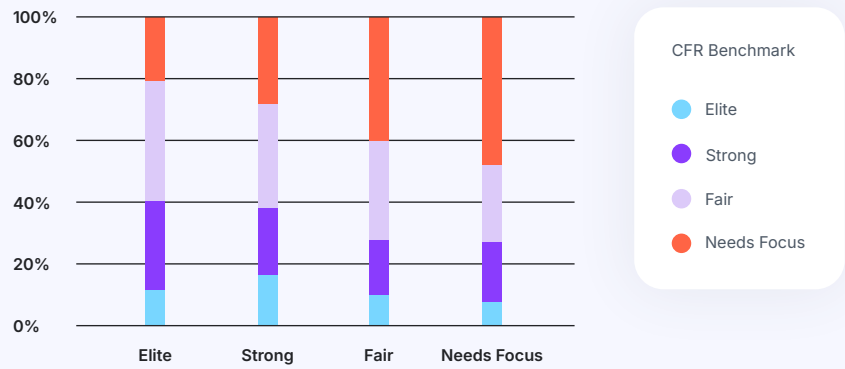
CFR (Change Failure Rate)

The percentage of deploys causing a failure in production.

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.

CFR vs. Deploy Time P75 Benchmark Distribution



DEPLOY TIME P75 BENCHMARK

TOOLTIP — 3 WAYS TO START IMPROVING YOUR DORA METRICS

Here are some best practices we recommend to dev teams looking to improve their DORA Metrics:

1. Set expectations with PR Labels

Using automated labels to categorize PRs (e.g. bug fix, high risk, documentation) can help reviewers prioritize the most important work and plan their days accordingly.

2. Request changes on deprecated APIs

Set up automated alerts whenever a PR includes use of a deprecated API. This allows engineering managers to encourage best practices while remaining hands off.

3. Reduce cognitive load with Idle PR Alerts

Cut dev idle time by 60% with real-time pull request alerts to nudge your team when a PR has been sitting idle for 2 days or more.

SUMMARY

Quality Insights

KEY TAKEAWAY

A higher PR Maturity ratio correlates with higher velocity.

Insight No. 1

The higher the PR Maturity Ratio, the higher the Merge Frequency.

Insight No. 2

The higher the PR Maturity Ratio, the shorter the Pickup Time.

Quality Insights Analysis

KEY TAKEAWAY

A higher PR Maturity ratio correlates with higher velocity.

The maturity of a team's PRs serves as a strong indicator for how efficiently code moves through the development pipeline. When developers take the time to ensure PRs are thoroughly prepared before publishing them, they reduce the delays caused by fixes and additional reviews. Since every handoff between developers and reviewers requires an expensive context switch, anything the PR author can do upfront while creating the PR is almost always more efficient than having a reviewer do it. One caveat is when PR Maturity is too high — signaling that the review process is not effective as it doesn't cause PRs to be modified.

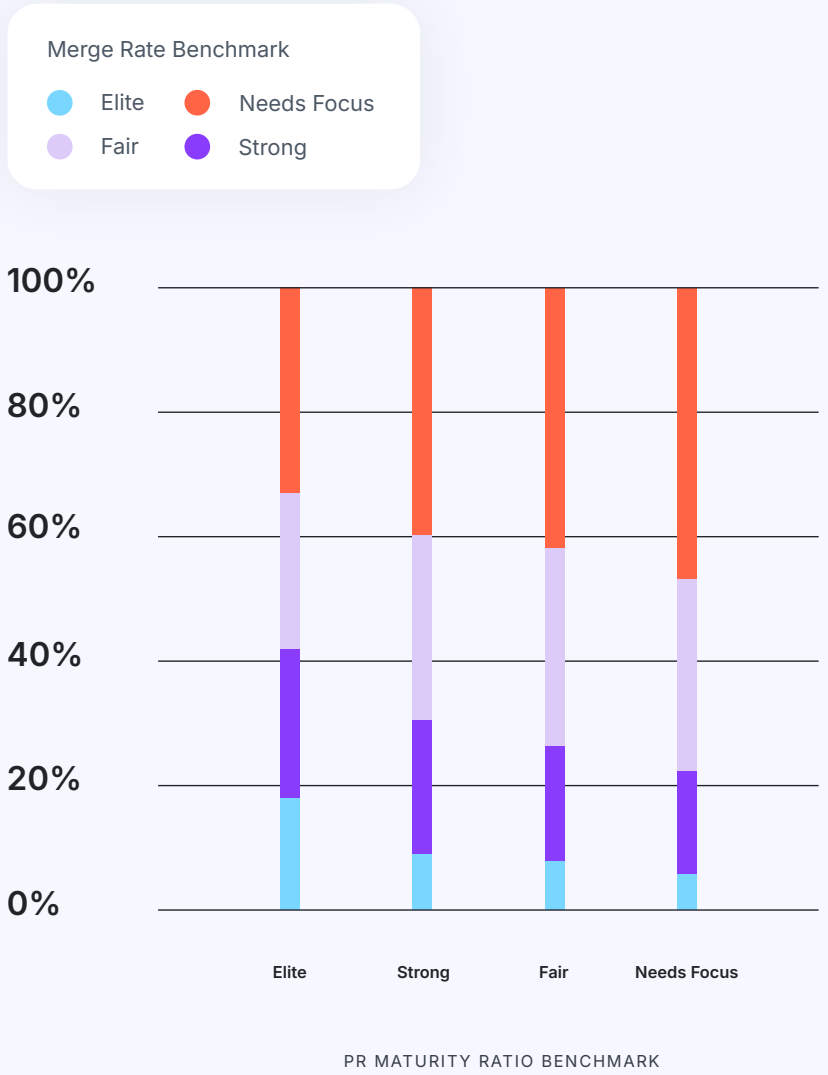
INSIGHT NO. 1

The higher the PR Maturity Ratio, the higher the Merge Frequency.

