A Guide for Engineering Leaders

QUANTIFYING THE IMPACT OF AI

A Co-Branded Report by
GitKraken and GitClear

GitKraken | GitClear

Table Of Contents

Executive Summary

Engineering leaders face a critical challenge: AI coding assistants are transforming how software gets built, but the true impact remains difficult to measure. While 81% of developers now use AI in their development process and report feeling more productive, new research reveals a concerning reality beneath the surface.

Key Findings

- AI adoption shows positive effects on individual productivity but continues to correlate with increased delivery instability
- 2024 marked the first year where code duplication exceeded refactoring in commits
- Teams saw a 10-fold increase in duplicate code blocks compared to 2022
- 80% of code changes now modify code less than a month old, up from 70% in 2020

The Paradox

Developers feel more productive while code quality and delivery speed decline. This guide provides engineering leaders with a framework to measure AI's true impact, balance productivity with maintainability, and build sustainable development practices in the AI era.

What You'll Learn

- How to measure AI impact beyond traditional productivity metrics
- Why developer experience is your early warning system
- Practical strategies to harness AI benefits while avoiding technical debt
- A roadmap for sustainable AI adoption in your engineering organization

PART 1

The Current Reality

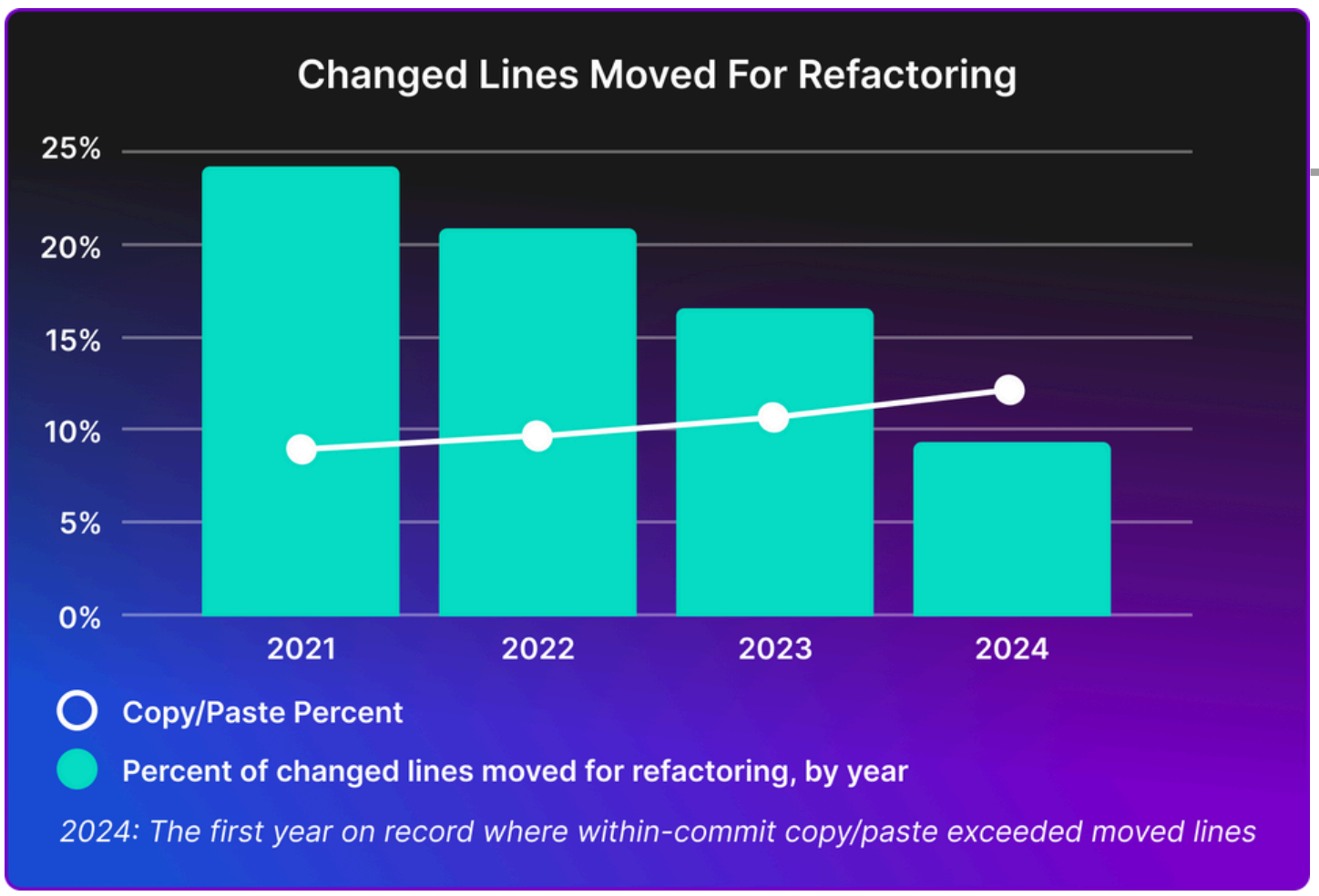
What the Data Shows

The widespread adoption of AI coding assistants has fundamentally changed how code is written. According to Stack Overflow's 2025 Developer Survey, 81% of professional developers currently use AI in their development process, with another 5% planning to adopt soon[1]. The primary motivation? Increased productivity.
But productivity isn't just about writing more code faster.

