GitKraken

A Practical Guide For Engineering Leaders

HOW TO MEASURE THE IMPACT OF AI CODING TOOLS

→ What we learned from 12 months of AI adoption at GitKraken

INTRODUCTION

When we rolled out AI coding tools across our engineering team, I faced the same question every engineering leader is hearing right now: “Is it working?”

I didn’t have a good answer. Our developers said they felt more productive. Acceptance rates looked healthy. But I couldn’t connect any of that to what actually matters: shipping quality software faster.

So we started measuring differently. We tracked the metrics we already cared about (lead time, code quality, throughput) and segmented them by how actively each team was adopting AI tools. Over twelve months, we learned what signals actually matter, which ones mislead, and how the same metric can mean completely different things depending on context.

This guide shares what we learned. It’s not theoretical. It’s based on real data from our own engineering organization. My hope is that it saves you some of the trial and error we went through and gives you a practical framework for answering that question with confidence.

INTRODUCTION BY

Anastasia Zamyshlyaeva

Anastasia is the VP of Engineering at GitKraken, where she leads teams building developer tools used by millions. A seasoned software executive and co-founder of Reltio, she's spent her career turning complex data into strategic leverage.

CONTENTS

1. THE PROBLEM WITH MEASURING AI IMPACT

You've invested in AI coding tools. Claude Code, GitHub Copilot, Cursor, Codeium, or whatever your team is using. There's now a line item on the budget and an expectation that it's delivering value.

The problem is that the most accessible metrics don't tell you much.

Vendor metrics are incomplete. Acceptance rates, completions per day, lines suggested and tokens consumed: these tell you that developers are using the tool. They don't tell you if that usage is translating into better outcomes. A developer can accept hundreds of suggestions and still ship at the same pace with the same quality.

Volume metrics can mislead. More lines of code isn't inherently valuable. More commits isn't inherently valuable. If AI helps your team write more code that requires more fixing, you haven't gained anything. You might have lost ground.

The real questions engineering leaders are trying to answer.

- Are teams shipping quality software faster?
- Does AI adoption create capacity leverage (same outcomes with fewer resources or more outcomes with the same resources)?
- Is the value delivered by AI tools economically superior to alternative investments such as incremental hiring?

Answering these questions requires looking at outcome metrics, not activity metrics, and understanding how to interpret what you see.

This guide provides a framework for doing exactly that.

2. THE METRICS THAT ACTUALLY MATTER

Measuring AI impact effectively requires looking at four categories of metrics together. No single metric tells the full story.

Speed

How quickly is work moving from start to finish?

Lead time measures the elapsed time from first commit to production deployment. This is your primary speed metric. It captures the entire journey, not just the coding portion.

Cycle time measures the time from work starting to work completing within a specific phase. Useful for identifying where delays occur.

Deploy frequency measures how often you're shipping to production. Higher frequency typically indicates smaller batches and more confidence in the pipeline.

Quality

Is the code holding up, or creating downstream problems?

Code churn (also called rework rate) measures the percentage of code that gets modified or deleted shortly after being written. Some churn is healthy; it indicates iteration. Excessive churn suggests code isn't landing well on the first pass.

Bug work percentage measures what share of engineering effort goes toward fixing defects versus building new capabilities. Rising bug work can indicate quality issues, but context matters.

Defect escape rate measures how many bugs make it to production versus being caught earlier. This is a lagging indicator but an important one.

WHY THESE METRICS?

These metrics align with established frameworks like *DORA* (DevOps Research and Assessment) and *SPACE (Satisfaction and well-being, Performance, Activity, Communication and collaboration, Efficiency and flow)*, but we've organized them around the specific question of AI impact. The goal isn't to track everything. It's to track the right combination of signals that reveal whether AI adoption is helping or creating hidden costs.

Throughput

How much work is the team actually producing?

Commit count provides a raw measure of activity. Useful as a directional signal, not an absolute measure of productivity. A sudden spike without corresponding speed improvements may indicate AI is generating more code without adding value.

PRs merged indicates how much work is making it through review and into the codebase.

PR cycle time measures how long pull requests take from open to merge. Increasing cycle time often signals review bottlenecks or PRs that are too large.

Estimated coding hours approximates active development time based on commit patterns. Helps distinguish between ”more output” and ”same output, faster.”

Efficiency

Where is time being spent, and where is it being lost?

PR review time measures how long pull requests sit waiting for review before being merged. AI tools can create more PRs; if review capacity doesn’t scale, this becomes a bottleneck.

Abandoned PRs tracks pull requests that are opened but never merged. Some abandonment is normal; experiments that don’t pan out. A significant increase might indicate thrashing.

First response time measures how long it takes for a PR to receive its first review comment. Can signal whether a team is well balanced or overloaded.

3. THE AI IMPACT MATRIX

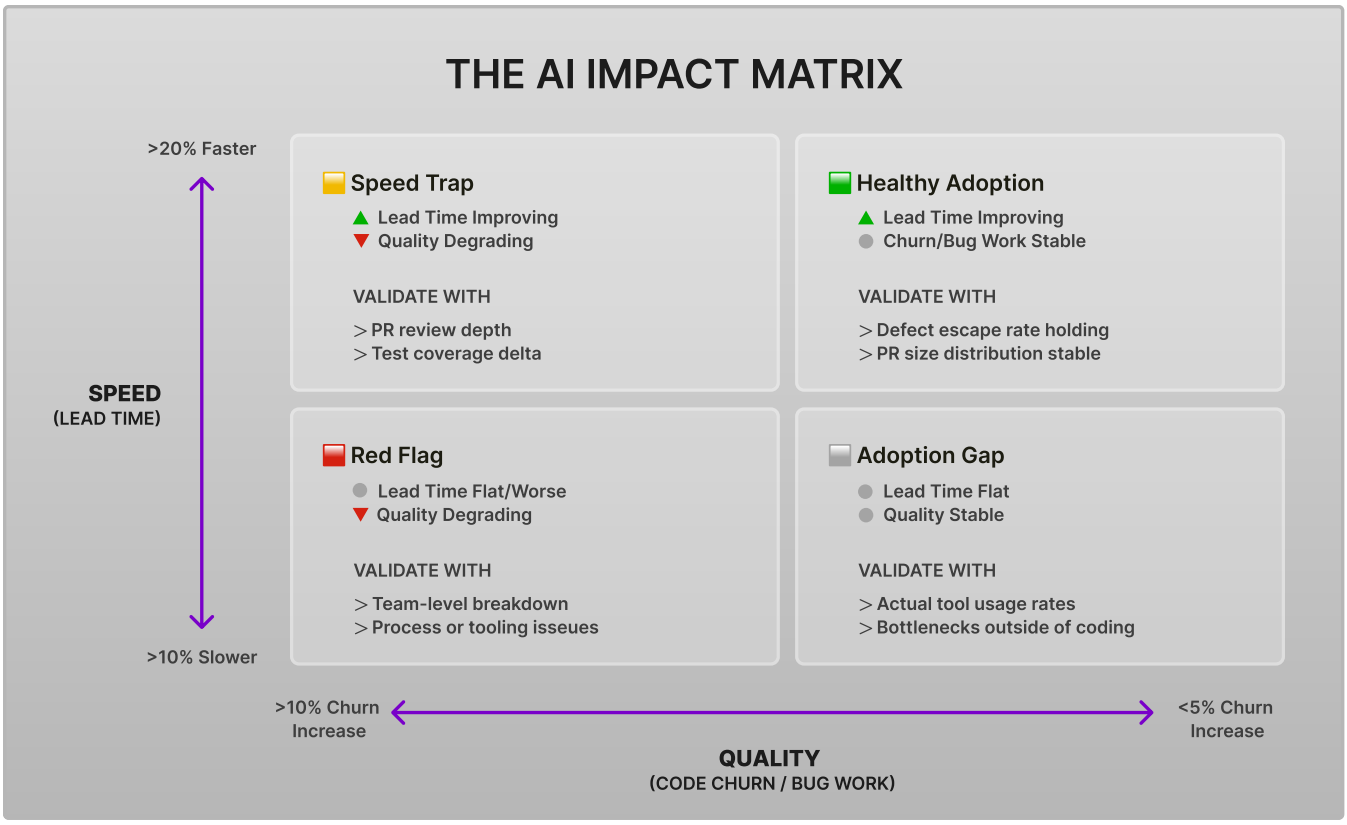
To make sense of what your metrics are telling you, we developed a simple framework: the AI Impact Matrix.

