What matters most for AI rollouts: How you lead

The actions you take as a leader have the biggest impact on the success of your AI rollout. Our original research shows what we learned and how you can get more from your AI tooling.

multitudes

Executive Summary

Despite 90% AI adoption among software teams, past research on AI's productivity impact has been contradictory. Studies show everything from 56% faster task completion to 19% slower performance.

To understand what's really happening with AI rollouts, we followed 500+ developers over several months while their organizations rolled out AI tooling and other interventions. We then measured the impact on productivity, quality, and wellbeing by combining telemetry data, surveys, and interviews.

Our ultimate goal: To understand how AI impacts outcomes, and what actually drives successful AI adoption.

Top findings

1 Buying AI tooling doesn't guarantee adoption – enablement matters.

Adoption accelerates when organizations invest in enablement.

2 Engineers merged 27.2% more PRs with AI – but did 19.6% more out-of-hours commits.

This is likely because AI rollouts also bring heightened AI expectations – and more delivery pressures.

3 Good code review practices can prevent AI quality issues.

For most, AI had an adverse impact on leading indicators code quality – but this reversed in a few cases, showing that good engineering practices can safeguard against AI-related quality erosion.

4 Find & follow your super-users.

Super-users know how to get AI working best on your codebase, and your other developers want to learn from peers – so set up more peer-to-peer learning to share the knowledge.

Recommendations for leaders

From our research, the top actions you can take to get more from AI are:

- **Set clear expectations** about how you see the role of AI in your org; share why you're rolling it out and what success looks like.
- **Track a holistic set of outcome metrics,** then use that to measure the impact of your different AI interventions, so you can learn and iterate.
- **Make code quality a goal:** Set the expectation that you want to maintain or improve code quality alongside the AI rollout, then measure leading indicators of AI's impact on quality.
- **Do more peer-to-peer sharing**, with a particular focus on your super-users

Research motivation: No consensus on how AI impacts developer productivity

Nearly every engineering organization has purchased AI coding tools. The 2025 DORA Report found that AI adoption has reached 90% among software development professionals, a 14% increase from the previous year. But when it comes to AI's actual impact on developer productivity, the research landscape is contradictory – here's a sample of findings:

- **DORA (2025):** AI increases organizational performance but negatively impacts software stability.
- **METR (2025):** AI tools make developers 19% slower (though when surveyed, they thought they were 20% faster).
- **MIT Study (2025):** 95% of AI initiatives at companies fail.
- **Peking University (2025):** There's a 9% competence penalty for using AI. That splits out into a 13% penalty for women vs 6% for men – and when men who don't use AI review code written by women with AI, they gave a 26% penalty.
- **GitClear (2025):** 2024 was the first year where "Copy/Pasted" lines exceeded "Moved" lines (a proxy for refactoring).
- **Cisco Study (2023):** 26-35% efficiency gains in constrained environments.
- **GitHub Copilot Impact Randomized Controlled Trial (2023):** This study showed that developers had +55.8% faster task completion with AI; it is the most-cited paper in this space.

How can AI make developers both 55.8% faster and 19% slower? Why do some organizations see massive gains and yet 95% of initiatives fail? To answer these questions, we dived into the data ourselves.

From Jan to October 2025, we followed 500+ developers across multiple organizations through their AI rollouts, combining telemetry data, surveys, and interviews to understand the full impact of AI. Our goal was to move beyond the headlines and understand the full picture of what happens when engineering teams adopt AI tools.

Our research focused on three key questions:

1. **Adoption:** What are the adoption patterns for AI tools – which tools, use cases, and what usage frequency?
2. **Impact:** How is AI impacting engineering productivity, quality, collaboration, and wellbeing?
3. **Best practices:** What are the best practices for getting the most out of AI coding tools?

This white paper shares our findings. Even when teams had access to the same AI tools, the results varied widely – showing that it's not the tooling but how your people use it that matters most.

About this research

This research draws on data collected from Jan-October 2025. We gathered data from a range of sources to help us see the full picture – specifically:

- Telemetry data from 500+ developers
- AI usage data from ChatGPT, Gemini, Cursor, Claude Code, and Github Copilot
- Survey responses from 191 leaders and team members with views on AI benefits, risks, and rollout success
- One-hour interviews with 19 engineering leaders, super-users, and skeptics

See appendix for more detail on data sources and methodology.

We're grateful to our spotlight partner organizations who gave their time and insights to be part of this work. Their openness in sharing both what worked and what didn't made this research possible.

PLEO eucalyptus Culture Amp mable

Buying AI tooling doesn't guarantee adoption – enablement matters

Our first research question looked at what drives AI adoption. When we plotted the adoption curves across organizations, it was clear that there's more to it than just buying AI tooling. Depending on an organization's practices, even the same AI tool would have a very different adoption curve.