The 2024 Inflection Point

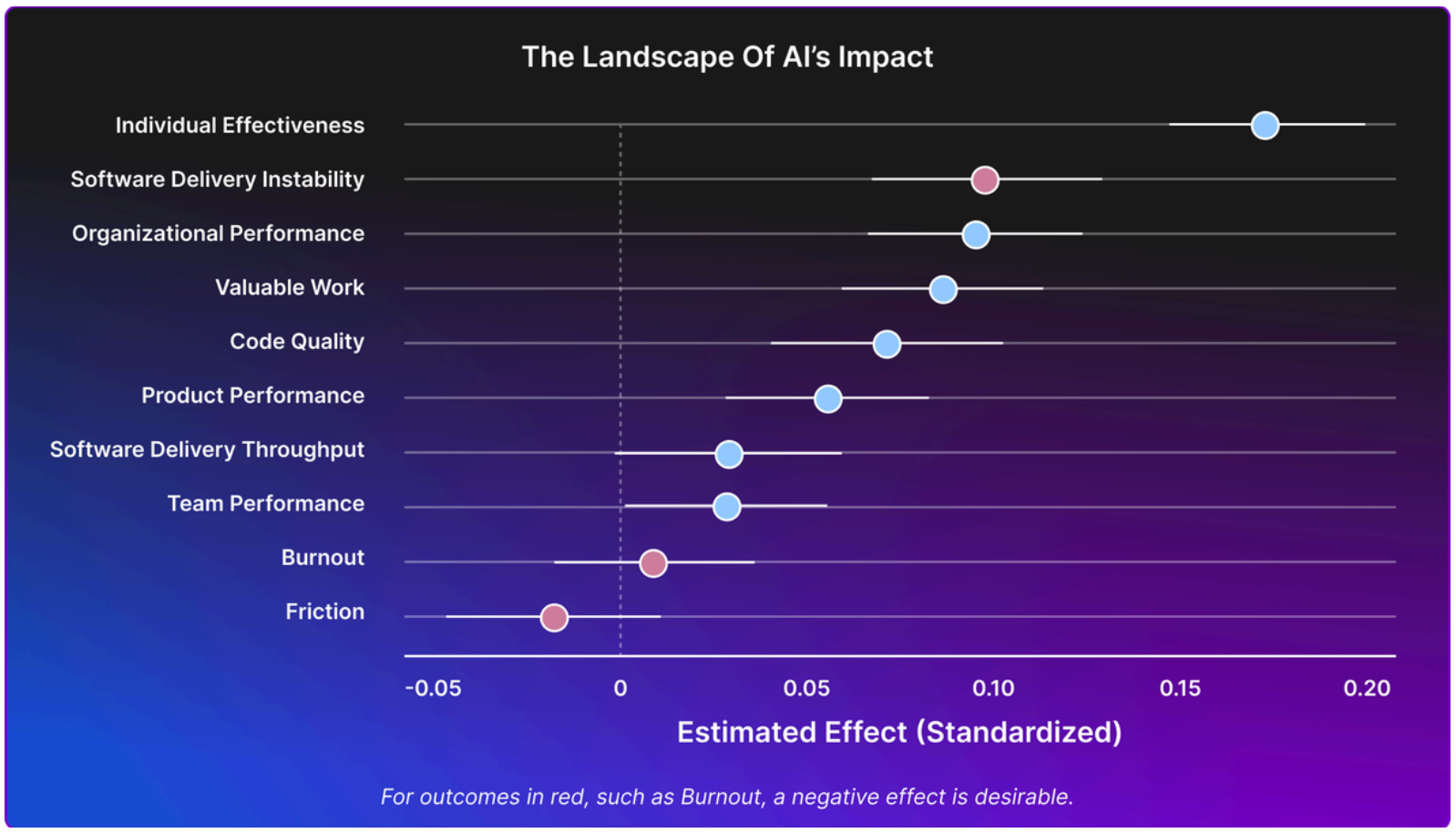
Analysis of 211 million lines of code reveals that 2024 was a watershed year. For the first time on record, the frequency of copy-pasted code exceeded the frequency of refactored (moved) code within commits. This represents a fundamental shift in how developers spend their time:

- **Copy/paste operations:** (The amount of code duplication within individual commits)
  - Increased from 8.4% (2021) to 12.3% (2024)
- **Refactoring operations:** (The amount of code changes that involve moving or reorganizing existing code)
  - Decreased from 24.8% (2021) to 9.5% (2024)
- **Duplicate code blocks:** (The amount of exactly duplicated code in the system)
  - Increased 10-fold from 2022 to 2024



Leadership Insight:
When "productivity" is measured by commit count or lines added, AI will optimize for volume over quality. The path of least resistance is duplication, not consolidation.

1: Stack Overflow Survey



Source: DORA State of AI-assisted Software Development 2025

The Evolving AI Landscape

Google's 2025 DORA research reveals a nuanced picture: AI adoption shows positive correlations with individual effectiveness, code quality, and team performance—yet software delivery instability remains stubbornly associated with increased AI usage[2]. The research suggests this isn't a failure of AI itself, but rather organizational systems struggling to adapt to the new paradigm.

What Your Developers Are Experiencing

The 2025 findings confirm what many teams feel: AI boosts individual productivity and code quality, yet critical systemic issues persist. Developers report higher effectiveness and more time on valuable work, but friction and burnout remain unchanged—suggesting these challenges exist beyond the keyboard, embedded in organizational processes.

The Productivity Perception Gap

After another year of AI evolution, some patterns have shifted positively—software delivery throughput and product performance now show positive correlations with AI adoption. Yet three critical challenges persist:

Three Systemic Costs Teams Face:

1. **Process Friction Unchanged:** Despite AI handling repetitive tasks, workplace friction persists—outdated documentation, inefficient processes, and coordination overhead neutralize individual gains
2. **Stability Degradation:** Increased code volume without evolved guardrails creates verification and coordination costs that current processes can't handle
3. **Expectation Inflation:** As one developer noted: "Stakeholders are expecting more work... deadlines and projects are on a shorter time crunch"

Quick Win: Ask your team: "Has AI reduced your overall friction, or just moved it from writing to reviewing and verifying?" If friction has merely shifted rather than decreased, your processes need evolution.

2: DORA State of AI-assisted Software Development 2025

PART 2

Understanding the Impact

The True Cost of AI-Accelerated Technical Debt

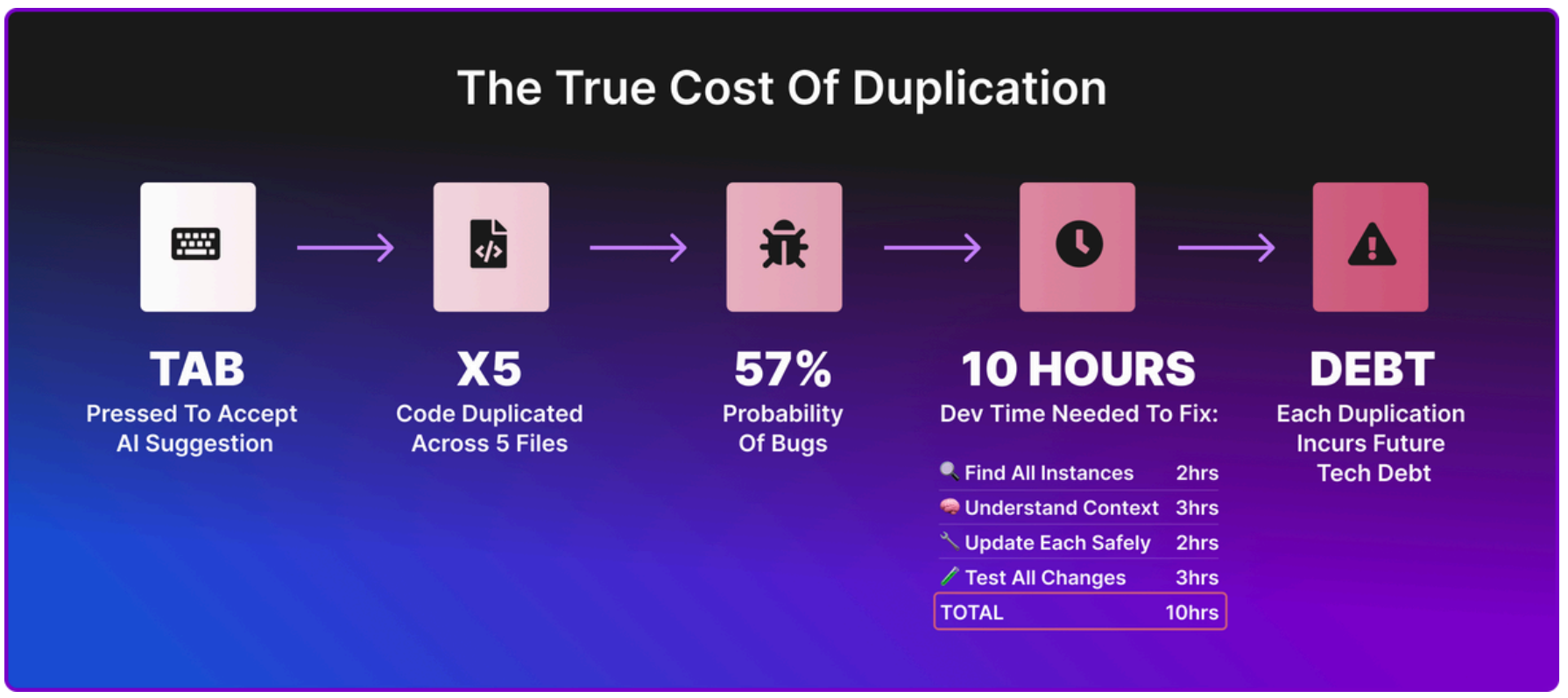
Technical debt isn't new, but AI has fundamentally changed its accumulation rate and nature. Traditional technical debt accumulated gradually through shortcuts and delayed refactoring. AI-accelerated debt accumulates at the speed of a tab key press, and will only increase as we enter the age of agentic development.