> **PR Maturity**
> The ratio between the total changes added to a PR branch after the PR was published and the total changes in the PR.

A higher PR Maturity Ratio — marked by fewer changes made after PRs are published — can lead to a higher Merge Frequency, since well-prepared PRs often result in higher-quality code that's more likely to be approved and merged quickly. This increased efficiency allows teams to move more quickly from review to merge, boosting Merge Frequency.

Merge Rate vs. PR Maturity Ratio Benchmark Distribution

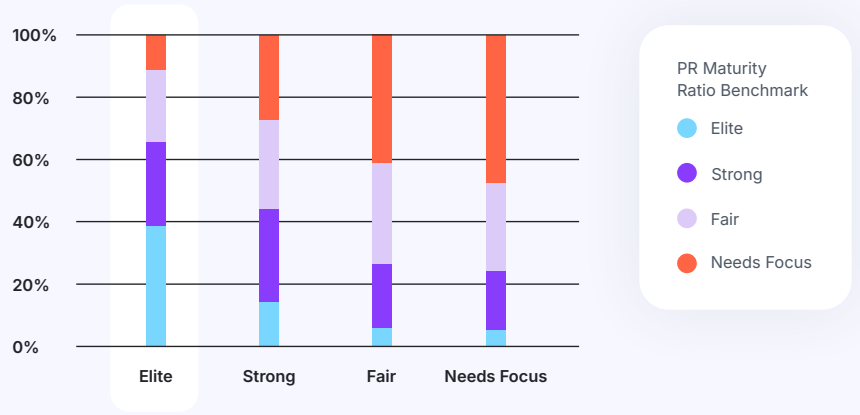


INSIGHT NO. 2

The higher the PR Maturity Ratio, the shorter the Pickup Time.

> **Pickup Time**
> The time a pull request waits for someone to start reviewing it.
> Low Pickup Time represents strong teamwork and a healthy review process.

Interestingly, we see a dramatic correlation here where reviewers prioritize quickly starting to review PRs when the PR Maturity is high. If specific developers author well-polished PRs, reviewers will naturally prefer to review their PRs. Further, reviewers taking an initial look at PRs that don't appear complete or ready will often tend to abandon or push the review to a later time, increasing Pickup Time as that measures the time to the first comment actually made by the reviewers.



PICKUP TIME BENCHMARK

TOOLTIP —— 3 WAYS TO ENHANCE QUALITY AND SECURITY WITH SEI+ AUTOMATION

For teams looking to enable cross-departmental collaboration and improve code quality, here are some tips from the LinearB dev team:

1. Require extra reviewers for complex PRs

Automatically assign two reviewers to those few PRs that are especially complex or affect risky components in the codebase.

2. Auto-approve documentation changes

Proper documentation plays an integral role in facilitating cross-departmental collaboration. We recommend leveraging a workflow automation tool like gitStream to verify and auto-approve all documentation changes.

3. Let DevSecOps take the reins on high-risk work

Flag risky code changes with workflow automation, based on security scanners' output. Automatically pull in DevSecOps as required reviewers for PRs with security implications.

SUMMARY

Org Size Insights

KEY TAKEAWAY

Start-up engineering organizations tend to ship code at a faster rate than Scale-ups and Enterprises.

Insight No. 1

Start-ups have higher Merge Frequencies than Enterprises and Scale-ups.

Insight No. 2

Start-ups have higher Deploy Frequencies than Enterprises and Scale-ups.

Org Size Insights Analysis

KEY TAKEAWAY Start-up engineering organizations tend to ship code at a faster rate than Scale-ups and Enterprises.

Unlike larger organizations, start-ups often work with smaller, cross-functional teams that operate with minimal bureaucracy, leading to quicker decision-making and less friction across the development process. Start-ups typically prioritize speed and market responsiveness over formal process, allowing engineers to quickly test, iterate, and release code. Additionally, start-ups are less likely to face complex compliance, security, or operational constraints, which often slow down deployment in larger companies.

In contrast, scale-ups and enterprises have to manage legacy systems, ensure stringent quality control, and coordinate across multiple teams, which can introduce delays downstream. This freedom from extensive coordination and risk mitigation requirements allows start-ups to adopt an agile mentality, which enables them to ship code at a pace that larger organizations may struggle to match.

> 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."