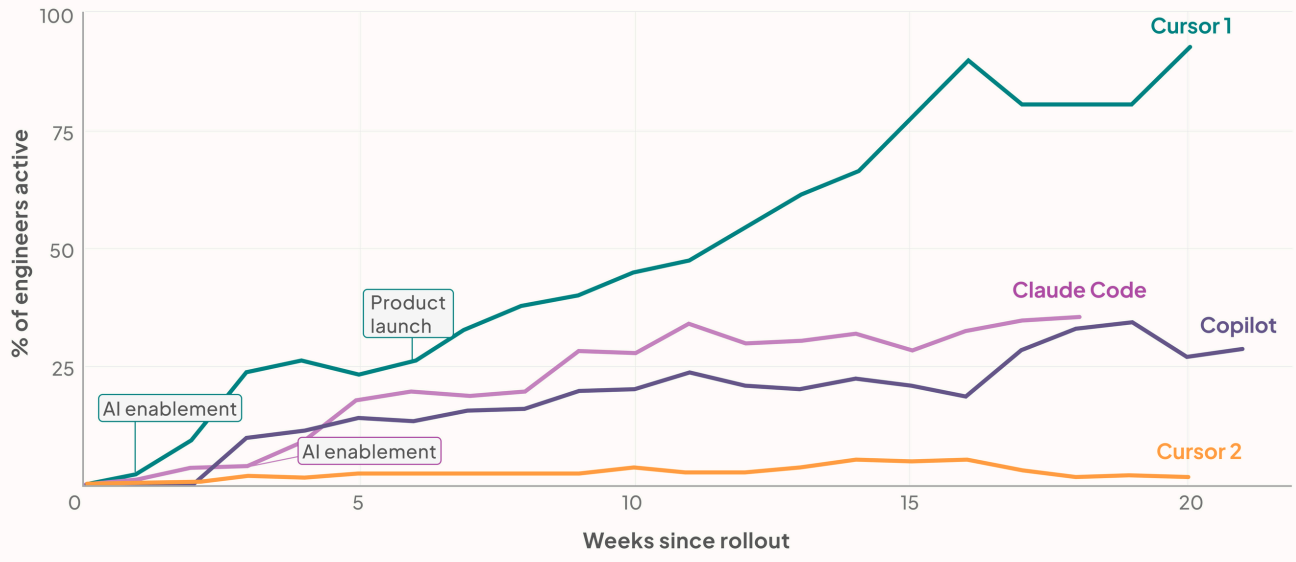
To see this, refer to Figure 1, which shows the AI adoption curves for different tools and organizations. T=0 is the date that everyone in the engineering team was given access to the named AI tool. We've also annotated key dates in the rollout – namely, when organizations launched AI enablement initiatives, and when there was a major product launch.

AI tooling doesn't guarantee AI usage

We can see from these curves that it's not that a specific AI tool guarantees adoption. The fastest adoption curve here is Cursor 1 – but we've added in Cursor data from another organization too, the Cursor 2 line, which shows a much slower adoption pathway. Clearly, it's not the tool that has the biggest impact.

Figure 1: AI coding tool adoption

Daily Active Users, averaged by week



Note on Figure 1: Daily active users (DAU) are measured at the tool level based on AI feature usage. For Copilot, we measure engaged users as reported by the Github Copilot metrics API; for Cursor, users who used at least one AI feature on a given day; and for Claude Code, users who incurred any spending on the platform that day.

Leadership actions determine adoption success

What did speed up adoption? When leaders made it a priority.

You can see this with the "AI enablement" annotations on the charts in Figure 1. The slopes of the adoption curves get much steeper (= faster adoption) after leaders start an AI enablement campaign.

What does AI enablement look like?

These **AI enablement campaigns** included initiatives such as establishing an AI guild (cross-functional groups) that runs workshops, manages dedicated Slack channels for sharing tools and best practices, and might even provide full-time resources dedicated to helping teams identify AI/automation opportunities. At another organization, the company launched their AI push with an all-hands where they shared the vision for AI at their organization and the expectations of engineers.

Set clear expectations

Both organizations with AI enablement initiatives also made it clear that using AI was part of being a developer at their company. As one leader shared, "we needed to mandate AI to see an increase in AI adoption." Call it a "mandate" or "clear expectations"; either way, they were upfront about the role they saw AI play.

As a note, many developers in our interviews had a negative reaction to mandating AI – which is probably why the organizations with AI enablement initiatives didn't mandate how to use AI, but instead made it clear that it was important. Leaving the how open gave team members more autonomy to define what AI usage works for them. In one of these companies, their engineers ended up using AI more for learning something new or summarizing information (the top 2 self-reported use cases) than for writing code (the 4th-highest use case). The other company mandated putting an AI-related learning goal in each personal development plan, but left each individual to define what that would look like – and they specifically chose to make it a learning goal rather than a usage goal because the value of learning is clear.

> "I am skeptical about the need to accelerate AI adoption with any kind of mandate. I think what is missing is not a push to use AI, but [instead] having time to experiment with AI tools outside of delivery pressures."
>
> – Tech Lead

Share the why

A final theme that came out of the survey and interviews was the importance of clarity from leaders about why the org is rolling out AI and what success looks like. Is it to increase developer productivity? Is it to support learning and development, so your engineers will be competitive applicants in their next job (since all engineering jobs are going in the direction of requiring AI skills)? Or is it something else? Without this context, developers are left uncertain about expectations and best practices.