When we started measuring AI impact internally, we found that existing frameworks didn't help us interpret what we were seeing. So we built our own. GitKraken tools are used by millions of developers worldwide, which gives us a unique vantage point on how engineering teams actually work. And this framework came directly from Stasia's experience tracking AI adoption across our own teams. The AI Impact Matrix is the same tool she uses with her team leads today.

The matrix plots two dimensions:

- **Speed (lead time):** Is your team shipping faster, at the same pace, or slower?
- **Quality (code churn/bug work):** Is code holding up, or creating downstream problems?

This creates four quadrants. Each one tells you something different about how AI adoption is landing, and what to look at next.



Healthy Adoption

Lead time improving, quality stable. This is the outcome you're looking for. AI tools are delivering real value without creating hidden costs. Validate by checking that defect escape rate is holding and PR size distribution remains stable. If those check out, document what's working and look for ways to spread that adoption pattern more broadly.

Speed Trap

Lead time improving, but quality degrading. Your team is moving faster, but the code isn't holding up. The productivity gains may be offset by increased rework. Investigate PR review depth and test coverage delta. Are developers reviewing AI suggestions carefully, or accepting too much without scrutiny?

Adoption Gap

Lead time flat, quality stable. AI tools may not be achieving meaningful adoption, or they're being used for tasks that don't impact your critical path. Check actual tool usage rates and look for bottlenecks outside of coding (review capacity, deployment pipeline, upstream dependencies).

Red Flag

Lead time flat or worse, quality degrading. Something is wrong. Look at the data by team to isolate where problems are concentrated. Diagnose whether this is an AI problem, a process problem, or a broader team health issue.

> The matrix isn't about passing judgment. It's about having a shared language for interpreting what the data is telling you. When I show this to my team leads, we can have a real conversation about what's happening instead of arguing about whether a number going up is good or bad.
>
> *-Stasia, VP of Engineering at GitKraken*

4. WHAT THE DATA ACTUALLY LOOKS LIKE

Here's where we share what we actually observed when we measured AI impact across GitKraken's engineering organization over twelve months.

We segmented our teams into three groups based on objective AI tool engagement metrics provided by our AI coding tool vendors. Rather than relying on self-reported usage (which we found to be unreliable), we used quantifiable signals of actual tool adoption and usage patterns to classify teams as High, Medium, and Low maturity. Then we tracked how each group's metrics evolved.

We analyzed data across our engineering organization (multiple teams spanning frontend, backend, and infrastructure), with enough teams in each cohort to observe statistically meaningful patterns. These classifications were based on rolling 4-week windows, which meant teams could move between maturity levels as their adoption patterns evolved over time.

How we isolated AI impact from other factors

Over the course of a year, many things change: teams restructure, processes evolve, and product priorities shift. How do we know the differences between high and low AI maturity teams were caused by AI adoption, not by other factors?

We used cohort analysis with matched controls. Teams in each maturity group were comparable in size, tenure, and type of work (mix of feature development, maintenance, and infrastructure). This wasn't a randomized experiment, but by comparing similar teams over the same time period, we could isolate AI adoption as the primary variable.

All teams were measured over the same time period, experiencing the same organizational changes:

- Same deploy pipeline updates
- Same process changes
- Same product priorities and market pressures
- Same company-wide tooling and infrastructure