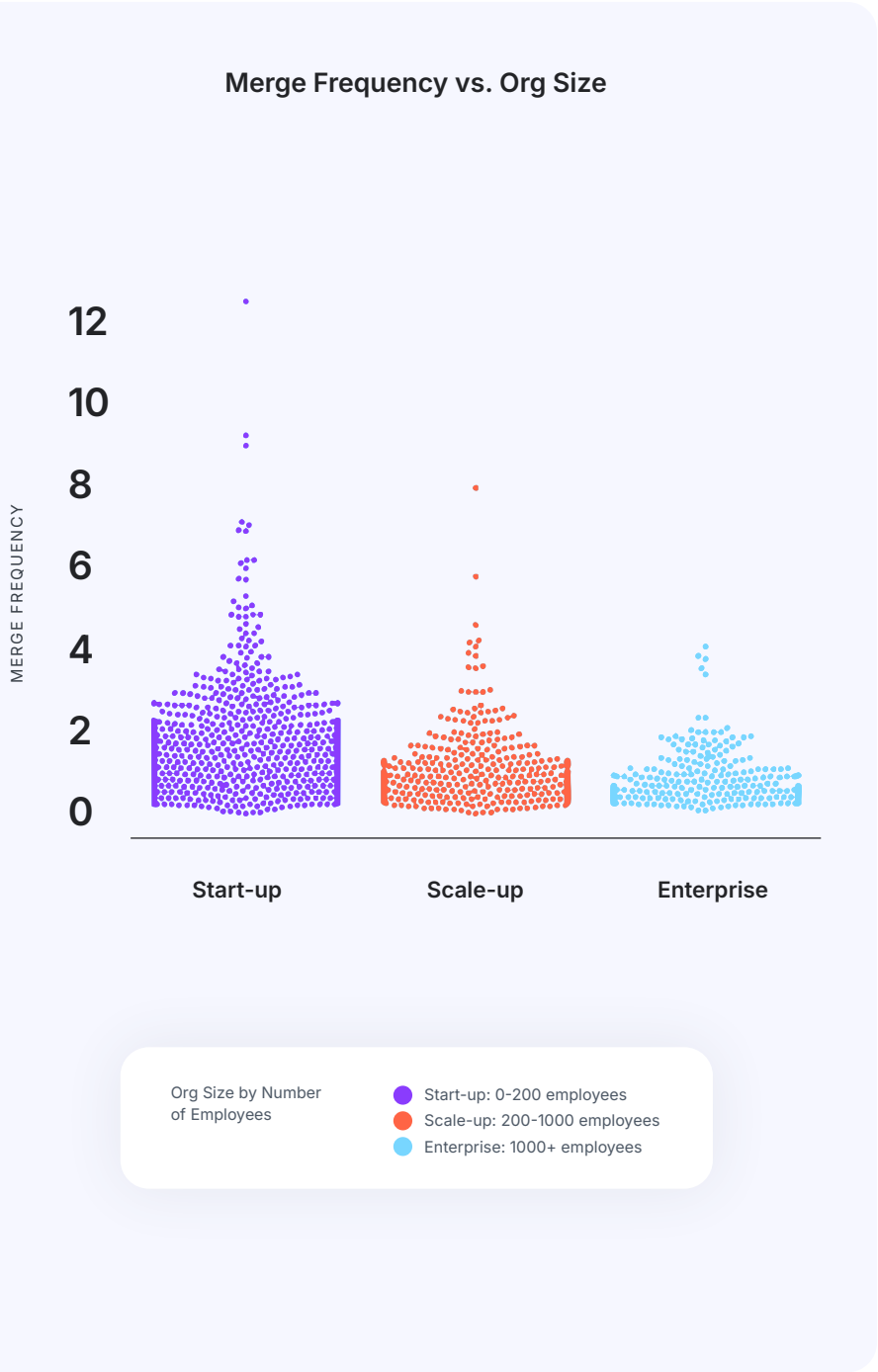
Marko T.
CTO, Assignar

INSIGHT NO. 1

Start-ups have higher Merge Frequencies than Enterprises and Scale-ups.

> **Merge Frequency**
> The total number of pull requests or merge requests merged by a team over a period of time.

Start-ups typically have less formalized processes, allowing developers to submit, review, and merge code changes more quickly to keep up with the high-paced demands of early-stage growth and market fit. The urgency to release features and validate assumptions with real users pushes start-ups to prioritize speed over extensive review processes, resulting in higher Merge Frequencies as they continuously iterate and adapt. In contrast, enterprises and scale-ups tend to have more established code review standards, compliance requirements, and a need for thorough testing due to larger user bases and complex, interdependent systems. This structured environment, though beneficial for stability and risk mitigation, often slows down Merge Frequencies. Start-ups, by necessity, often embrace the famed “move fast and break things” mindset.