In teams where AI adoption was slower, team members often proactively mentioned the lack of clarity – like this person:

> "We need clarity – what purpose are we introducing AI for? Is it to improve throughput and productivity, or is it because we want to help our staff continue to be employable and maintain their skills? What we need to focus on and measure depends on the goal."
>
> – Engineering Team Lead, organization without AI enablement initiative

In these organizations, engineers weren't sure what to focus on when it came to AI, and it made their learning less focused.

In organizations with Ai enablement, the goal was usually more clear. That said, some engineers still wanted even more clarity, especially around looking past the hype to the underlying problem to solve.

> "I am extremely jaded by all the AI hype. I don't like when companies jump on the bandwagon for something just because everyone is talking about it. Tell me, what problem are we fixing? If it's fixing a problem, then let's get in."
>
> – Senior Software Engineer, organization with AI enablement initiative

This shows a clear opportunity for every leader: Make sure the "why" of your AI rollout is clear, and repeat it until everyone on your team knows it by heart.

Delivery pressure can slow AI adoption

Alongside all these examples of what speeds up adoption, Figure 1 also shows us something that slowed it down: The delivery pressures of a product launch.

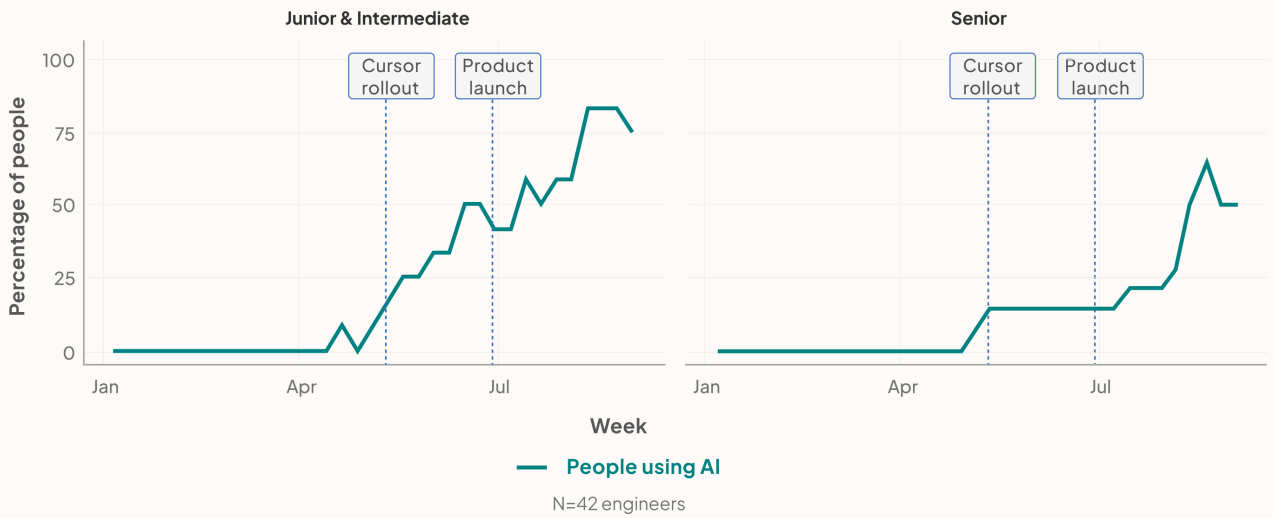
You can see this in the Cursor 1 adoption curve – it flattened in the weeks leading up to a product launch. This wasn't just any launch – it was an all-hands push. 2 weeks before the launch, leadership told the team that everyone would need to push hard to get this shipped on time.

We sliced the adoption data by seniority level and discovered something unexpected: Junior and intermediate developers actually increased their AI usage during this high-pressure period while senior engineers remained flat – so it was actually the adoption slow-down from seniors that drove the flattening of the overall adoption curve.

However, once the launch was done, seniors started picking up AI again.

Figure 2: AI adoption by seniority during delivery pressure

Percentage of people using AI tools by week and seniority



N=42 engineers

Why the difference? In our interviews, we saw a common concern from senior engineers about the hype of AI, and skepticism about whether the productivity impact of AI would be as big as the headlines claim.

> "People start believing that if you add AI, it will increase productivity 30% – but what's real and what's hype?"
> – Senior Software Engineer

Given that concern, we postulate that in the face of tight deadlines, senior engineers relied on the workflows and tools that they already knew worked. AI represented another uncertainty – something that would take time to learn. Under the pressure of a big release, they stuck with what they already knew. Once delivery pressures slowed, senior uptake of AI tooling picked up again.

However, the effort calculation was different for junior and intermediate engineers, who kept adopting AI even during crunch time. Even with the learning cost, AI was helpful for filling knowledge gaps.

Junior engineers spoke about how AI helped them navigate unfamiliar parts of the codebase and get unstuck faster during the high-pressure period. And in fact, having AI as a resource gave them confidence that they were using other people's time better with their asks. (See quotes to the right.)

That seems to be why juniors and intermediates kept adopting AI through this period – because the potential gains of AI seemed bigger than the learning cost.