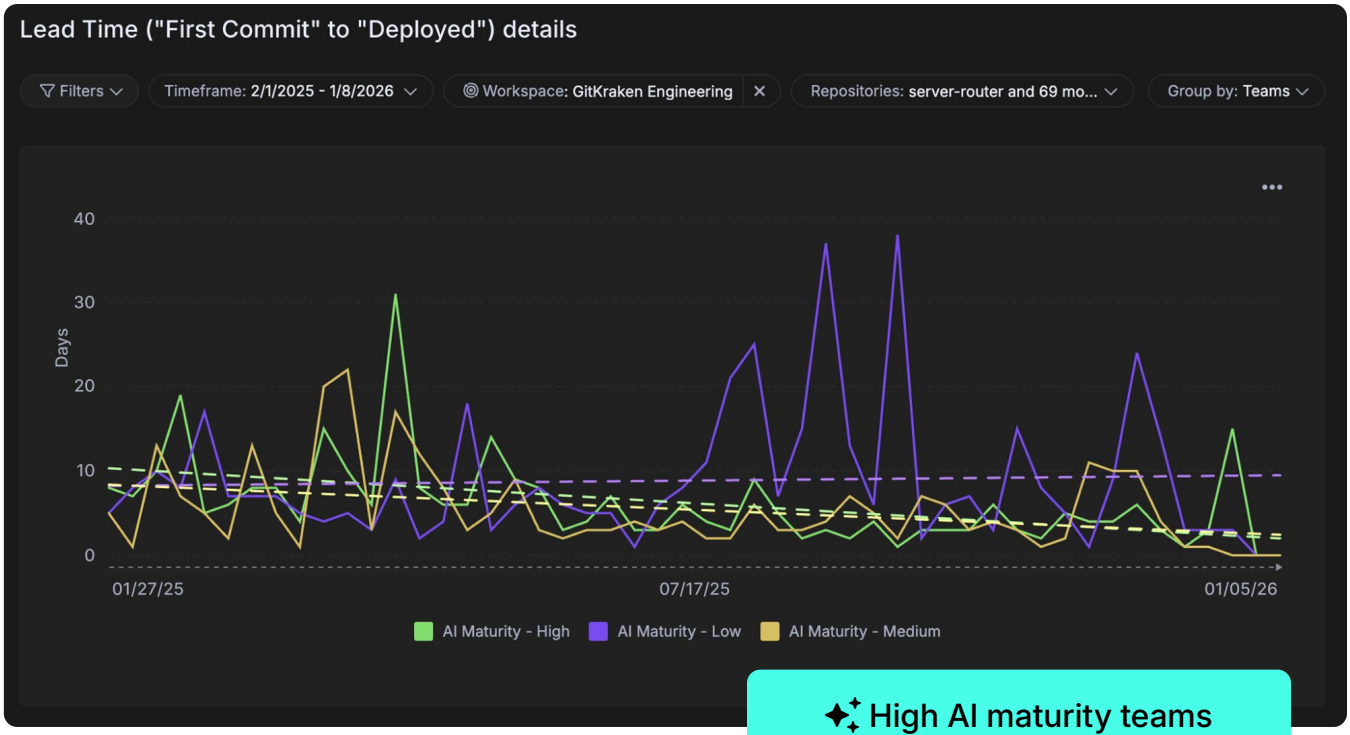
If high-AI-adoption teams improved while low-adoption teams stayed flat - despite experiencing identical external factors - the AI adoption is the differentiating variable.

This doesn't eliminate all confounding factors (high performers might adopt new tools more aggressively in general), but it controls for the most common alternative explanations.

Speed: Lead Time

What we saw: High AI maturity teams reduced lead time by 2-3x over the year. Low maturity teams stayed essentially flat.

This was the clearest signal in our data. Teams that embraced AI tools shipped significantly faster. Teams that didn't saw no change.



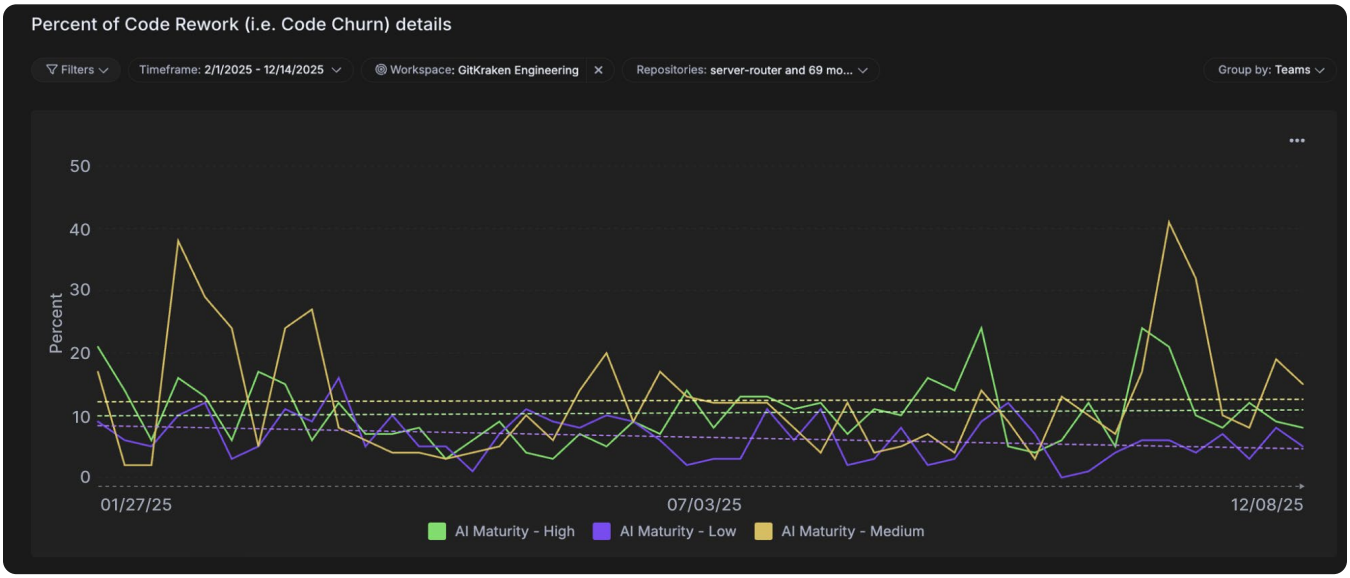
✨ High AI maturity teams reduced lead time by **2-3X**

Quality: Code Churn

What we saw: Code churn for high AI maturity teams increased slightly over the year.

At first glance, this looks like a warning sign. More rework could indicate that AI-generated code isn't holding up.