INSIGHT NO. 2

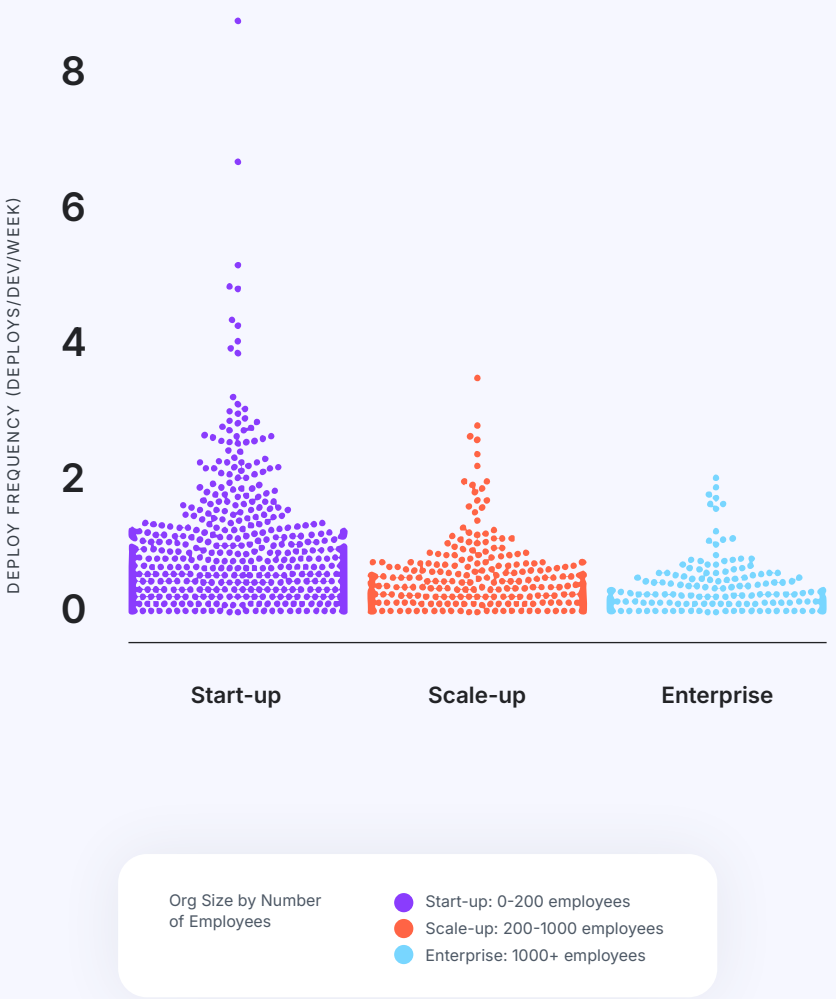
Start-ups have higher Deploy Frequencies than Enterprises and Scale-ups.

> **Deploy Frequency**
> A measurement of how often code is released. Elite Deploy Frequency represents a stable and healthy continuous delivery pipeline.

Start-ups typically have higher Deploy Frequencies than enterprises and scale-ups because their leaner infrastructure and agile methodologies support frequent deployments.

Smaller, cross-functional teams within start-ups often work closely together and can push updates directly with minimal handoffs or delays, resulting in faster deployment cycles. In contrast, enterprises and scale-ups operate within a more complex structure, with multiple teams, dependencies and compliance requirements that necessitate rigorous testing and coordination, which naturally slows down Deploy Frequencies.

Additionally, larger organizations tend to prioritize stability and risk management to avoid disruptions in services for their large user base, often opting for more controlled, scheduled releases. This focus on quality control over speed means that start-ups generally see higher Deploy Frequencies as they prioritize adaptability and speed over the formality and risk management processes more typical of enterprises and scale-ups.



NEW

Bot-Generated PR Research

A study of key open-source repos shows a surge in bot-created PRs, with the share of bot PRs rising from 5% to 15% over the past two years. Similar results appear across thousands of dev teams in LinearB's user base. This highlights a broader shift towards automation in software development.

Vendors like Dependabot and Renovate have pioneered this category. Their acquisitions in 2019 by Github and Mend, respectively, heralded the growing importance of automation in code management and compliance best practices.

A second wave of PR bots is coming, driven by advancements in AI agents and machine learning. The potential for automation to revolutionize code management becomes increasingly apparent. Sometimes referred to as Agentic AI, this future promises a new generation of even "smarter" bots that will be able to automate more complex tasks. Given the rise of AI-generated code, you can likely imagine a world in which the percentage of bot-authored code rises from 15% today to 50% or more in the very near future.

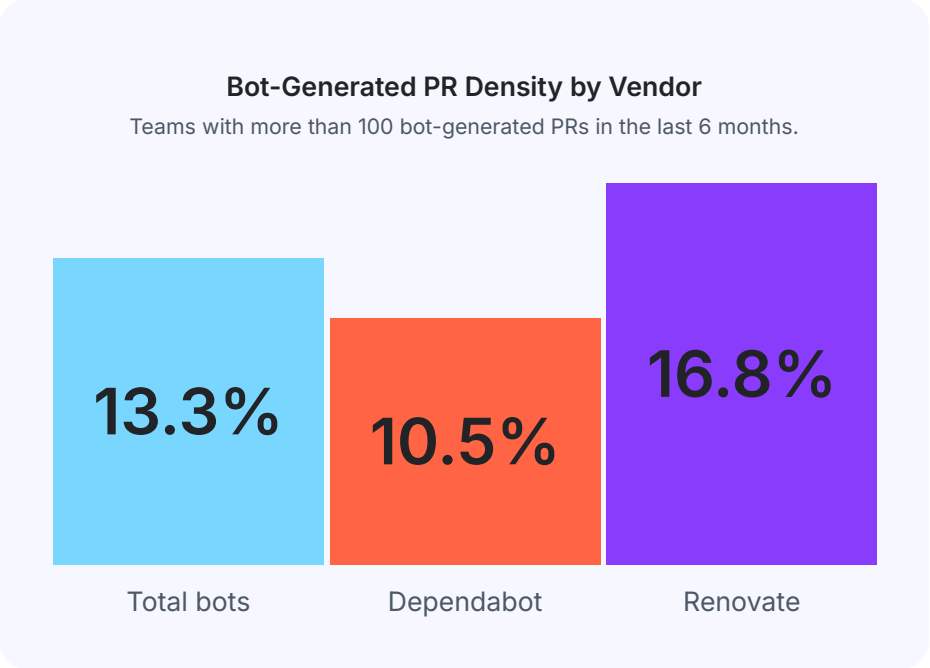
This is why we're including a brand new section in this year's report all about the state of bot usage today.