> "New starters like myself gain a lot from asking Cursor questions like, 'How does the codebase handle X case?'."
> – Junior Software Engineer

> "Asking AI first gives me confidence that I'm not wasting another person's time when I ask them how to fix something. I have more options to fix a problem myself first."
> – Junior Software Engineer

The key takeaway is that people won't have as much time to learn when they're facing big delivery pressures. Of course, sometimes it happens that we're juggling a tooling rollout alongside a big release. In that case, one potential mitigation is to work with your early adopters to show examples of how AI could help people move faster. What we saw in our research is that when AI is seen as something that can help speed up the work, then people are more motivated to learn it even when facing delivery pressures.

Key actions for leaders

From our research, the top actions you can take to get more from AI are:

- Don't agonize over what tooling to use – the important thing is how you support your teams to use the tool.
- Treat AI rollout as a capability-building activity, not just a procurement decision.
  - Adoption won't pick up until leaders make it clear that AI is part of the expectations of the role.
  - Be clear about why AI. Communicate why your organization is adopting AI, what problems you're trying to solve, and what success looks like. Ambiguity and half-measures create friction.
- Ideally you can give your people time for learning, with less delivery pressure – AI is a new skill, after all, and skill development takes time. If you have to roll out AI during a period with a delivery crunch, you'll need to do more upfront to show your team use cases where AI could make their job easier *even with the learning cost*. Otherwise, adoption will be slower until delivery pressures ease.

Engineers merged 27.2% more PRs with AI – but did 19.6% more out-of-hours commits.

Our second key research question was about impact – how is AI impacting productivity, quality, wellbeing, and more?

To assess this, we measured the impact of AI adoption on DORA and SPACE metrics. We looked at standard productivity metrics like Change Lead Time and Deployment Frequency, quality indicators like Change Failure Rate and Mean Time to Recovery, and people metrics like the amount of out-of-hours work, the quality of code review feedback, common themes in code reviews, and more. Most of these metrics varied widely across organizations (which reinforces our previous finding that what leadership does matters more than the AI tool you choose).

However, there were two metrics that showed consistent changes when organizations rolled out AI tooling: Merge frequency and out-of-hours work (see figures 3 and 4).

Figure 3: Trends in Merge Frequency relative to AI rollout

N=372 contributors across 4 organizations
Faded lines show individual averages across orgs
Bold lines show regression trends (blue = pre, orange = post)

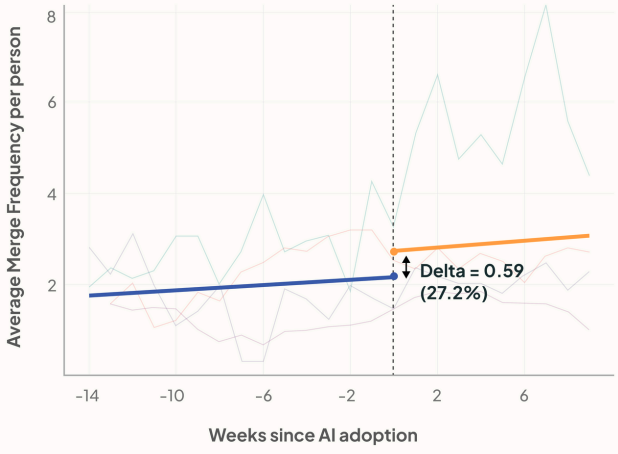


Figure 4: Trends in Out-of-Hours Work relative to AI rollout

N=372 contributors across 4 organizations
Faded lines show individual averages across orgs
Bold lines show regression trends (blue = pre, orange = post)

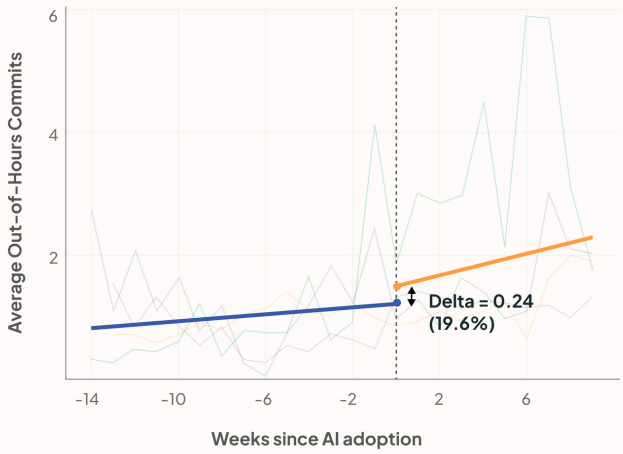


Figure 3 shows what changed with merge frequency: Across 372 individual contributors* and 4 organizations, average merge frequency per engineer rose by 27.2% after AI adoption rose. This doesn't prove that AI caused this change, but the consistency of this shift across diverse teams and codebases suggests that you're likely to get a meaningful increase in the number of code changes when you roll out AI.

Note on Figures 3 and 4: T=0 in these figures is different here from the T=0 in Figure 1. In figures 3 and 4, T=0 is the week where there was a meaningful shift in the AI adoption curve – rather than when the tool was made available or another AI enablement initiative was launched. This is because we don't expect AI outcomes to change until after we see a significant change in adoption.

