DORA

AI Capabilities Model

2025

Google Cloud

Contents

Executive summary

Key takeaway: AI is an amplifier

The DORA State of AI-assisted Software Development report[1] established who is adopting AI (nearly 90% of professionals) and how they are using it.

Based on over 100 hours of qualitative data and survey responses from nearly 5,000 technology professionals, our research reveals a critical truth: AI's primary role in software development is to amplify. It magnifies the strengths of high-performing organizations and the dysfunctions of struggling ones.

The greatest returns come not from the tools themselves, but from investing in the foundational systems that enable success.

DORA AI Capabilities Model

The DORA State of AI-assisted Software Development report introduced the DORA AI Capabilities Model, which identifies the seven foundational capabilities—spanning both technical and cultural domains—that are proven to amplify the positive impact of AI on performance.

This new model is a companion to the long-standing DORA Core Model.[2] While DORA's Core Model identifies the foundational capabilities that drive software delivery and organizational performance, the DORA AI Capabilities Model focuses specifically on the elements that organizations must cultivate to maximize the value and success of AI adoption.

The seven key capabilities are:

Clear and communicated AI stance

Ambiguity creates risk. A clear policy provides the psychological safety needed for effective experimentation.

Healthy data ecosystems

The benefits of AI are significantly amplified by high-quality, accessible, and unified internal data.

AI-accessible internal data

Connecting AI to your internal documentation and codebases moves it from a generic assistant to a specialized expert.

Strong version control practices

As AI increases the velocity of change, version control becomes the critical safety net that enables confident experimentation.

Working in small batches

This discipline counteracts the risk of AI generating large, unstable changes, ensuring that speed translates to better product performance.

User-centric focus

A focus on user needs is essential to ensure that AI-accelerated teams are moving quickly in the right direction.

Quality internal platforms

A platform provides the automated, secure pathways that allow AI's benefits to scale across the organization.

Using this report

This report serves as a practical guide to the DORA AI Capabilities Model. It moves beyond the "what" and "why" to provide actionable guidance on the "how." For each of the seven core capabilities, this report details implementation strategies, specific tactics for teams to get started, and methods for monitoring progress and fostering continuous improvement. The goal is to equip technology leaders and practitioners with the knowledge to build an organizational environment where the transformative potential of AI in software development can be fully realized.

We recommend using this report as a companion to the 2025 State of AI-assisted Software Development report.[3]

While every organization is unique, our findings provide a framework to inform your strategy and guide your teams. Use this research to form hypotheses, run experiments, and measure the results to discover what drives the highest performance in your specific context.

1. State of AI-assisted Software Development. https://dora.dev/dora-report-2025
2. "DORA's Research Program." https://dora.dev/research
3. State of AI-assisted Software Development. https://dora.dev/dora-report-2025

AI is an amplifier

Nathen Harvey
DORA Lead, Google Cloud

A central question for technology leaders is how to best realize the value of AI. To help them answer this, DORA conducted research in 2025 that included more than 100 hours of qualitative data and survey responses from nearly 5,000 technology professionals from around the world.[1]

The research reveals a critical truth: AI's primary role in software development is to amplify. AI magnifies the strengths of high-performing organizations—and the dysfunctions of struggling ones.

More than tools, AI success depends on culture and capabilities

The value of AI is unlocked by the surrounding technical and cultural environment. We've identified seven foundational capabilities—including a clear AI policy, a healthy data ecosystem, a quality internal platform, and a user-centric focus—that are proven to amplify the positive impact of AI on performance.

Treat your AI adoption as an organizational transformation. The greatest returns will come from investing in the foundational systems that amplify AI's benefits: your internal platform, your data ecosystem, and the core engineering disciplines of your teams.

These elements are the essential prerequisites for turning AI's potential into measurable organizational performance.

Indeed, many of these are the same core capabilities[2] that have long been proven to enable high-performing, technology-driven teams. AI is the latest significant change to this landscape, bringing both exciting possibilities and new questions.