> "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

INSIGHT NO. 1

Our research revealed that bot-generated PRs make up 13.3% of the average number of code submissions at software engineering organizations that use tools like Renovate or Dependabot. Renovate creates, on average, 16.8% of PRs, while Dependabot creates 10.5%.

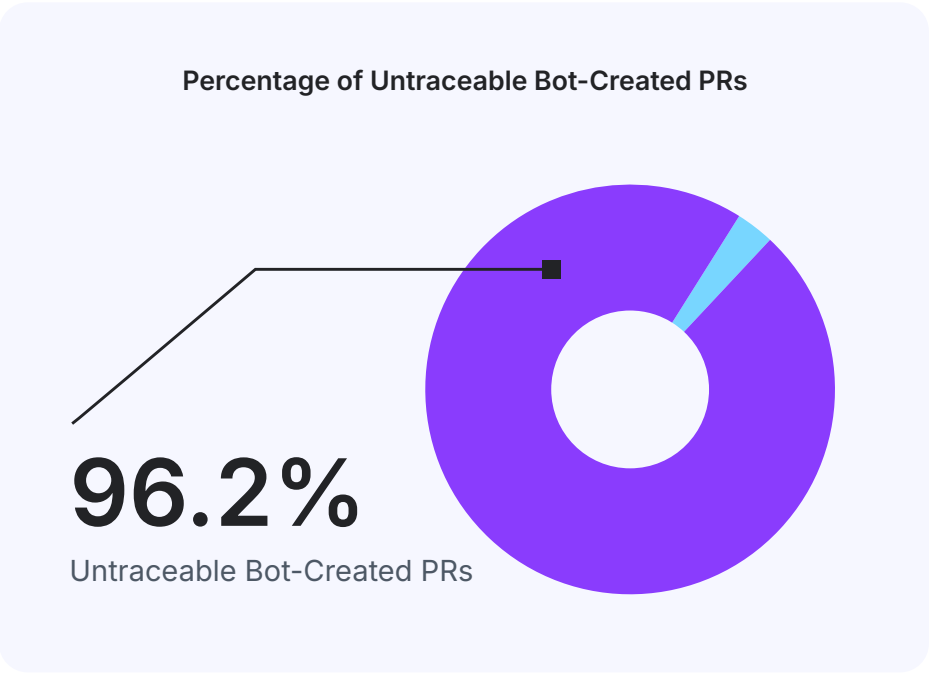
Depending on your engineering org size, this translates to thousands, or even tens of thousands, of bot-created PRs every year.



INSIGHT NO. 2

96.2% of all bot-created PRs are not linked to a PM issue (96.6% for Dependabot and 98.7% for Renovate).

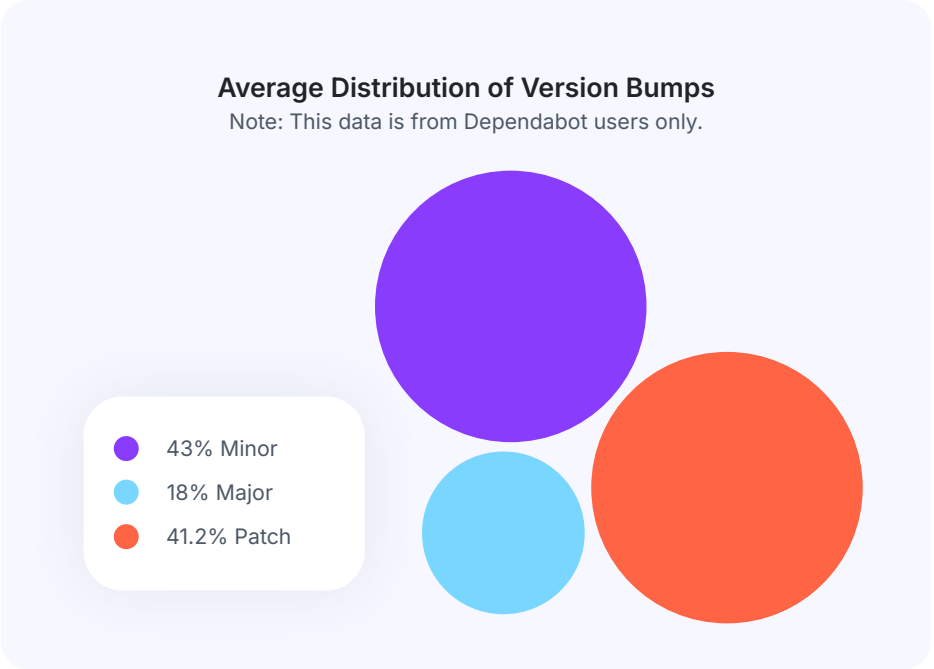
This presents a major risk to PM hygiene, untraceable work, and compliance (e.g. SOC 2, which typically requires that every PR be linked to a Project Management ticket).



INSIGHT NO. 3

The data revealed that 41.3% of all Dependabot PRs are patch updates (which are often rubber-stamp approved, and can typically be safely auto-merged) and 42.96% are minor updates (which can be auto-approved).