The Multiplication Effect

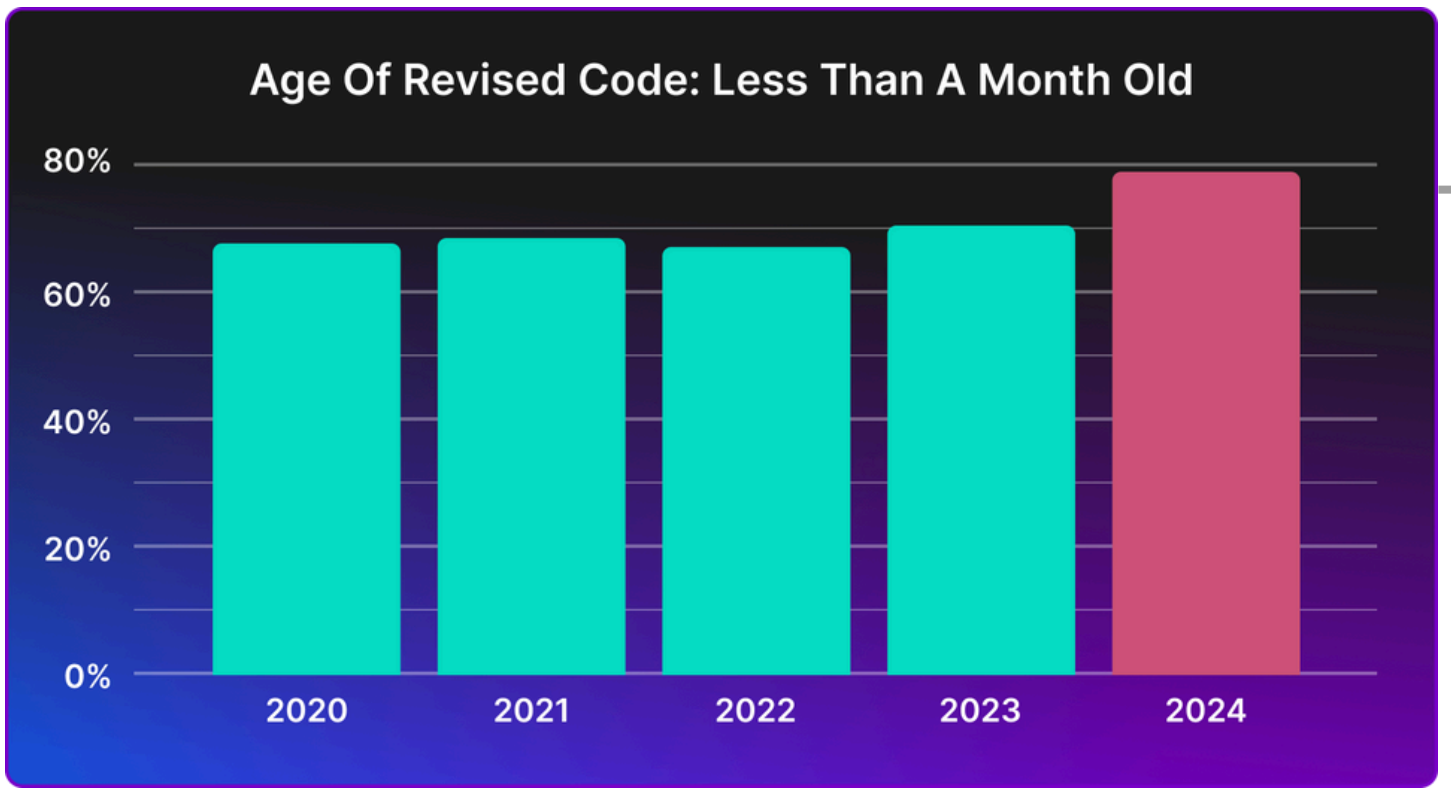
When a code block is duplicated rather than refactored:

- **Immediate cost:** Near zero (one tab press)
- **Ongoing cost:** Multiplied by every instance that needs updating
- **Hidden cost:** Cognitive load on every developer who encounters it

GitClear's analysis of code clone research found that 57% of co-changed code clones are involved in bugs. This means when duplicated code needs fixing, there's better than even odds that not all instances will be correctly updated, leading to subtle, hard-to-track defects.



The true cost of code block duplication is an exponential burden of future maintenance cost. This cost is multiplied each time blocks are blindly duplicated, and this tech debt is likely to be assigned to your senior developers, costing you and your team countless hours of productivity.



There is a worrying trend from modifying older code to predominantly modifying code <1 month old.

From Feature Development to Defect Remediation

The data reveals a concerning shift in how developer time is allocated:

- 79% of modified code is now less than a month old (up from 70% in 2020)
- Only 20% of modifications touch code older than a month
- New code requires 20-25% more revisions within the first month

This pattern could reflect multiple factors: teams fixing recent additions, but also the acceleration of software development with faster release cycles, more experimentation, and iterative feature development. However, when combined with the increase in code duplication and decrease in refactoring, it suggests that at least some of this churn represents quality issues rather than intentional iteration.

The Developer Experience Dimension

Developer experience isn't just about satisfaction—it's a leading indicator of systemic issues that will eventually surface in your metrics.