We fit interrupted time-series regressions across all individuals, with separate trend lines pre- and post-intervention

*While our broader research included 500+ participants across 4 organizations, one organization's telemetry data came via a third party and lacked the granularity needed for individual-level analysis; it was also missing some of the metrics we show here (namely, out-of-hours work). As a result, we brought in an additional organization from Multitudes's customer base to supplement this data. After filtering for active code contributors within engineering, the telemetry metrics analysis includes 372 individual contributors. This supplementary organization's data is included only in the DORA/SPACE metrics analysis, not in the AI adoption surveys or interviews.

However, there's a hidden cost, which we can see in Figure 4: Alongside the increase in PRs merged, was a 19.6% increase in out-of-hours commits. This means that developers were doing more work in evenings and weekends than they did before the AI rollout. These extra hours indicate a human cost, and they also imply that the 27.2% increase in merge frequency is partially because people are putting in more working time. It could be that 19.6% of the merge frequency increase is just from longer hours, which would leave just 7.6 percentage points of change that could be explained by the AI tooling itself.

To reiterate, both the merge frequency and out-of-hours work shifts are relational rather than causal – the timing and consistency of these changes coincide with AI rollouts, but they could be influenced by other factors. One possible explanation is that the AI rollout also brings increased AI expectations, a.k.a. greater delivery pressures on teams. That could be something that pushes people to put in longer hours. Another possible explanation is that AI makes coding more fun and less tedious, so people want to stay late to experiment with new workflows and explore what's possible.

To understand which of these explanations is right, we looked to our qualitative research. Across interviews and surveys, delivery pressures emerged as the clear theme – see examples below.

> "Our team had a reorg and the subsequent pressure of delivering while resetting as a team, plus then you bring in AI. It's been very hard."
>
> – Engineering Manager

> "We haven't had enough time to play, as delivery pressures and the way the team is structured require balancing AI learning with delivery."
>
> – Engineering Manager

It's still possible that some engineers are using AI more out-of-hours because they're excited about these tools – but our qualitative data didn't show that. Instead, the dominant signal from our interviews points to pressure rather than play.

Key actions for leaders

- The most common outcomes from an AI rollout are that people merge more PRs and do more out-of-hours work, with developers referencing delivery pressures as a key driver (this could be contributing to both the merge frequency and out-of-hours work trends). As a leader, be sure to check in with your people through the rollout to see how these changes are impacting them.
- Given the above, it's especially important to track a holistic set of metrics, from productivity and quality to human factors like quality of reviews and wellbeing. This will help you stay across the intended and unintended consequences of increased AI usage.

Good code review practices can prevent AI quality issues.

Another key part of the AI impact assessment is considering how AI impacts code quality. Looking at code quality reveals whether throughput gains are sustainable – or if we're taking shortcuts now that will slow us down later.

As we mentioned above, we looked at the DORA quality metrics – Change Failure Rate (CFR) and Mean Time to Recovery (MTTR). However, there were no consistent trends in these metrics organizations; in some cases, they improved alongside the AI rollout, and in others they got worse. Moreover, in cases where we did see big spikes in MTTR (= it got worse), when we looked into the data with our partner organizations, the failures were often caused by something separate from AI. Ultimately, we concluded that it was still too early to see the impact of AI on CFR and MTTR, because it can take time for issues in the codebase to surface as a failure.

Instead, we needed reliable, leading indicators of code quality – metrics that we could look at now to help us understand if we were moving code quality in the right direction.

We chose two types of metrics to look at, specifically:

- ***PR size:*** Longer PRs are more likely to introduce bugs, have more merge conflicts, and are harder to roll back. When PRs balloon in size, review quality suffers, review wait times lengthen, and the collaboration benefits of incremental development erode (see Microsoft research). This means that an increase in PR size is a warning sign for potential problems later. Google's Engineering Practices Documentation shares more about why small code changes are important; research with Cisco showed that 200-400 lines of code changed is the optimal size.
- ***Quantity and quality of human reviews:*** As Artie Shevchenko lays out in his book, humans are your code health guardians. This role is more important than ever with AI-generated code coming through. So the quantity and quality of human code reviews being given on AI-written code shows how well our teams will do at catching issues in the code changes.

In particular, we saw a pattern with PR size across teams: Typically, PR size increased after AI rollouts. This is not surprising given how verbose AI code outputs tend to be – but as we saw above, it is a warning sign for the degradation of quality in our codebase.

AI slop was also a common complaint in the survey and interviews – showing that the negative human impact is substantial. Developers spoke about how AI slop reduced team trust and their enjoyment at work.

> "With AI, I've seen people raise PRs that have to go through 40 iterations and 100 comments. It's frustrating and time-consuming for other engineers. Plus, you lose a lot of trust from your peers."
> – Tech Lead

The increase in PR size was expected – but what surprised us was when we saw an organization completely buck that trend. Amazingly, developers with high AI usage at this company actually decreased their PR size by 8.5%. Figure 5 shows more: On the left, we see an example of a typical organization, where PR size increases for high AI adopters after AI adoption increased. On the right, we see the data for this organization – where PR size actually decreases for the high AI adopters.

Figure 5: PR Size - High AI Usage Cohorts

Pre/Post = 10 weeks before/after AI intervention
High AI users in org with no standards enforcement: N=41; High AI users in org with standards enforcement: N=17

Note on Figure 5: "High AI usage" is defined based on the AI telemetry data; it's >50% daily active usage of any AI tool, excluding weekends. PR size is displayed on a log scale for interpretability.