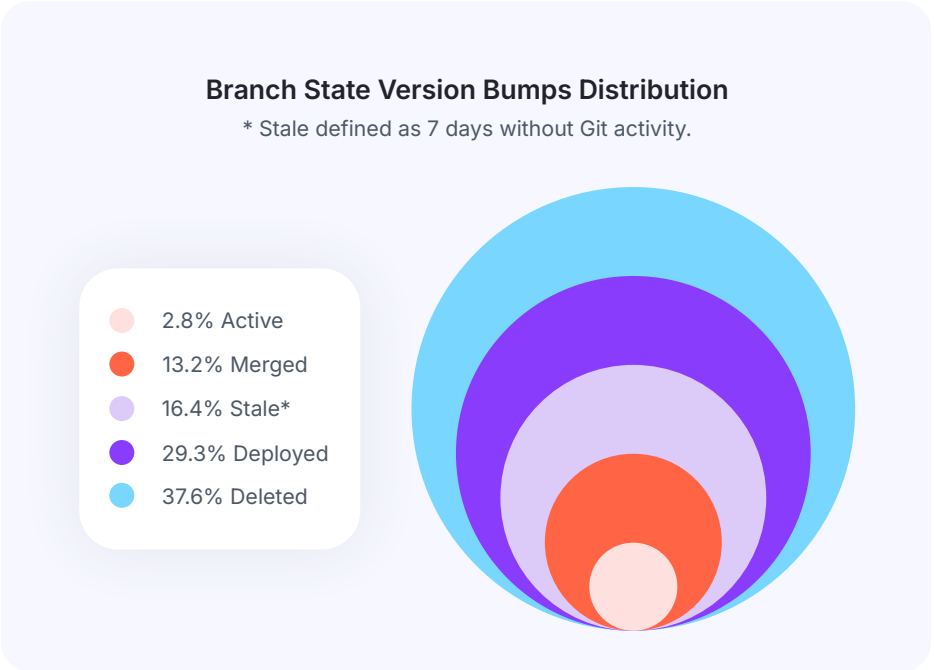
In total, SEI automation can safely auto-approve up to 84% of your bot PRs (patch and minor updates) and auto-merge 41.15% (patches only).



INSIGHT NO. 4

A large portion of these bot-authored PRs are ignored and eventually deleted – no doubt due to their huge volume. Over 37.5% of all Dependabot updates (major, minor & patch updates) are deleted and never acted upon. In addition, some 16.4% of these PRs are stale, likely to be deleted at some point in the future.

This dynamic hints at the toil required to handle these PRs. Even just cleaning them up is work developers shy away from. But the deeper impact of ignoring or stalling these PRs is in the longer periods where the codebase has deprecated or vulnerable dependencies, defeating the entire purpose of the dependency bots.



As a quick refresher, updates to the library dependencies are broken down into the following three categories:

Patch

Patch updates typically address bug fixes and security vulnerabilities. They do not introduce new features or functionality, and are, generally, backward-compatible. For most teams, bot-authored patch updates can be auto-merged.

Minor

Minor updates usually introduce new features or enhancements to existing functionality. They may also include bug fixes, but they tend to focus primarily on new capabilities. In most cases, bot-created minor updates are backward-compatible and can be auto-approved.

Major

Major updates represent significant milestones in the software's development. They include new features, enhancements, and in some cases, substantial changes to existing functionality. They often introduce breaking changes, meaning existing functionality or APIs might not work as before without modifications. Generally, bot-authored major updates need to be thoroughly reviewed and tested before they are merged.

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 any of what was covered in this report in more detail, or to see some of the more advanced features.

MORE INSIGHTS FOR R&D LEADERS

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 DevProd 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

Managing Bot-Generated PRs & Reducing Team Workload by 6%

How dependency management bots are affecting your Developer Experience and Productivity, plus how to leverage PR automation to better manage them.

Download Report

Appendix

Software Engineering Benchmarks (Averages)

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

Category	Metric	Elite	Good	Fair	Needs Focus
DevEx	Coding Time (hours)	< 19	19 - 34	35 - 66	> 66
	First Commit Coding Time (hours)	< 18	18 - 30	30 - 54	> 54
	Ticket Coding Time (hours)	< 36	36 - 75	76 - 123	> 123
	Pickup Time (hours)	< 8	8 - 14	15 - 23	> 23
	Approve Time (hours)	< 15	15 - 27	28 - 45	> 45
	Merge Time (hours)	7	7- 13	14 - 24	> 14 - 24
	Review Time (hours)	< 12	12 - 21	22 - 37	> 37
	Deploy Time (hours)	< 21	21 - 76	77 - 163	> 163
	Merge Frequency (per dev/week)	> 2.25	2.25 - 1.35	1.34 - 0.75	< .75
	PR Maturity (%)	> 91%	91 - 84%	83 - 77%	< 77%
DORA	Cycle Time (hours)	< 50	50 - 90	91 - 156	> 156
	Deploy Frequency (per service)	> 1	1 - 0.5	.4 - 0.15	< .15
	Change Failure Rate (%)	< 1%	1 - 4%	5 - 23%	> 23%
	MTTR (hours)	< 6	6 - 11	12 - 30	> 30
Predictability	PR Size (code changes)	< 194	194 - 339	340 - 661	> 661
	Refactor Rate (%)	< 11%	11 - 16%	17 - 20%	> 20%
	Rework Rate (%)	< 3%	3 - 5%	6 - 7%	> 7%
	PRs Without Review (%)	< 0.7%	.7 - 3%	4 - 13%	> 13%
PM Hygiene	Issues Linked to Parents (%)	> 91%	91 - 72%	71 - 62%	< 62%
	Branches Linked to Issues (%)	> 81%	81 - 64%	63 - 51%	< 51%
	In Progress Issues with Estimation (%)	> 59%	59 - 32%	31 - 20%	< 20%
	In Progress Issues with Assignees (%)	> 97%	97 - 86%	85 - 80%	< 80%