Why Developers Report Feeling Productive While Quality Declines

The disconnect between perceived and actual productivity stems from how we naturally measure our own work:

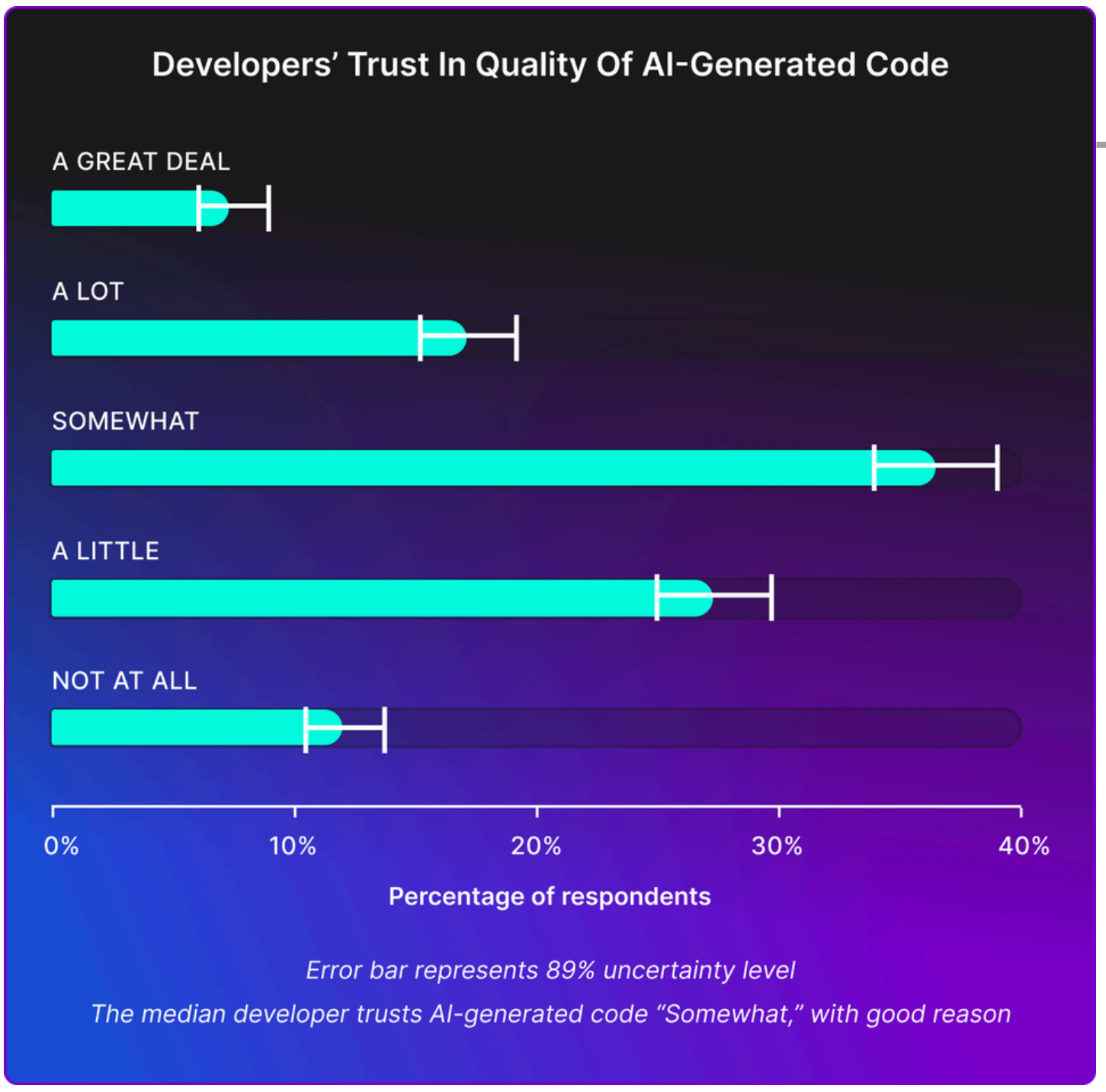
- **Visible output** (lines written, features completed) is easily observed
- **Invisible burden** (future maintenance, team confusion) is hard to quantify
- **Immediate gratification** (AI suggestions accepted) overshadows long-term costs

1: 2025 Market Research commissioned by GitKraken

Early Warning Signals in Developer Sentiment:

Monitor these sentiment indicators as predictive metrics:

1. **Autonomy Decline:** "I feel like I'm just accepting suggestions rather than designing solutions"
2. **Mastery Concerns:** "I'm not sure I could implement this without AI assistance"
3. **Purpose Questions:** "I spend more time fixing issues than building features"



Leadership Insight:
When developers express these concerns, you're 3-6 months away from seeing the impact in your delivery metrics.

PART 3

The Breakthrough - Finally Measuring AI's True Impact

Beyond Guesswork: Direct AI Measurement Is Here

Until recently, engineering leaders could only infer AI's impact through indirect metrics. Now, using the latest APIs and integrations, we can directly measure AI usage and correlate it with engineering outcomes. This changes everything.

What We Can Now Track

- **Exact AI usage patterns:** Which developers use AI, how often, and for what types of tasks
- **Acceptance rates:** The percentage of AI suggestions accepted vs. rejected
- **Line-level impact:** Precisely how many lines are AI-generated vs. human-written
- **Team variations:** How AI usage differs across teams and correlates with their performance

The New Measurement Framework

Modern engineering intelligence platforms can now provide visibility into four critical dimensions:

1. Direct AI Usage Metrics

Metric	What It Reveals	Key Insight
AI Suggestion Count	Volume of AI assistance	Engagement level
Tab/Line Acceptance Rate	Quality of AI suggestions	Developer trust
Prompted vs. Unprompted	Active vs. passive usage	Intentionality
Lines Added/Deleted by AI	Code volume impact	True AI contribution
Active vs. Inactive Users	Team adoption patterns	Cultural acceptance

2. Code Health Indicators

Metric	AI Correlation	Target Range
Refactoring Ratio	Decreases with high AI usage	0.75:1 to 1.5:1
Duplicate Block Frequency	Increases with AI acceptance	<5% of commits
Code Churn Rate	Correlates with AI volume	<7% within 2 weeks
Legacy Code Interaction	Inversely related to AI usage	>25% of changes
Refactoring Ratio	Decreases with high AI usage	0.75:1 to 1.5:1

3. Developer Experience Signals

Signal	Measurement Method	Healthy Range
Cognitive Load	Time understanding vs. writing	40-60% understanding
AI Dependency	Confidence without AI	>60% confident
Review Effectiveness	PR review quality	>70% thorough
Knowledge Transfer	Learning from reviews	Positive trend
Signal	Measurement Method	Healthy Range