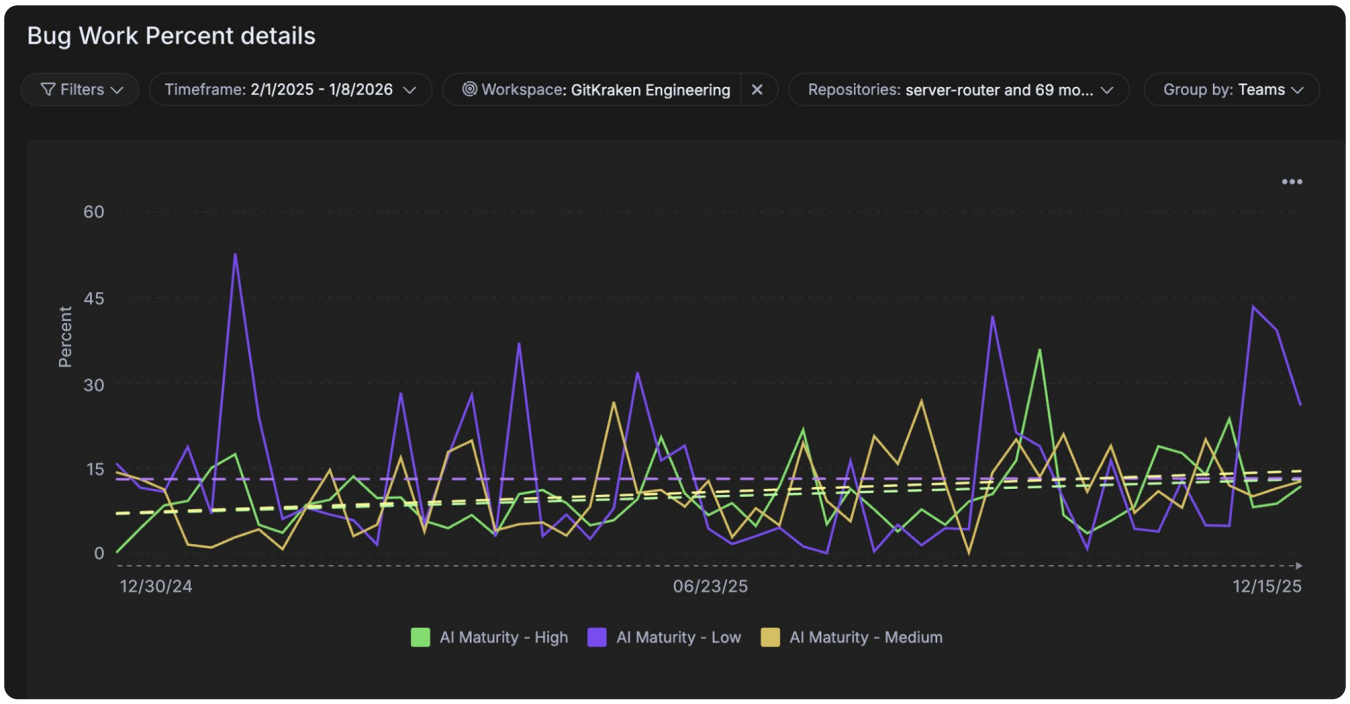
But context matters.



> When I saw churn trending up for our highest-performing teams, I was concerned. But when I talked to the team leads, the picture was different. They weren't fixing broken AI code. They were shipping leaner MVPs faster and then iterating based on customer feedback. The increased churn was intentional. They'd changed their development style.
>
> *-Stasia, VP of Engineering at GitKraken*

Quality: Bug Work Percentage

What we saw: Bug work percentage increased for high AI maturity teams. Again, this requires interpretation. High AI adopters were producing more code overall. More code typically means more bugs in absolute terms, even if the rate of defects stays constant. The question is whether bug work is growing faster than overall output. In our case, it wasn't.

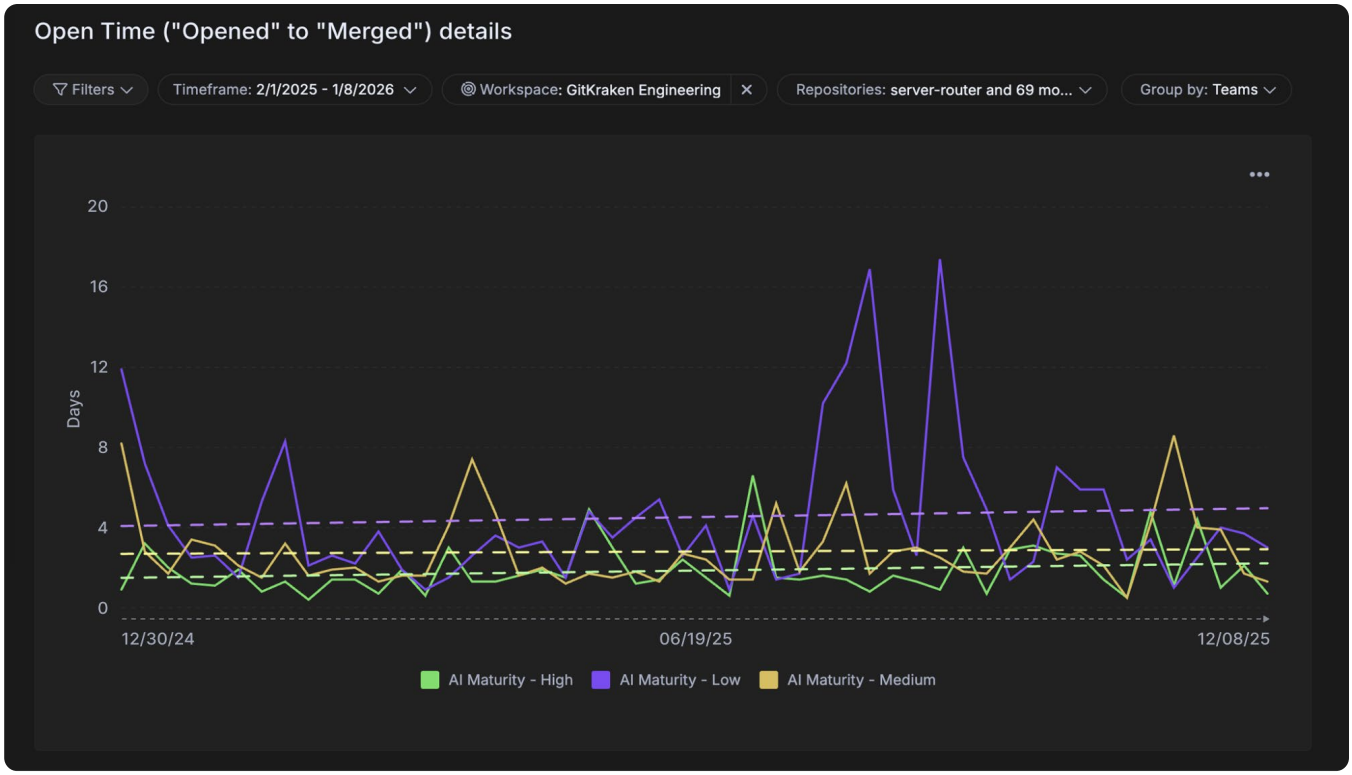


Efficiency: PR Review Time

What we saw: PR review time stayed essentially flat across all groups.

This was unexpected. External research, including the 2026 DORA report, highlights PR overload as a common side effect of AI adoption. Teams generate more PRs than reviewers can handle, and review times increase.

We didn't see that. Our review times stayed flat, which surprised us until we dug into why.