Software Engineering Benchmarks (P50)

Benchmarks by Org | 3,026 Orgs | 6,100,878 Pull Requests | 167,437 Active Contributors | Time metrics in minutes or hours, as noted
* Averages

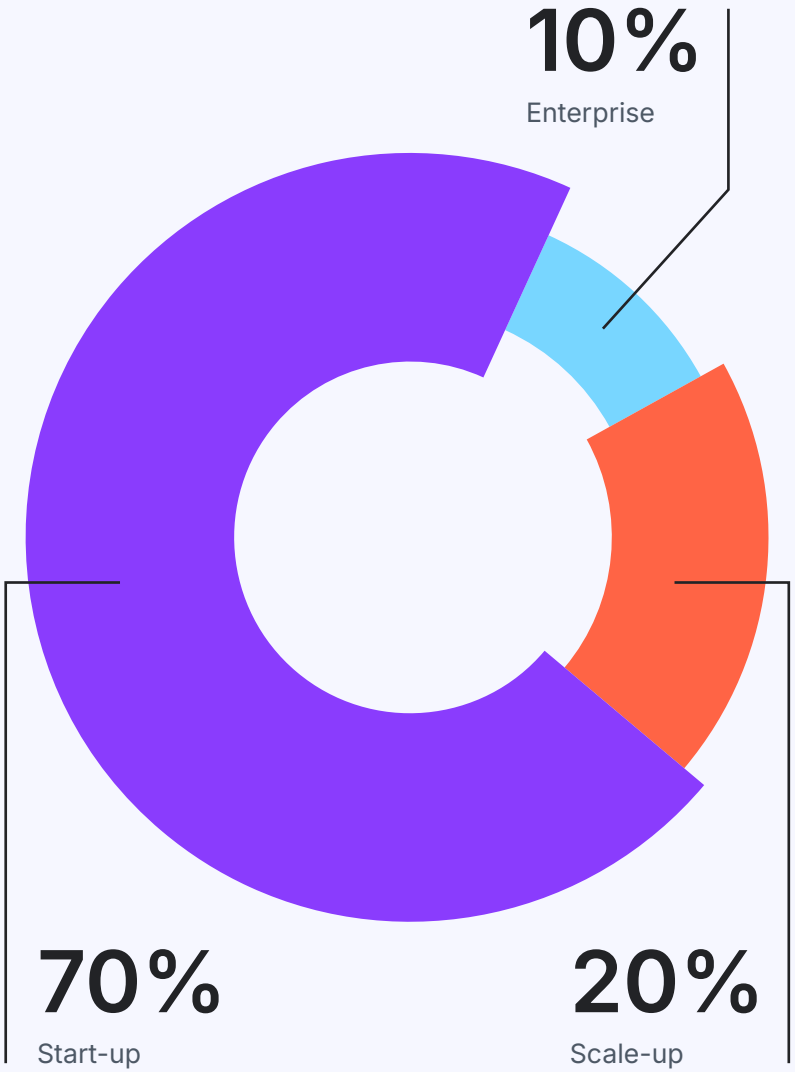
Category	Metric	Elite	Good	Fair	Needs Focus
DevEx	Coding Time (mins)	< 4 mins	4 - 7 mins	8 - 20 mins	> 20 mins
	First Commit Coding Time (mins)	< 4 mins	4 - 6 mins	7 - 14 mins	> 14 mins
	Ticket Coding Time (mins/hours)	< 7 mins	7 mins - 1 hour	1 - 11	> 11
	Pickup Time (mins/hours)	< 12 mins	12 - 26 mins	27 mins - 1 hour	> 1
	Approve Time (hours)	< 1	1 - 3	4 - 6	> 6
	Merge Time (mins/hours)	13 mins	13 - 30 mins	31 mins - 2 hours	> 2
	Review Time (mins/hours)	< 20 mins	20 mins - 1 hour	1 - 3	> 3
	Deploy Time (mins/hours)	< 50 mins	50 mins - 22 hours	23 - 116	> 116
	Merge Frequency (per dev/week)*	> 2.25	2.25 - 1.35	1.34 - 0.75	< .75
	PR Maturity (%)*	> 91%	91 - 84%	83 - 77%	< 77%
DORA	Cycle Time (hours)	< 3	3 - 18	19 - 46	> 46
	Deploy Frequency (per service)*	> 1	1 - 0.5	.4 - 0.15	< .15
	Change Failure Rate (%)*	< 1%	1 - 4%	5 - 23%	> 23%
	MTTR (hours)	< 6	6 - 11	12 - 30	> 30
Predictability	PR Size (code changes)	< 18	18 - 30	31 - 47	> 47
	Refactor Rate (%)*	< 11%	11 - 16%	17 - 20%	> 20%
	Rework Rate (%)*	< 3%	3 - 5%	6 - 7%	> 7%
	PRs Without Review (%)*	< 0.7%	.7 - 3%	4 - 13%	> 13%
PM Hygiene	Issues Linked to Parents (%)*	> 91%	91 - 72%	71 - 62%	< 62%
	Branches Linked to Issues (%)*	> 81%	81 - 64%	63 - 51%	< 51%
	In Progress Issues with Estimation (%)*	> 59%	59 - 32%	31 - 20%	< 20%
	In Progress Issues with Assignees (%)*	> 97%	97 - 86%	85 - 80%	< 80%