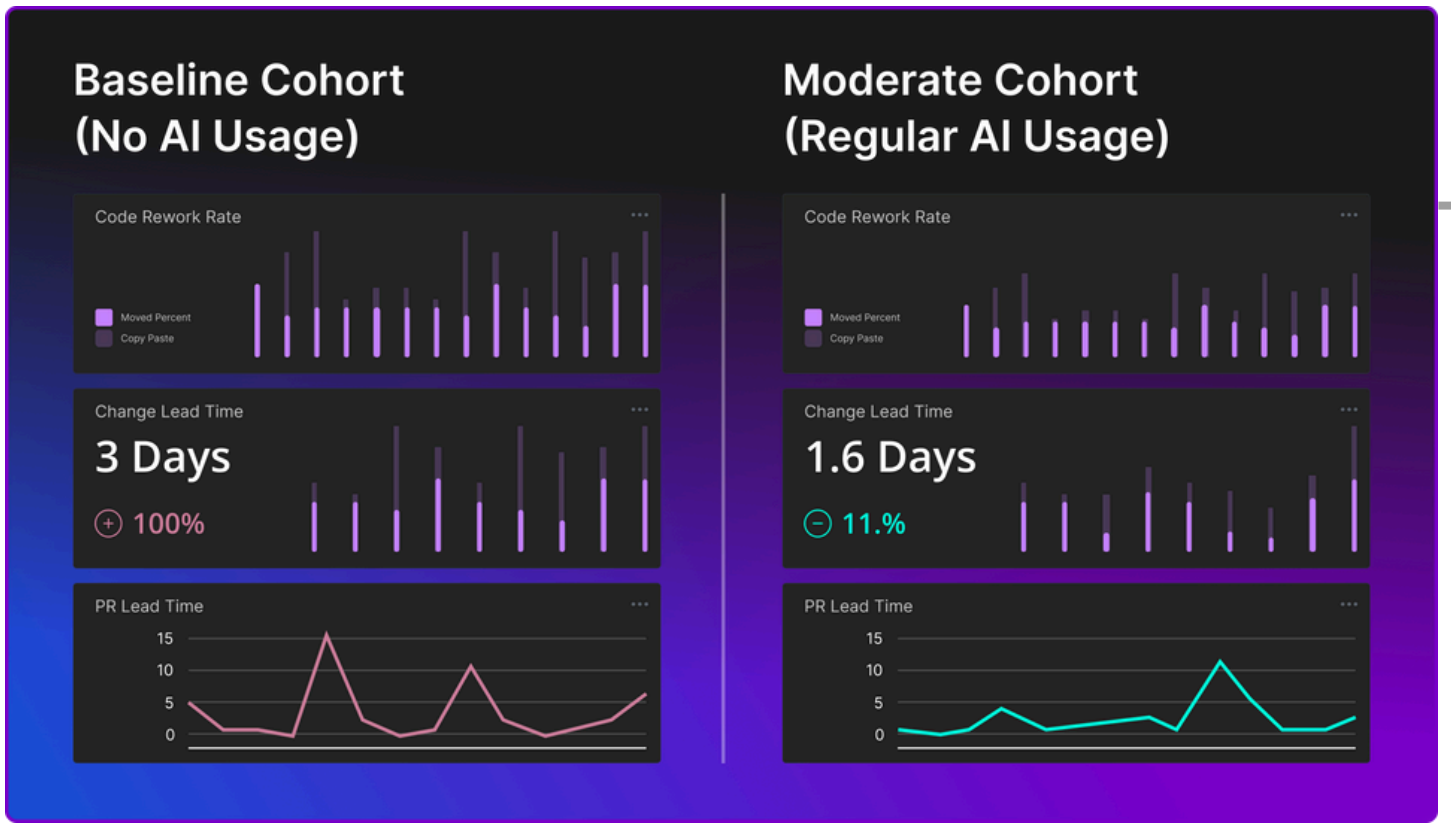
4. Business Impact Metrics

Outcome	Expected Correlation	Monitoring Frequency
Velocity	Initial increase, then plateau	Weekly
Defect Rate	Increases with duplication	Bi-weekly
Cycle Time	U-shaped curve	Monthly
Team Stability	Correlates with experience metrics	Quarterly

The Power of Correlation Analysis

With direct AI measurement, we can finally answer critical questions:

- **What's the optimal AI usage level for your team?** Not all teams benefit equally from the same level of AI assistance
- **Which metrics are most impacted by AI?** Velocity might increase while quality decreases—understanding the trade-offs is crucial
- **How do different AI usage patterns affect outcomes?** Passive acceptance of suggestions vs. active prompting have different impacts



Finding your optimal AI usage level can cut lead time in half while dramatically reducing rework—but more isn't always better.

Cohort Analysis: Understanding Your AI Usage Spectrum

By categorizing developers into cohorts based on their AI usage patterns, teams can identify optimal usage levels:

AI Usage Cohorts (Illustrative Example)

- **Inactive Users:** No AI usage detected
- **Light Users:** Occasional AI assistance (e.g., <25% of code)
- **Moderate Users:** Regular AI usage (e.g., 25-50% of code)
- **Heavy Users:** Significant AI reliance (e.g., 50-75% of code)
- **AI-Dependent:** Majority AI-generated (e.g., >75% of code)

Note: These percentages are illustrative, your team's actual cohort boundaries will depend on your codebase, team composition, and measurement methods. A team doing primarily greenfield development might see different patterns than one maintaining legacy systems.

What matters most is the correlation between usage patterns and outcomes. A senior developer using AI efficiently for 60% of their code might produce better results than a junior using it for 30%. High AI usage doesn't automatically indicate poor quality, it depends on how it's being used and by whom.

The key is identifying which patterns correlate with your desired outcomes:

- **Quality metrics:** defect rates, code duplication, maintainability
- **Velocity metrics:** cycle time, throughput, feature delivery
- **Developer satisfaction and growth:** learning, autonomy, engagement

Comparing metrics across these cohorts reveals patterns unique to your team, helping you identify where AI accelerates development without compromising quality, and where it might be hindering either productivity or learning. This data-driven approach moves beyond assumptions to show what actually works for your specific context.