DORA has long strived to help organizations use data-backed decisions to navigate precisely these kinds of evolutions and meet the moment. This year, we went beyond questions of who is adopting AI (90% of our survey respondents use AI as part of their work)[3] and how they're using it, to investigate the conditions in which AI-assisted software developers observe the best outcomes.

We present these findings as our first DORA AI Capabilities Model. The seven AI capabilities in this inaugural model are shown to amplify the benefits of AI adoption. Our research encompasses both the technical and cultural aspects of an organization, and suggests that investing in developing these areas can help unlock the potential of AI tools.

The DORA AI Capabilities Model is designed to be complementary to the DORA Core Model.[4] It does not replace it. While the Core Model identifies the foundational drivers of software delivery performance, this new model investigates the specific conditions under which AI assistance drives better outcomes for technology-driven teams and organizations.

As with the DORA Core Model,[5] we will continue validating, revising, and refining the DORA AI Capabilities Model with further research.

DORA AI Capabilities Model

Figure 1 visualizes which capabilities amplify the effect of AI adoption on specific outcomes. This is the DORA AI Capabilities Model.

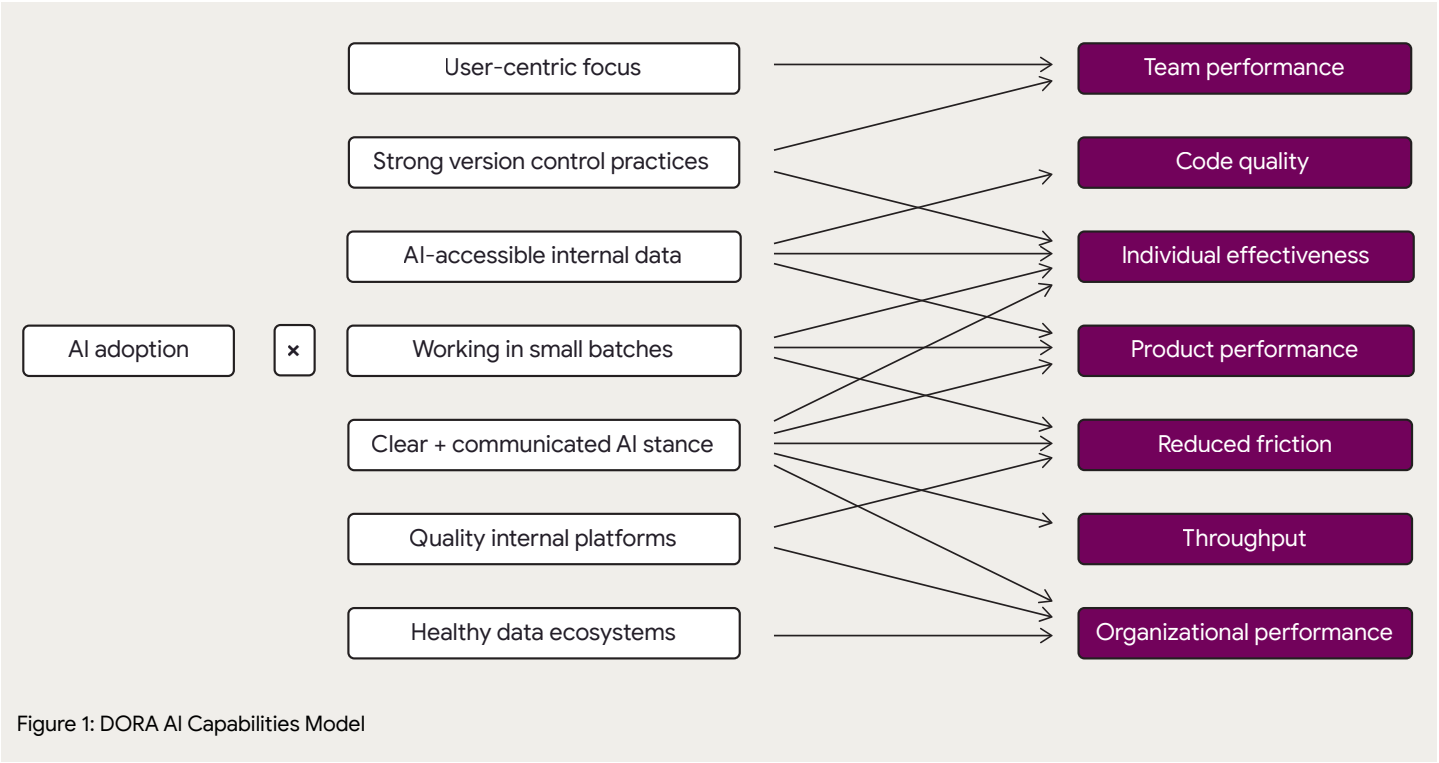


Figure 1: DORA AI Capabilities Model

Successfully leveraging AI in software development is not as simple as adopting new tools. Rather, organizations must cultivate a specific technical and cultural environment to reap the greatest rewards.

AI's relationship to key outcomes

Our research seeks to determine whether different levels of AI adoption predict differences in key outcomes:

Organizational performance

The overall success of the organization, based on characteristics like profitability, market share, and customer satisfaction.

Team performance

The perceived effectiveness and collaborative strength of an individual's immediate team.

Product performance

The success and quality of the products or services the team is building, based on characteristics like helping users accomplish important tasks and keeping information safe, and performance metrics such as latency.

Software delivery throughput

The speed and efficiency of the software delivery process.

Software delivery instability

The quality and reliability of the software delivery process.

Code quality

An individual's assessment of the quality of code underlying the primary application or service they work on.

Individual effectiveness

An individual's self-assessed effectiveness and sense of accomplishment at work.

Valuable work

The self-assessed amount of time an individual spends doing work they feel is valuable and worthwhile.

Friction

The extent to which friction hinders an individual's work. Lower amounts of friction are generally considered to be a positive outcome.