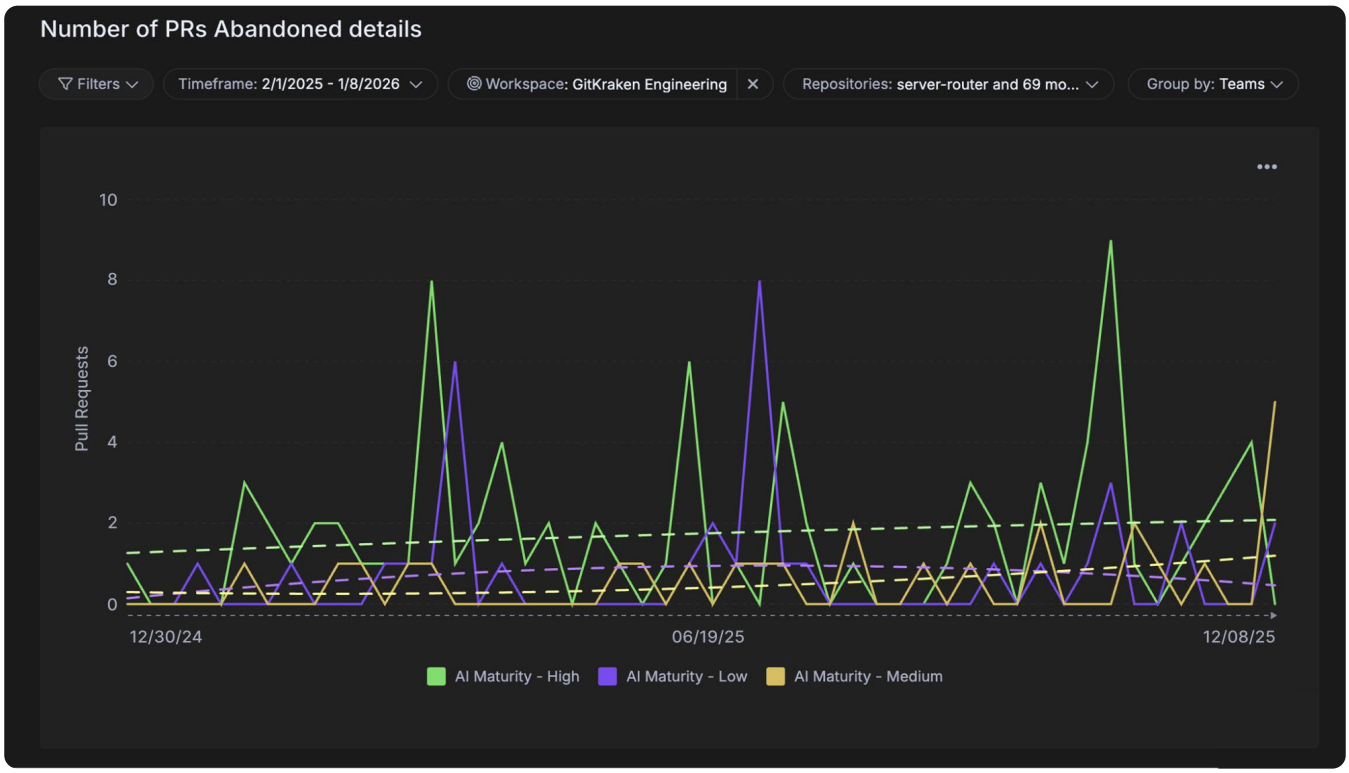
Our review times staying flat was actually a sign of discipline. Teams were keeping PRs small, and authors were reviewing AI output before submitting, not just accepting everything blindly. If we'd seen review times spike, that would have told me we were scaling adoption faster than our review capacity.

-Stasia, VP of Engineering at GitKraken

Throughput: Abandoned PRs

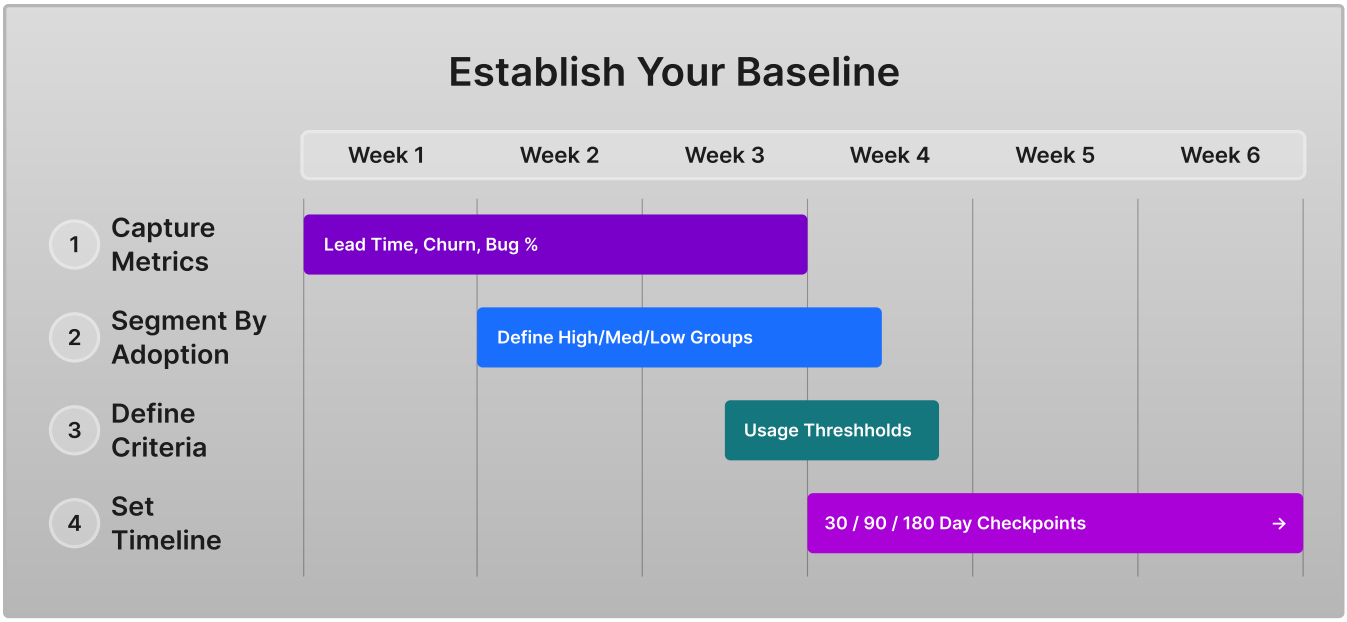
What we saw: High AI maturity teams abandoned more PRs than low maturity teams.

More abandoned PRs sounds negative, but it can actually indicate healthy experimentation. Teams that move fast will try things that don't work out. The cost of starting a PR and abandoning it is much lower when AI accelerates the initial work. As long as the completed work is shipping faster and maintaining quality, elevated abandonment isn't a problem.



5. SETTING UP YOUR BASELINE

Before you can measure AI impact, you need a baseline. Here's how to set one up.



Step 1: Capture current state metrics

Measure your key metrics before rolling out AI tools or increasing adoption. You need at least 4-6 weeks of baseline data, ideally more. The metrics to capture:

- Lead time (first commit to deploy)
- Deployment frequency
- Defect Rate
- Code churn / rework rate
- Bug work percentage
- PR review time
- Commit count / PRs merged

If you've already rolled out AI tools, you can still establish a baseline by segmenting teams or individuals by adoption level.