PART 4

Your Action Plan

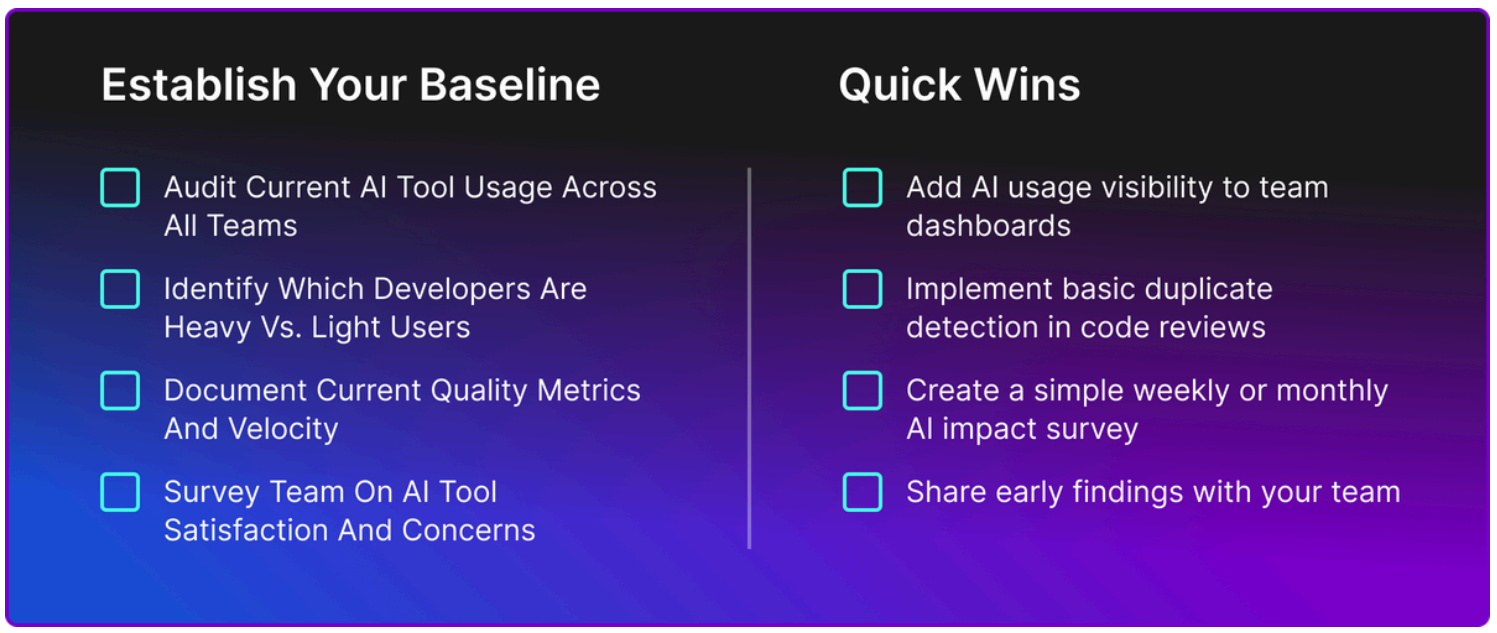
Immediate Steps (Week 1-2)

Establish Your Baseline

- Audit current AI tool usage across all teams
- Identify which developers are heavy vs. light users
- Document current quality metrics and velocity
- Survey team on AI tool satisfaction and concerns

Quick Wins

- Add AI usage visibility to team dashboards
- Implement basic duplicate detection in code reviews
- Create a simple weekly or monthly AI impact survey
- Share early findings with your team



Short-Term Initiatives (Month 1-3)

Build Your Measurement Infrastructure

Deploy comprehensive measurement that captures:

- Direct AI usage metrics from your development tools
- Code quality indicators from your repositories
- Developer experience signals from regular surveys
- Business outcomes from your project management systems

Acceleration Tip: Platforms like GitKraken Insights and GitClear can correlate AI usage from tools like GitHub Copilot and Cursor with PR metrics and team sentiment, eliminating weeks of manual setup.

Short-Term Initiatives Cont'd

Optimize Your AI Usage

Based on your cohort analysis:

1. Identify your optimal AI usage level using the frameworks discussed in this guide
2. Create guidelines for different task types
3. Train teams on effective AI collaboration
4. Monitor and adjust based on outcomes

Cultural Initiatives

- Launch "Refactoring Fridays" to combat duplication
- Recognize code consolidation achievements
- Create AI usage best practices documentation
- Establish peer review for AI-heavy features
- Host "AI Olympics" or hackathons for guided team learning and friendly competition
- Dedicate time and resources for AI experimentation and tool exploration

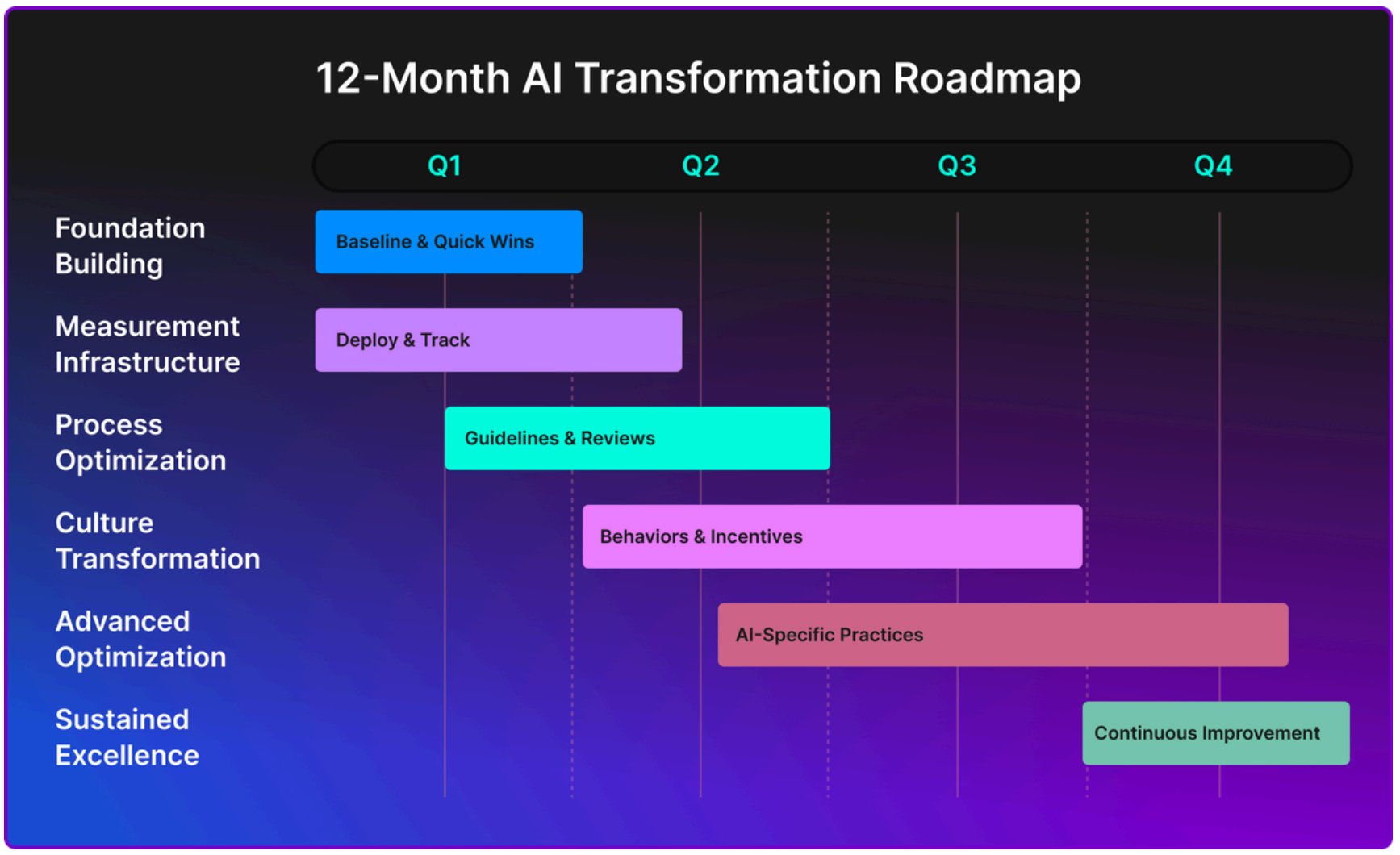
Long-Term Strategy (Quarter 2+)