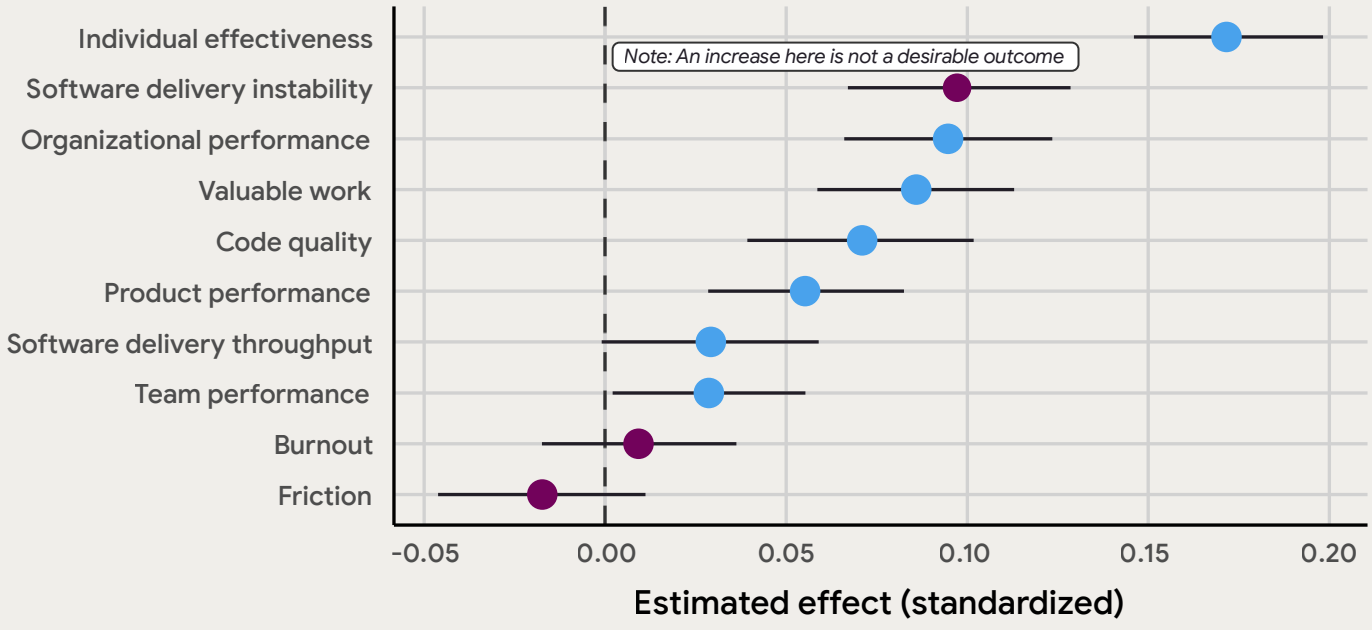
Burnout

Feelings of exhaustion and cynicism related to one's work. Lower amounts of burnout are generally considered to be a positive outcome.

Figure 2 visualizes the relationships AI adoption has with these outcomes.

Read more about how we measure AI adoption and how it relates to the key outcomes in the "Exploring AI's relationship to key outcomes" chapter of the State of AI-assisted Software Development report.[6]

The landscape of AI's impact
Estimated effect of AI adoption on key outcomes, with 89% credible intervals



For outcomes in purple, such as burnout, a negative effect is desirable.
Figure 2: The landscape of AI's impact

1. Additional detail about who participated in this year's research is available in the "Demographics and firmographics" chapter of the State of AI-assisted Software Development report. https://dora.dev/dora-report-2025
2. "DORA's Research Program." https://dora.dev/research/?view=detail
3. State of AI-assisted Software Development. 24. https://dora.dev/dora-report-2025
4. "DORA's Research Program." https://dora.dev/research
5. Ibid.
6. State of AI-assisted Software Development. 33–48. https://dora.dev/dora-report-2025

DORA COMMUNITY

The DORA Community[1] provides a platform for professionals to engage with this research and apply it to improve their own organizational performance.

Why join the DORA Community?

There are several reasons why you should be a part of the DORA Community:

Learn from experts and peers: The community offers opportunities to learn from guest speakers and other members through presentations and discussions.

Stay up to date with research: Be the first to know about new information and publications from DORA.

Collaborate and discuss: The DORA Community Google Group[2] provides a forum for asynchronous conversations, announcements, and event invitations. This allows members to discuss topics and share their experiences with others in the field.

Engage in community events: A calendar of events, both virtual and in-person, is available on DORA.community.

Contribute to the conversation: Contribute to the conversation by listening, talking, and participating in chats. The community values the input of its members and provides a space for ongoing discussions on topics like leadership, team empowerment, and the evolution of technology practices.

1. The DORA Community. https://dora.community
2. The DORA Community Google Group. https://groups.google.com/g/dora-community/about

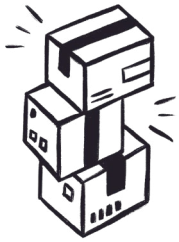
The seven AI capabilities

Not everyone experiences the same outcomes from adopting AI. We wanted to understand the particular capabilities and conditions that enable teams to achieve positive outcomes.

Nathen Harvey
DORA Lead, Google Cloud

These seven capabilities form the core of our new model:

Clear and communicated AI stance

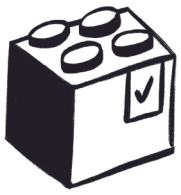


Working in small batches

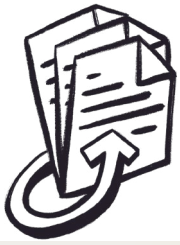
Healthy data ecosystems

User-centric focus

AI-accaccessible internal data



Quality internal platforms



Strong version control practices

This report provides practical advice based on the seven DORA AI capabilities. Here is a quick summary to help you get started.