Software Engineering Benchmarks (P90)

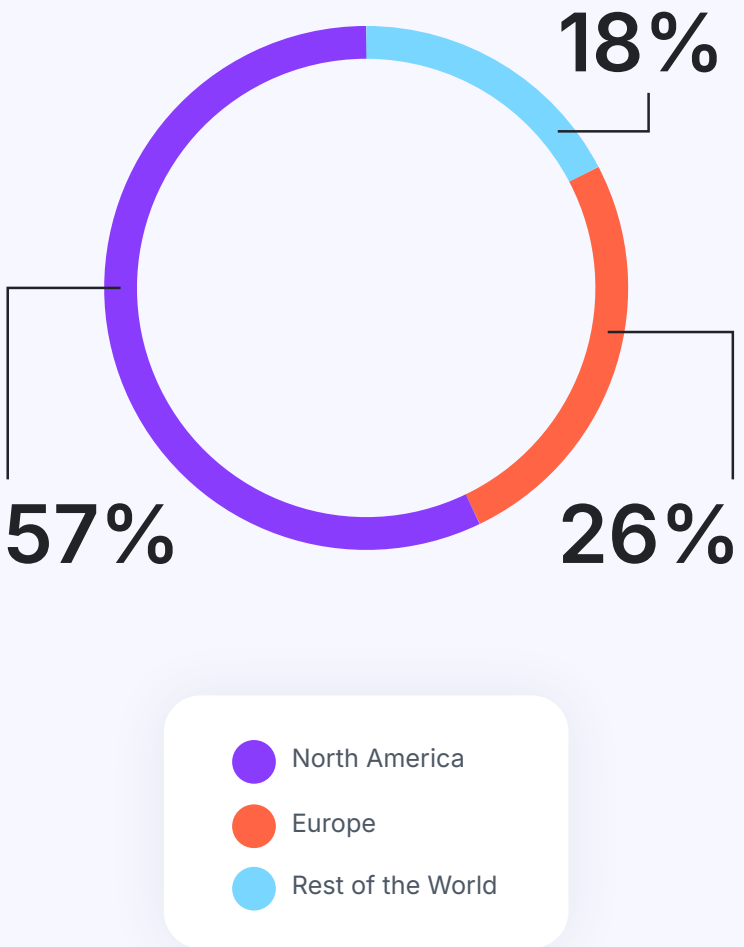
Benchmarks by Org | 3,026 Orgs | 6,100,878 Pull Requests | 167,437 Active Contributors | Time metrics in minutes or hours, as noted
* Averages

Category	Metric	Elite	Good	Fair	Needs Focus
DevEx	Coding Time (hours)	< 26	26 - 72	73 - 146	> 146
	First Commit Coding Time (hours)	< 25	25 - 68	69 - 122	> 122
	Ticket Coding Time (hours)	< 76	76 - 169	170 - 292	> 192
	Pickup Time (hours)	< 16	16 - 25	26 - 63	> 63
	Approve Time (hours)	< 31	31 - 73	74 - 119	> 119
	Merge Time (hours)	16	16 - 23	24 - 64	> 64
	Review Time (hours)	< 21	21 - 50	51 - 96	> 96
	Deploy Time (hours)	< 40	40 - 197	198 - 420	> 420
	Merge Frequency (per dev/week)*	> 2.25	2.25 - 1.35	1.34 - 0.75	< .75
	PR Maturity (%)*	> 91%	91 - 84%	83 - 77%	< 77%
DORA	Cycle Time (hours)	< 119	119 - 218	219 - 410	> 410
	Deploy Frequency (per service)*	> 1	1 - 0.5	.4 - 0.15	< .15
	Change Failure Rate (%)*	< 1%	1 - 4%	5 - 23%	> 23%
	MTTR (hours)	< 6	6 - 11	12 - 30	> 30
Predictability	PR Size (code changes)	< 310	310 - 470	471 - 717	> 717
	Refactor Rate (%)*	< 11%	11 - 16%	17 - 20%	> 20%
	Rework Rate (%)*	< 3%	3 - 5%	6 - 7%	> 7%
	PRs Without Review (%)*	< 0.7%	.7 - 3%	4 - 13%	> 13%
PM Hygiene	Issues Linked to Parents (%)*	> 91%	91 - 72%	71 - 62%	< 62%
	Branches Linked to Issues (%)*	> 81%	81 - 64%	63 - 51%	< 51%
	In Progress Issues with Estimation (%)*	> 59%	59 - 32%	31 - 20%	< 20%
	In Progress Issues with Assignees (%)*	> 97%	97 - 86%	85 - 80%	< 80%

ORG SIZE DISTRIBUTION



REGION DISTRIBUTION



INDUSTRY DISTRIBUTION