Sustainable Practices

- 20% Technical Excellence Budget: Dedicate sprint capacity to code health
- AI Usage Guidelines: Clear policies on when and how to use AI
- Continuous Learning: Regular training on both AI tools and fundamental skills
- Architecture Reviews: Evaluate AI's impact on system design

Advanced Optimization

- Develop custom AI prompts aligned with your patterns
- Create team-specific AI training based on your codebase
- Implement automated refactoring suggestions
- Build feedback loops between metrics and practices



PART 5

Leading Engineering Excellence in the AI Era

The Strategic Perspective

As an engineering leader, you're not just managing current productivity—you're building sustainable competitive advantage. Teams that successfully integrate AI while maintaining engineering excellence will outperform those that optimize for only one dimension.

Finding Your Optimal AI Strategy

Every team's optimal AI usage level is different. The key is finding yours through data.

Potential Patterns to Investigate: Based on early observations and engineering principles, you might discover patterns like:

- Higher AI usage for repetitive tasks (boilerplate, tests, documentation)
- Lower AI usage for critical components (security, core business logic)
- Varying acceptance rates by code type (higher for standard patterns, lower for unique solutions)

The important point: These are hypotheses to test, not proven benchmarks. Your data will reveal what actually works for your team. Some teams might thrive with high AI usage across the board; others might find it most valuable for specific use cases.

What to measure:

- AI usage rates by code type and developer experience
- Correlation between usage patterns and quality outcomes
- Team satisfaction and velocity at different usage levels
- Long-term maintainability of AI-generated vs. human-written code

Building Your AI-Augmented Culture

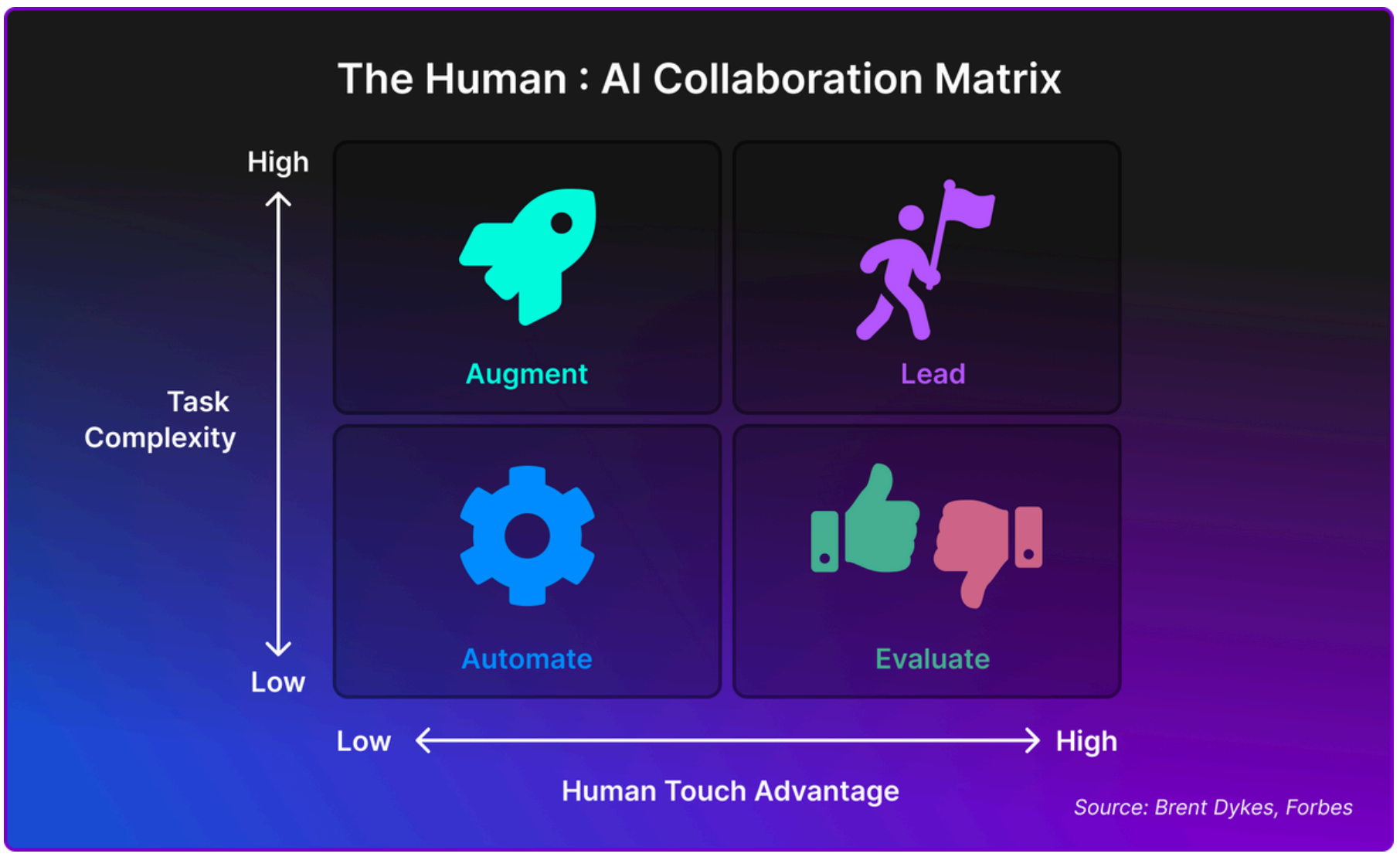
Three Cultural Pillars:

1. **Measurement as Empowerment**
   - Teams own their metrics
   - Data drives discussions, not blame
   - Continuous improvement over perfection
2. **Balance as a Core Value**
   - Speed and quality in harmony
   - AI assistance and human judgment
   - Innovation and sustainability
3. **Continuous Adaptation**
   - Regular evaluation of AI impact
   - Evolving practices based on data
   - Investment in both AI and human capabilities

The Human Advantage

As AI handles routine coding, human developers must focus on uniquely human capabilities:

- **System Design:** Understanding how pieces fit together
- **Simplification:** Recognizing opportunities to consolidate
- **Context Application:** Knowing why, not just how
- **Quality Judgment:** Determining "good enough" vs. "needs improvement"
- **Knowledge Transfer:** Teaching and mentoring others



Understanding Brent Dykes' *Human : AI Collaboration Matrix* can help you and your team maximize AI through ideal use cases, avoiding the pitfalls of improper AI utilization.

Conclusion:

The Path Forward

AI coding assistants are not going away, nor should they. They offer genuine value in accelerating development and reducing boilerplate work. However, without careful measurement and management, they can undermine the very productivity gains they promise to deliver.