We were impressed by this outcome, so we dived into it in the follow-up interviews to learn more.

How did this organization keep PR sizes low through their AI rollout? There were two key things they did well:

1 They had strong code review norms

Their developers cited several code reviews norms they were thinking of when they used AI:

- All PRs require two PRs and they follow the Google Style Guide for PRs (as we saw, smaller code changes is one of the things that Google advocates).
- There was a strong culture of being a good colleague – to not put something forward for a review unless you would want to review it yourself. That means not sending AI slop to your PRs.

> "This organization has optimized for code review as opposed to code output."
> – Senior Engineer

2 The expectations with AI were clear – move faster, but maintain code quality

The big example here was that when the platform team rolled out AI tooling, their leadership explicitly stated that they wouldn't review PRs if they were too long – so developers across the org knew it was important to keep PR size low even while using AI.

With clear cultural norms and goals with AI, their engineers then figured out how to use AI to reduce PR sizes, instead of the opposite. Specifically, they did things like:

- Give AI clear baseline instructions to be concise – even putting it in their rules or markdown file
- Have AI do an initial review before requesting a review from humans, to make sure there was nothing obvious to fix in the PR before asking for feedback. As one dev shared, "I want to make sure I'm not wasting another person's time when I ask for something – it's better to save human input for the harder questions."

The takeaway here is an optimistic one – with the right culture and AI expectations, leaders can point their team in the direction of code quality. Your team is smart, so as long as the goal is clear and they have autonomy, they'll get you to where you need to go.

Key actions for leaders

- When leaders set code quality as a goal and have strong code review norms, they can overcome the quality issues of AI – to scale AI benefits without compromising code health.
- Alongside that, getting leading indicators of how AI is impacting code quality can help leaders stay ahead of issues and make sure their AI interventions are taking the organization in the desired direction.

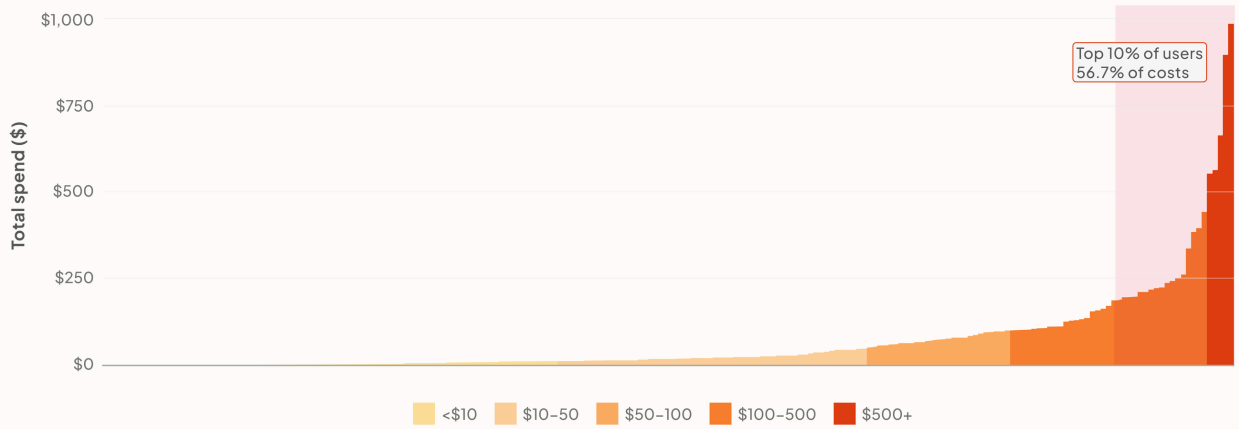
Find & follow your super-users

Our third and final research question was around best practices – for the teams that get more benefits and fewer costs from AI, what do they do differently?

Not surprisingly, AI usage follows a Pareto distribution. Figure 6 shows an example of this with one organization's Claude Code spend (a proxy for usage intensity; the org size is~200 engineers). The chart shows that the top 10% of users account for 56.7% of total costs.

Figure 6: Claude Code Total Spend by Individual

Color-coded by spending tier | Total spend between 2025-07-09 to 2025-10-12



This distribution isn't unique to this organization; we saw similar patterns across every team and tool that we tracked.

Alongside the AI adoption data, we could also see the telemetry data to get a sense of the kinds of outcomes these people were getting. As it turns out, many of the super-users were also high-performers. The question then was: What do they do differently with AI to get more from it?

We share a case study below of one of these users, a Lead Engineer who is a Claude Code super-user. AI practices are changing quickly, so we offer this success story less as a playbook of what to do and more for insights into how to approach AI to get more value from it.

> "I'd love to learn from people who've used AI extensively – how they use it day-to-day, built a feature, or solved a problem. I think that's what we need more of next."
>
> – Staff Engineer

As it turns out, there's also a ton of demand from developers to learn from their peers. Even in organizations where hugely successful AI rollouts, developers proactively spoke about how they'd like more peer-to-peer learning. One honest software engineer was blunt in the interview about having more trust when a peer makes an AI recommendation (full quote below).