Clarify and socialize your AI policies

Ambiguity around AI stifles adoption and creates risk. Establish and socialize a clear policy on permitted tools and usage to build developer trust. This clarity provides the psychological safety needed for effective experimentation, reducing friction and amplifying AI's positive impact on individual effectiveness and organizational performance.

Treat your data as a strategic asset

The benefits of AI on organizational performance are significantly amplified by a healthy data ecosystem. Invest in the quality, accessibility, and unification of your internal data sources. When your AI tools can learn from high-quality internal data, their value to your organization increases.

Connect AI to your internal context

Connect your AI tools to your internal systems to move beyond generic assistance and unlock boosts in individual effectiveness and code quality. This means going beyond simply procuring licenses, and investing the engineering effort to give your AI tools secure access to internal documentation, codebases, and other data sources. This provides the company-specific context necessary for the tools to be maximally effective.

Embrace and fortify your safety nets

AI-assisted coding can increase the volume and velocity of changes, which can also lead to more instability. Your version control system is a critical safety net. Encourage teams to become highly proficient in using rollback and revert features, as this practice is associated with better team performance in an AI-assisted environment.

Reduce the size of work items

While AI can increase perceptions of individual effectiveness by generating large amounts of code, our findings show this isn't necessarily the most important metric. Instead, focus on outcomes. Enforce the discipline of working in small batches, which improves product performance and reduces friction for AI-assisted teams.

Center users' needs in product strategy

Professionals can experience large increases in their personal effectiveness when they adopt AI. However, if the needs of people they're building for aren't their focus, they may be moving quickly in the wrong direction. We found that adopting AI-assisted development tools can harm teams that don't have a user-centric focus. Conversely, keeping the users' needs as a product's North Star can guide AI-assisted developers toward appropriate goals and has an exceptionally strong positive effect on the performance of teams using AI.

Invest in your internal platform

A quality internal platform is a key enabler for magnifying the positive effects of AI on organizational performance. These platforms provide the necessary guardrails and shared capabilities that allow AI benefits to scale effectively and securely across the organization.

Clear and communicated AI stance

Nathen Harvey

DORA Lead, Google Cloud

Vivian Hu

10X Lead, delta Google Cloud Consulting, Google Cloud

A clear and communicated stance amplifies AI adoption's positive influence on:

- Individual effectiveness
- Organizational performance
- Software delivery throughput

A clear and communicated stance on AI adoption decreases friction.

Why a clear AI stance matters for AI adoption

Ambiguity is the enemy of both innovation and safety. A clear stance provides psychological safety, empowering developers to experiment effectively and use AI tools with confidence, knowing they are operating within approved boundaries.

Clarity can also reduce friction and amplify AI's positive impact on individual effectiveness and organizational performance.

A clear and communicated AI stance is an organization's official position on developer use of AI-assisted development tools.

A successful stance is not about being "permissive" or "restrictive" but about being:

Comprehensible: It is well-defined, easily understandable, and provides practical guardrails.

Communicated: It is widely known, accessible, and socialized across the development teams.

This clarity must apply to four key areas:

1. Expectation of use: Does it feel like using AI at work is expected?
2. Support: Does the organization support developers in experimenting with AI?
3. Permission: Is it clear which AI tools are permitted for use?
4. Applicability: Does the organization's AI policy feel like it directly applies to your role?

Throughout the in-depth interviews we conducted this year, developers consistently revealed a lack of clarity and awareness of their organization's AI stance. This manifested in two ways:

1. Developers acting too conservatively, fearing they'll overstep boundaries.
2. Developers acting too permissively, using AI in unapproved ways.

Neither of these cases is optimal.