Key Takeaways for Engineering Leaders

1. **Direct measurement changes everything:** You can now see exactly how AI impacts your team, not just guess
2. **Optimal usage varies by team:** Use cohort analysis to understand your patterns, but don't slow AI adoption. Instead, focus on better safeguards: automated testing, thorough CI/CD pipelines, deeper reviews of AI-generated code, and smaller incremental changes. The goal isn't less AI, it's better AI integration.
3. **Balance is achievable:** With the right metrics and practices, you can have both speed and quality
4. **Early action matters:** Teams that establish measurement now will outperform those that wait
5. **Culture drives success:** Technology enables, but culture determines outcomes

Your Competitive Advantage

While your competitors guess at AI's impact, you'll have data. While they chase vanity metrics, you'll optimize for sustainable velocity. While they accumulate technical debt, you'll build maintainable systems.

Your Next Step

Start with measurement. The ability to directly track AI usage and correlate it with engineering outcomes is a game-changer. In one week, you can have visibility. In one month, you can have insights. In one quarter, you can have optimization. In one year, you'll have sustainable competitive advantage.

LEVEL UP

Tools for the Journey: Implementing Your Measurement Strategy

The Challenge of Manual Measurement

Many teams attempt to track these metrics using spreadsheets and custom scripts. However, manual approaches fail because:

- Data collection becomes unsustainable
- Correlation between metrics requires significant effort
- Real-time visibility is impossible
- Developer survey fatigue leads to declining responses

The Breakthrough: GitKraken Insights and GitClear

GitKraken Insights and GitClear are tools specifically designed to solve the AI measurement challenge outlined in this guide. For the first time, engineering leaders can:

Direct AI Measurement

- Automatic tracking from GitHub Copilot, Cursor, and other AI tools
- Count of lines added and deleted by AI
- Acceptance rate analysis by developer and team
- Real-time usage patterns and trends

Intelligent Correlation

- Automatic correlation between AI usage and velocity metrics
- Quality impact analysis including defect rates and code duplication
- Developer experience correlation with productivity outcomes
- Custom correlation analysis for your specific concerns

Cohort Analysis & Optimization

- Automatic categorization of developers by AI usage level
- Performance comparison across cohorts
- Identification of optimal usage patterns for your team
- Recommendations based on successful patterns

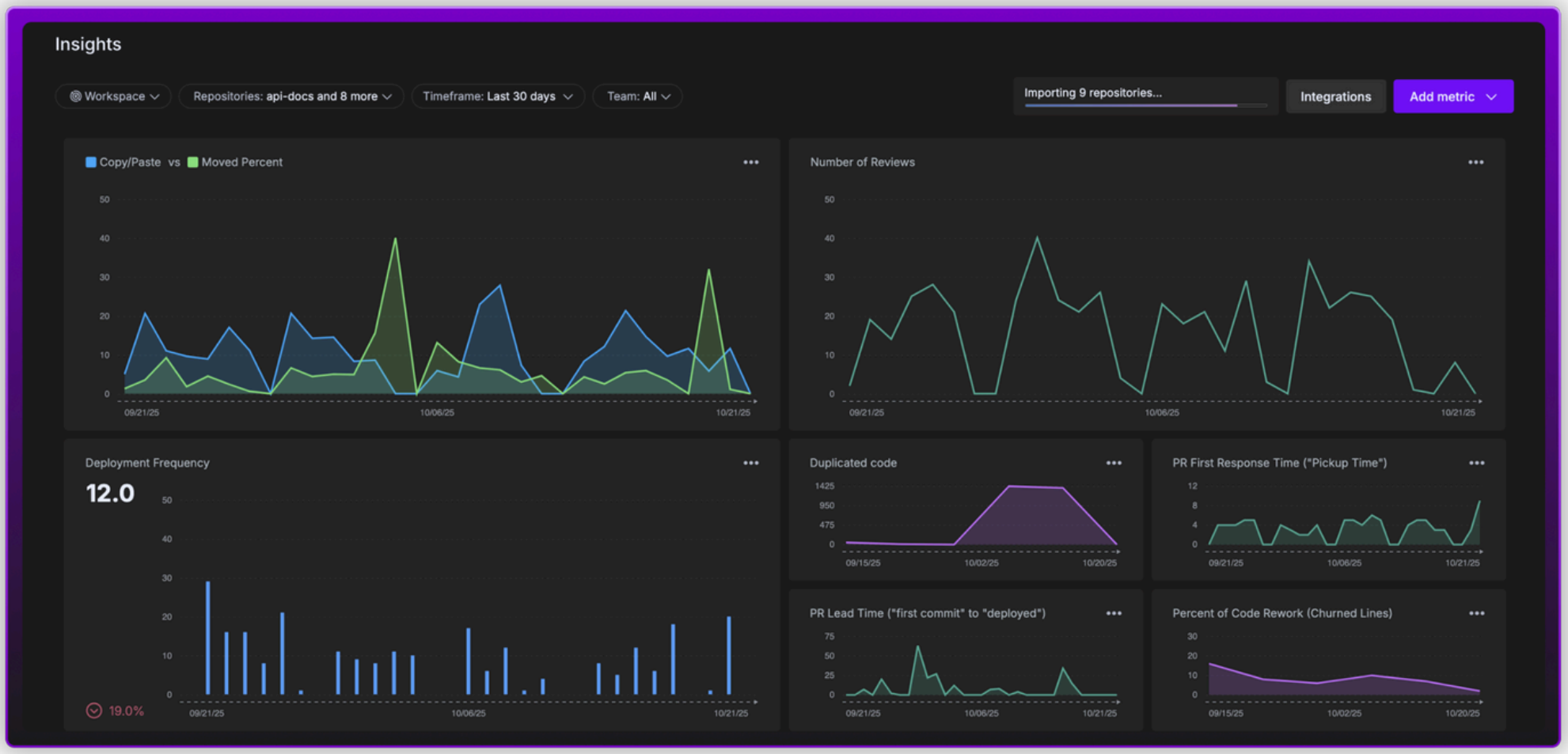
For more resources on code quality measurement and developer experience optimization, visit **GitKraken.com/insights** and **GitClear.com**

This report is based on **GitClear's analysis of 211 million lines of code** from 2020-2024 and insights from thousands of development teams worldwide.

Ready to see what productivity killers are affecting your team?

GitKraken Insights helps engineering leaders identify and address the hidden factors impacting team productivity, from context switching patterns to AI tool effectiveness to developer satisfaction trends. Built by developers for developers, it provides the comprehensive intelligence you need to optimize your team's performance.

Learn more about how GitKraken Insights can help you build a more productive engineering organization at **gitkraken.com/insights**



GitKraken is trusted by over **40 million developers** worldwide for critical development workflows. Our **new Insights platform** brings the same developer-trusted approach to engineering productivity measurement and optimization.

GitKraken