Compare high adopters to low adopters, as we did.

Critical: Establish cohorts. To isolate AI's impact, you need comparison groups that experience the same conditions. Don't compare Team A's metrics from Q1 2025 to Team B's metrics from Q1 2026 - too many other variables changed. Instead:

- Measure all teams over the same calendar period
- Ensure all teams experience the same organizational changes (process updates, tooling changes, etc.)
- Segment by AI adoption level, not by team characteristics

This cohort approach lets you say: "These teams went through the same year. The primary difference was AI adoption intensity. Here's what diverged."

Step 2: Segment by adoption level

Not everyone adopts at the same rate. To see AI impact clearly, you need to compare groups with different adoption levels.

Options for segmentation:

By team: If some teams are heavy adopters and others aren't, compare them directly.

By individual: If adoption varies within teams, segment by individual usage patterns and aggregate.

By time period: Compare the same team's metrics before and after meaningful adoption began.

The key is having a comparison group. Without one, you can't distinguish AI impact from other factors affecting your metrics.

Step 3: Define your adoption criteria

What does "high adoption" mean for your organization? Define it explicitly. Options include:

- Self-reported usage (surveys, check-ins)
- Tool-provided usage data (if available)
- Observable signals in the workflow (AI-generated commit patterns, etc.)

At GitKraken, we categorized teams into High, Medium, and Low maturity based on a combination of self-reporting and observable usage patterns.

Step 4: Set a realistic time horizon

AI impact doesn't show up overnight.

Expect:

30 days

Early signals, mostly around activity (commits, PRs). Too early for reliable quality or speed conclusions. Focus on validating your data collection and segmentation, not drawing conclusions.

90 days

Clearer patterns emerge. You should start seeing meaningful differences between adoption groups. This is a good checkpoint to identify teams that may need additional support or enablement.

180 days

Full picture. Quality metrics have had time to develop. Trends are reliable. Now you have enough data to make informed decisions about process changes, tooling adjustments, or scaling what's working.

Don't draw conclusions too early. A team's lead time in week two of AI adoption doesn't tell you much.

Office Hours for Engineering Leaders

Measurement is just the start. If you're figuring out how to roll this out at your organization, we've been there.

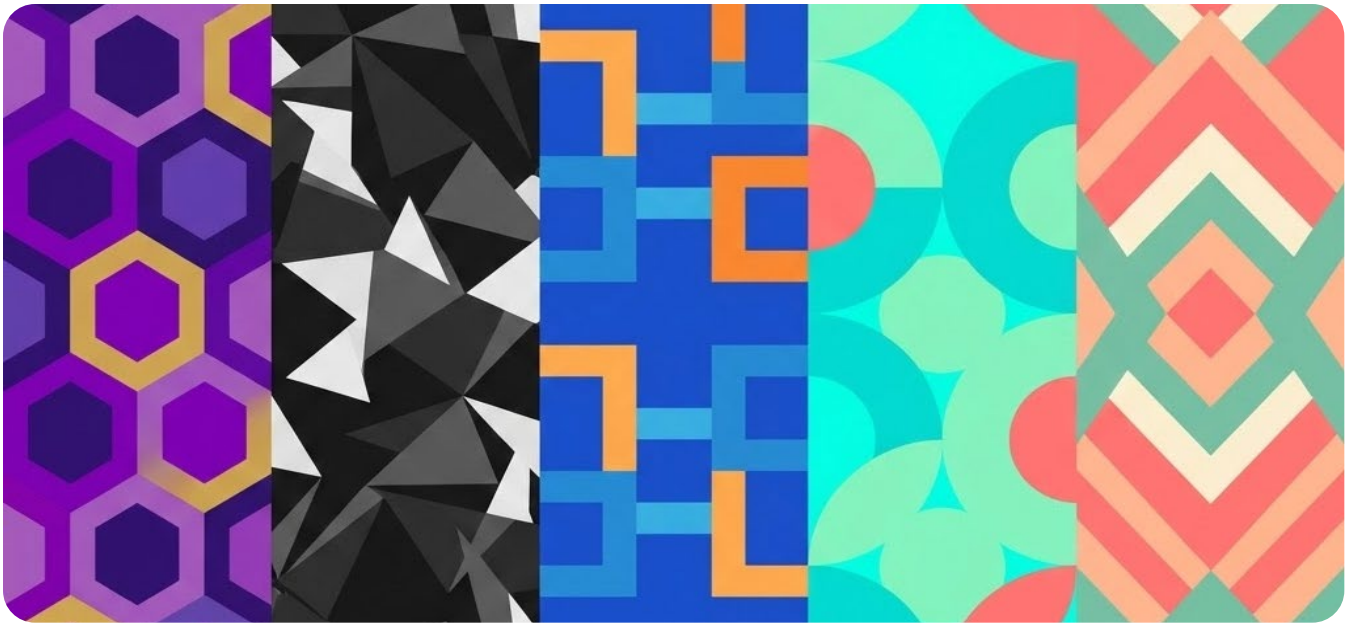
Our team is available for peer conversations with engineering leaders who want to go deeper. No sales pitch, just a practical discussion about what's working for us and how it might apply to your situation.

We can help you think through:

- How to evaluate AI coding tools for your team
- How to quantify impact in a way that resonates with leadership
- How to build the case for budget and resources (and win)
- How to prioritize AI opportunities within your roadmap

Interested? Book a free 30-minute session with one of our engineering leaders.

Book Office Hours



6. COMMON PATTERNS AND WHAT THEY MEAN

Based on our experience and conversations with other engineering leaders, here are the patterns you're most likely to see, and what they typically indicate.

Pattern A: Speed up, quality flat

What it looks like: Lead time drops 20-50%. Code churn and bug work stay roughly where they were.

What it usually means: Healthy adoption. AI is accelerating development without introducing quality problems. This is the ideal outcome.

What to watch for: Make sure quality isn't a lagging indicator. Keep monitoring for another quarter to confirm the trend holds.

Pattern B: Speed up, quality down

What it looks like: Lead time drops, but churn and/or bug work increase meaningfully.

What it usually means: The team is moving faster but cutting corners, intentionally or not. AI suggestions might be accepted without sufficient review. Tests might not be catching issues.

What to investigate: Talk to the team. Are they aware of the quality shift? Is it intentional (shipping MVPs) or accidental (insufficient review)? Look at where defects are concentrated.

Pattern C: Volume up, speed flat

What it looks like: Commits and PRs increase, but lead time doesn't improve.