> "It's useful to hear about AI use cases from a peer who's not invested in how much money the company makes."
>
> – Software Engineer

There's a natural opportunity here: Super-users are high AI users because they enjoy learning the latest AI practices, and they're already learning what works best on your organization's codebase. (For example, AI still struggles with complex code-bases – but your AI super-users may have developed unique ways to get around this.) Meanwhile, other people on your team want to know the AI use cases relevant to your organization, and would rather learn from peers anyway. So pair them up with time for peer-to-peer learning.

Put together, all of this indicates that creating structured opportunities for super users to share their practices with teammates – such as through dedicated AI guilds, peer-to-peer learning sessions, and super user support roles – can accelerate adoption and strengthen productivity gains more effectively than relying solely on formal training programs or tool purchases without enablement.

Your super-users have already figured out things that the majority will take a few months to get to. Create systems to capture and scale their practices – because making expertise transferable, not buying more tools, will increase the benefits you get from AI.

Case Study: A super-user's practices with AI

One of the top Claude Code users in Figure 5 is JT, a lead engineer who's honed his approach to using AI. His workflow reveals three key principles that help him maintain quality and manages AI risks, while still being able to boost productivity.

(1) Humans are accountable for AI outputs

From JT to other experienced engineers we spoke to, a common theme emerged: Humans, not AI, are still responsible for the work.

Effective AI users don't treat it as a "hands-off" solution. They maintain tight control over the AI's direction and take full accountability for validating AI outputs.

JT put this into practice when he needed to write a "Request for Comment" (RFC) document based on a Slack conversation with a colleague. The colleague had shared what needed to go into the document, so then JT used Claude Code and MCPs to pull in information he needed – with the AWS Knowledge Base to pull in relevant information, and the MCP with Confluence to pull in the RCP template.

AI did the draft, but then JT took responsibility for the work. As he shared,

> "[Claude gave] me a well-referenced document, probably better referenced than what I would create for myself. But I'm the one that's still providing the steering. And I'm also responsible for checking that the quotes exist and that those references are correct."

Ultimately, he got both speed and quality – he had an RFC in 10 minutes, and then made sure it was accurate before sending it on.

JT pointed out that AI doesn't remove the work that a human needs to do, it just changes it:

> "There's this idea that using AI makes things super quick and simple. But in reality, what it more expands is the amount of context that you're able to hold and use and gather and apply towards a particular task. If you aren't prepared to work hard, but in a different way, to get good results, you're gonna get rubbish results."

*As a bonus tip, JT notes that you should turn MCP servers off and on to save tokens. Think about which servers the LLM will need for a task, and only turn those ones on.

Case Study: A Lead Engineer's AI Workflow

(2) Avoid context rot through progress tracking

AI models have limited context windows that can become "rotted" with irrelevant information over long sessions (for example, this research found that LLM performance degrades significantly as additional context is provided). Advanced AI users develop systems such as "context pruning" (removing outdated or conflicting information) to maintain continuity across tasks while keeping the AI system focused on what matters.

JT noticed context rot in his own work –

> "If you don't clear your context often, your context will get poisoned by all of the things it's mucking around with. What you really want to do is take the actual things that have worked, summarize those and get rid of that other stuff and help it forget."

His approach to managing this context rot bakes in this lesson: he uses two markdown files (progress.md and lessons.md) to track progress and learning. These markdown files serve as a memory state for the AI tool across different work sessions.

> "No plan survives first contact with an LLM. The key thing I've baked in is having a way of keeping progress through the tasks, but also keeping memory of its lessons learned."

This approach lets him avoid context rot while maintaining continuity:

> "I'd say to Claude: 'Pick up this task, here's the task spec, here's the overall design. And here are the lessons that you've learned from previous things.' So you get a fresh context window, but primed with authoritative data from the beginning."

(3) The best use of AI depends on task complexity

JT has learned to calibrate his approach based on task complexity:

- For **simple tasks,** he'll execute in Claude without advance planning – trying out a one-shot approach to see what works.
- If it's a **task with mid-level complexity**, he'll do some planning and iteration, then execute.
- For **large, complex tasks**, he'll invest time into breaking things down and providing more guidance to the LLM. His approach here is: Do a high level plan, split into lower-level chunks, then track progress and lessons learned to put into your memory files (see point 2 above).

Being mindful of which approach to use for the specific tasks saves time and improves outcomes. Overall, we know that breaking smaller tasks down gets better outcomes, so that's why the upfront is so important the more complex the task is.

A few examples of what this looks like for JT with different use cases:

- **Documentation:** "Don't try and write essays with it, but small, focused documentation and keeping the documentation up to date with the code, AI's great for that. So leverage it, because then that makes humans and AI better."
- **Doing research:** "I used to do a lot of research, and put a lot of effort into understanding the whole thing before I started on implementation. LLMs allow me to have that same rigor but not take so much time."
- **On writing code requirements:** He uses AI to iterate on clear requirements and criteria so that tests are directly related to actual use cases rather than made-up scenarios. "I get the AI to write the code and the tests together, and then get the AI to check them both for spec compliance."

Case Study: A Lead Engineer's AI Workflow

The role of super-users in AI rollouts

JT is self-aware that not everyone has the same love that he does for adopting new things:

> "Some people are really change-happy, love to adopt new things, love to try things out, and I'm often firmly in that camp. However, that's not everyone."