What it usually means: AI is helping developers write code faster, but the bottleneck is elsewhere. Common culprits: review capacity, deployment pipeline, upstream dependencies.

What to investigate: Look at time-in-stage breakdowns. Where is work sitting? If PRs are piling up waiting for review, you have a review capacity problem, not an AI problem.

Pattern D: Everything flat

What it looks like: No meaningful change in any metric.

What it usually means: Adoption isn't real yet. Developers might have access to AI tools but aren't using them meaningfully, or they're using them for tasks that don't affect your measured outcomes.

What to investigate: Talk to developers. Where are they finding value? Where aren't they? Consider whether additional training or enablement would help.

Pattern E: Speed and quality both up

What it looks like: Lead time improves and quality metrics improve or stay stable.

What it usually means: You're in good shape. The team has found a sustainable way to leverage AI tools that makes them genuinely more effective.

What to investigate: Nothing urgent, but document what's working. What practices or team dynamics are driving the results? Can you replicate them elsewhere?

7. WHAT TO DO WITH THE DATA

Measurement is only valuable if it informs action. Here's how to use what you learn.

Communicating to leadership

When presenting AI impact to executives or finance, focus on:

Business outcomes, not tool metrics. "We reduced time-to-market by 30%" matters more than "Our developers accepted 50,000 AI suggestions."

Trends over snapshots. A single month's data is noisy. Show the trajectory over multiple months.

Honest tradeoffs. If quality dipped while speed improved, acknowledge it. Credibility matters more than spin.

Cost context. Connect the outcomes to the investment. If you're spending $X on AI tooling and seeing Y% improvement in lead time, that's the ROI story.

Knowing when to course-correct

The data should inform decisions. Here's what we actually did when we saw different patterns:

When we saw the "Speed Trap" pattern *(one team's lead time improved but code churn spiked):*

We investigated and found developers were accepting AI suggestions without adequate review. The intervention: We paired them with a high-maturity team for a two-week knowledge-sharing rotation. The high-maturity team demonstrated their practice of reviewing AI output before committing, keeping PRs small, and writing tests first. Within 6 weeks, the struggling team's code churn stabilized while maintaining their speed gains.

When we saw the "Adoption Gap" pattern *(teams with access to AI tools but flat metrics)*:

We ran a survey and discovered two bottlenecks: (1) developers didn't know when to use AI tools effectively, and (2) PR review capacity was the actual constraint, not coding speed. The interventions: We ran brown-bag sessions where high-adoption developers shared their workflows, and we adjusted team capacity to add more review bandwidth. Within 90 days, adoption increased and lead time began dropping.

When we saw elevated bug work percentage alongside increased throughput:

We initially panicked - more bugs seemed like a quality problem. But when we dug into the data, we found that absolute bug counts had increased proportionally with code output, but the defect rate stayed constant. High-AI-adoption teams were simply producing more code overall. We validated this by checking that defect escape rates remained stable and customer-reported issues hadn't increased. No intervention needed - this was healthy growth.

When quality metrics confused us *(code churn trending up for highest performers):*

We talked directly to the team leads. They explained they'd shifted to a "lean MVP → iterate based on feedback" development style, enabled by AI tools letting them prototype faster. The increased churn was intentional refinement, not broken code. This taught us that the same metric can mean completely different things depending on team strategy. We updated our dashboards to track this context alongside the raw numbers.

General principles we learned:

- **Identify teams that need support.** If one team is in the "red flag" quadrant while others are thriving, that team needs attention, not blame. We investigated root causes (training gaps, process mismatch, or tool friction) rathern than assuming the team was “
- **Adjust rollout pace.** If quality is degrading across the board, slow down. Give teams time to build good habits before scaling further. Reallocate investment. If certain tools or approaches aren't showing impact, consider alternatives. The goal is outcomes, not tool loyalty.
- **Celebrate and document what works.** When teams identified sustainable patterns in the "Healthy Adoption" quadrant, we had them present their practices at engineering brown-bag sessions. This accelerated adoption across other teams.

Avoiding the surveillance trap

Measurement can easily become surveillance. Developers are understandably wary of having their output tracked and compared. A few principles to avoid eroding trust:

Measure teams, not individuals. Team-level metrics drive collaboration. Individual metrics drive fear and gaming.

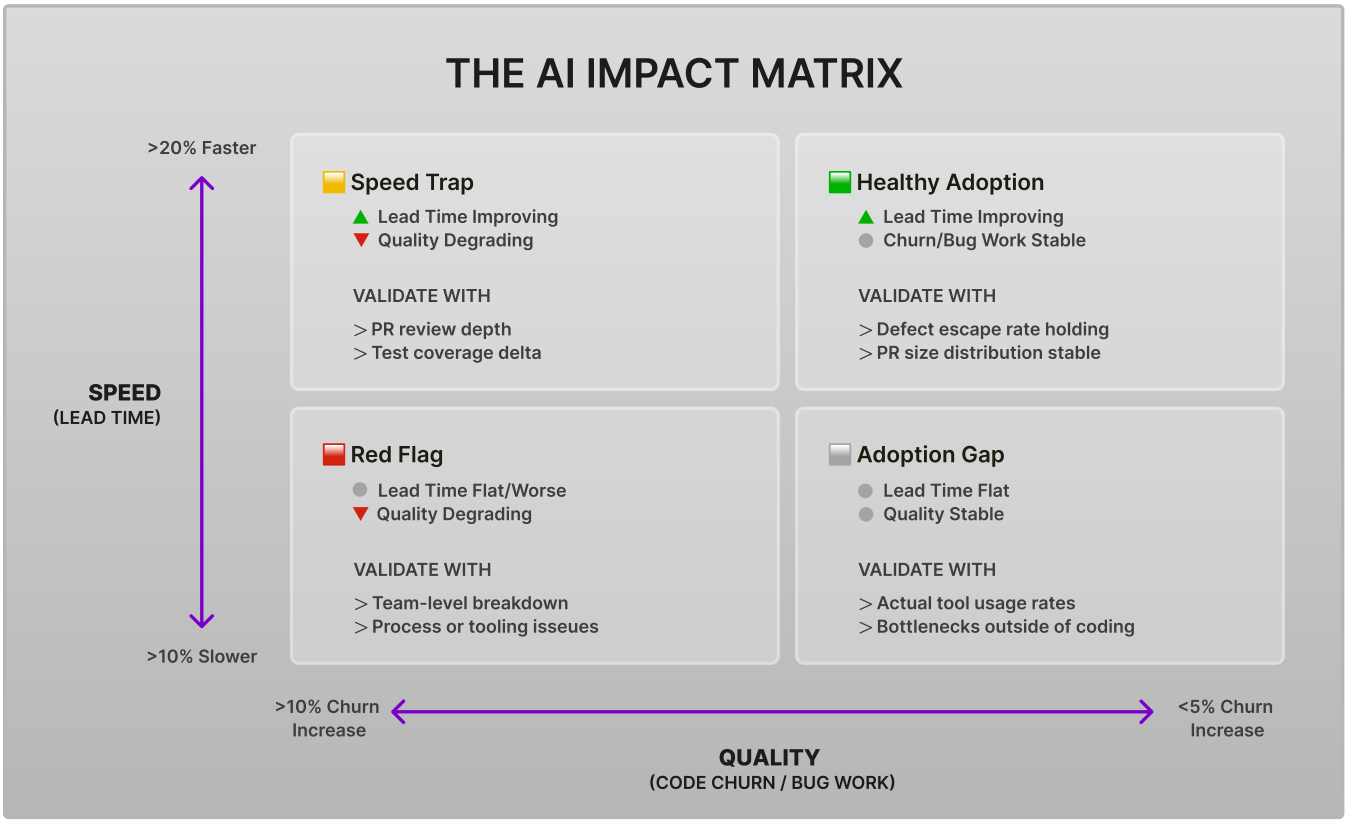
Share the data openly. If you're measuring, let the team see what you're seeing. Dashboards shouldn't be management secrets.

Focus on improvement, not judgment. The goal is to ask "how do we get better?" not "who's underperforming?"

Let teams own their metrics. The best outcomes happen when teams use metrics to improve their own work, not when managers use metrics to pressure them.

> The moment I started sharing these dashboards with my team leads, not hoarding them for leadership meetings, everything changed. They started asking questions, proposing experiments, and taking ownership of their own metrics. That's when measurement became useful.
>
> *-Stasia, VP of Engineering at GitKraken*

SUMMARY: THE AI IMPACT MATRIX



Healthy Adoption (Speed ↑, Quality stable): Keep going. Document what works.

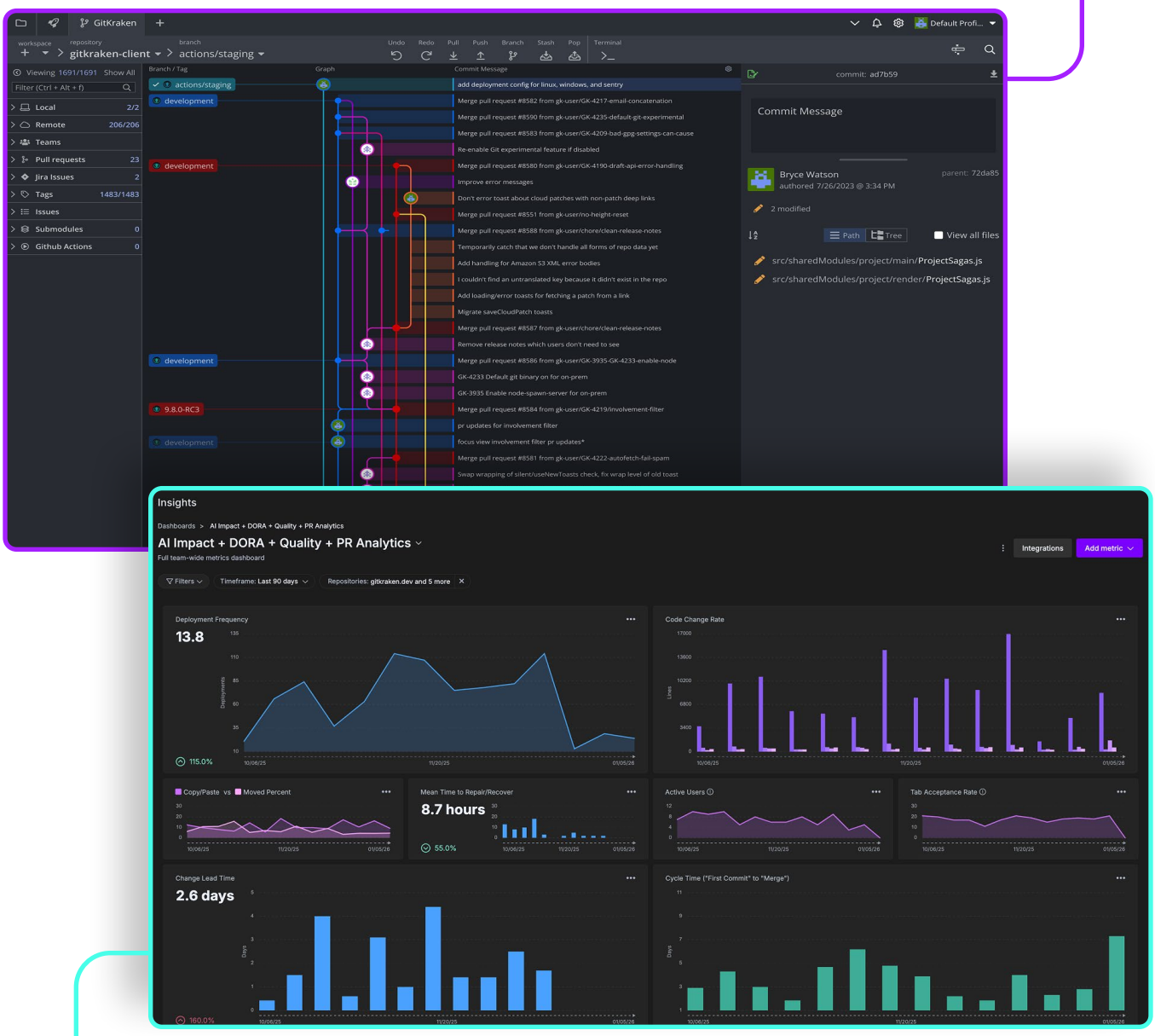
Speed Trap (Speed ↑, Quality ↓): Slow down. Investigate quality practices.

Adoption Gap (Speed flat, Quality stable): Dig into adoption patterns. Find the blockers.

Red Flag (Speed flat/↓, Quality ↓): Pause and diagnose. Something's broken.

About GitKraken

GitKraken is the AI-powered developer experience platform used by over 40 million developers at more than 100,000 organizations, from startups to teams at Microsoft, Google, and Amazon. We build tools for developers and the leaders who support them.



About GitKraken Insights

GitKraken Insights gives engineering leaders visibility into the metrics that matter: lead time, code quality, throughput, and efficiency across your entire development workflow. See how AI adoption is actually impacting your team's outcomes.

Learn More About GitKraken Insights


GitKraken
Learn More About GitKraken