Not everyone is a super-user, but most organizations have at least a couple of them – and that's the opportunity for leaders. JT's view is that the best way to support middle and late adopters is with incentives and enablement to support them in their journey – and he's happy to share his learnings as part of that.

As a final note, our conversation with JT was a reminder that this is a very new technology still and most of us feel behind. Despite his sophisticated approaches to using AI (and the telemetry data we have that shows he's a super-user!), JT remains humble about the pace of change:

> "I feel like I'm continually catching up. Do you know what I mean? It's moving so, so fast."

Key actions for leaders

- **Find your super-users and see what they do differently**: Identify and learn from your super-users. Document the specific prompts, workflows, and review practices that work on *your* codebase.
- **Support peer-to-peer learning:** Host regular sessions where high-performing super-users demo their work. Developers trust their peers, and seeing AI work on their actual codebase and their specific problems accelerates adoption.

Getting more from AI depends on how you lead

The bad news is that buying AI tooling doesn't guarantee good outcomes – or even decent AI adoption.

The good news is that what does make a big difference is what you do as a leader. The culture and expectations you set for your people seems to be what make the difference between a successful AI rollout and a lackluster one.

From our research, the top actions you can take to get more from AI are:

- **Set clear expectations** about how you see the role of AI in your org; share why you're rolling it out and what success looks like.
- **Track a holistic set of outcome metrics,** then use that to measure the impact of your different AI interventions, so you can learn and iterate.
- **Make code quality a goal:** Set the expectation that you want to maintain or improve code quality alongside the AI rollout, then measure leading indicators of AI's impact on quality.
- **Do more peer-to-peer sharing**, with a particular focus on your super-users.

And remember – we're still in the early days of this technology. No one has all the answers (not even your super-users!), so the key right now is to experiment and learn as much as you can. As long as you and your teams are trying new experiments, measuring progress, and iterating, then you're heading in the right direction. Good luck!

Appendix:
Data and Methodology Notes

Data Sources

This research draws on four complementary data sources:

1. **Telemetry data** from 500+ developers across multiple organizations tracked from Jan to Oct 2025; the data goes back to months before the AI rollouts to create a historic baseline for the impact analysis.
2. **AI usage telemetry data** from the AI tools implemented across the organizations (ChatGPT, Cursor, Claude Code, Gemini, Github Copilot), covering the period of March to Oct 2025.
3. **A comprehensive survey** on AI rollouts, perceived benefits, and risks from 191 survey respondents. Survey responses were collected from June to Oct 2025.
4. **1:1 interviews** with engineering leaders, super-users, and skeptics. We did one-hour interviews with 19 people to explore insights from the telemetry and survey data. These were conducted from August to Oct 2025.

Note that the organizations were a mix of Multitudes and non-Multitudes customers, to make sure we got a diversity of perspectives.

One research participant shared telemetry data via a third party that lacked the granularity needed for cross-organization impact analysis. To address this, we brought in an additional organization from Multitudes's customer base of a similar size. For this supplementary organization, we had Multitudes telemetry and information on AI rollouts and interventions but their data is not included in the AI adoption analysis, surveys or interviews.

Methodology for Qualitative Analysis

For open-ended survey responses and interview transcripts, we implemented thematic analysis following Braun & Clarke's framework, coding iteratively to surface patterns across both data sources.

Daily Active User (DAU) calculations

DAU are measured at the tool level based on AI feature usage, per these rules:

- For Copilot: Engaged users as reported by the Github Copilot metrics API
- For Cursor: Users who used at least one AI feature on a given day
- For Claude Code: Users who incurred any spending on the platform that day

Defining “High AI Usage”

We define high AI usage differently depending on the data source:

- Telemetry insights: “High usage” is defined as >50% daily active usage of any AI tool, excluding weekends.
- Survey insights: Usage levels are based on self-reported frequency:
  - High Usage: "Daily" or "Multiple times a day"
  - Medium Usage: "Weekly" or "Multiple times a week"
  - Low Usage: “Once a week”, “Multiple times a month”, “Once a month” or “No Usage”

Measuring AI Impact

For the AI impact charts (Figure 3 and Figure 4), we standardized timing across organizations:

- Unlike in Figure 1 where T=0 is the date that AI tooling was made available, in the AI impact charts, T=0 was set at the week where there was a meaningful shift in the AI adoption curve. We used an increase in adoption (instead of tool availability) as T=0 because we don’t expect AI outcomes to change until after we see a significant change in adoption.
- We fit interrupted time-series regressions across all individuals, with separate trend lines pre- and post-intervention

Authors

This research was conducted by:

Lauren Peate

Founder, CEO,
Multitudes

Dr. Vivek Katial

Lead Data Scientist,
Multitudes

We're grateful for the feedback and support provided by Dr. Kelly Blincoe (Associate Professor of Software Engineering, University of Auckland), Dr. Thomas Fritz (Associate Professor, University of Zurich), and Nathen Harvey (DORA Lead, Google Cloud).

For more about our other research and academic relationships, please visit www.multitudes.com/research.

multitudes

multitudes.com