Organizations having a clear and communicated stance about the expectations and acceptability of AI in software development can help foster developers' trust,[1] assuage data privacy concerns,[2] and scale AI adoption.[3]

A clear and communicated AI stance moderates AI's impact on individual effectiveness

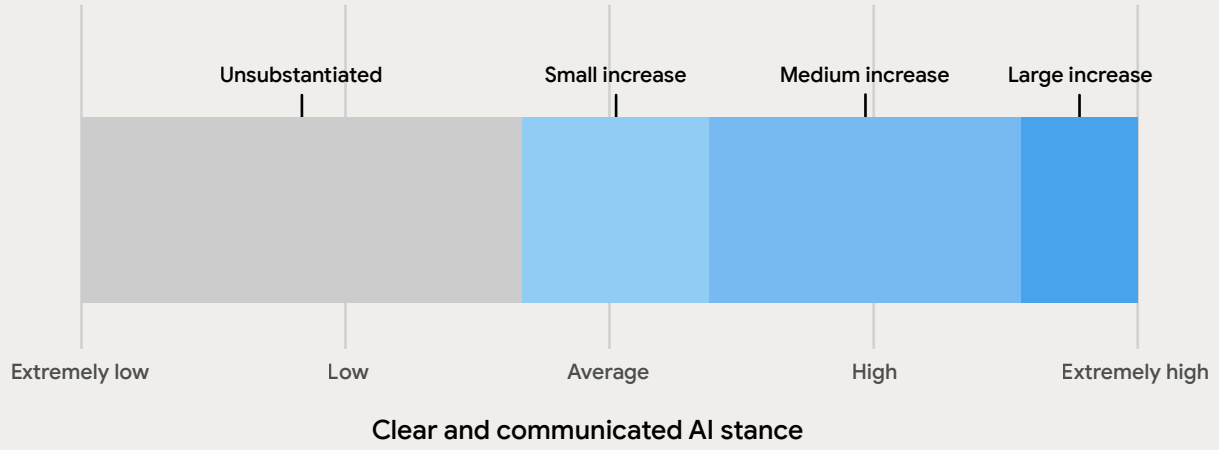


Figure 3: A clear and communicated AI stance moderates AI's impact on individual effectiveness

A clear and communicated AI stance determines AI's impact on organizational performance

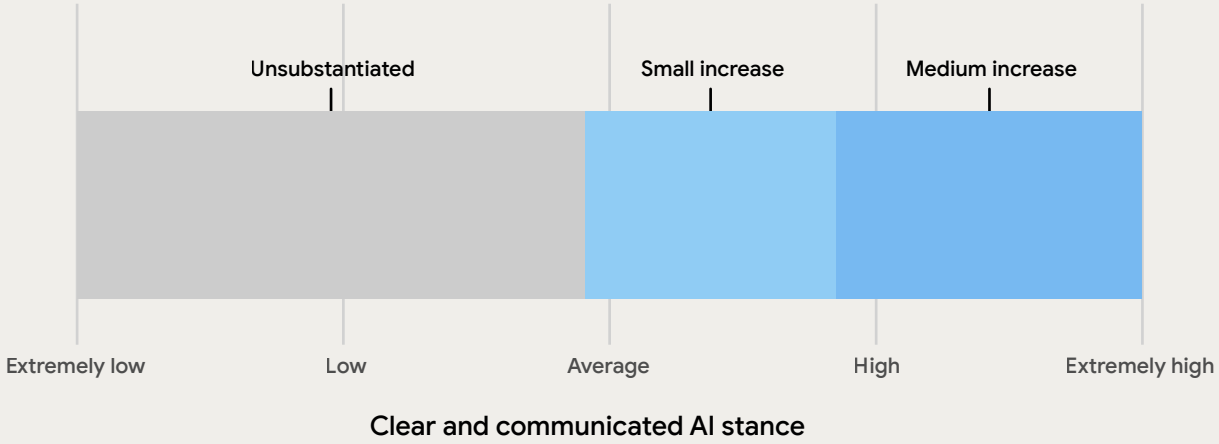


Figure 4: A clear and communicated AI stance determines AI's impact on organizational performance

A clear and communicated AI stance moderates AI's impact on throughput

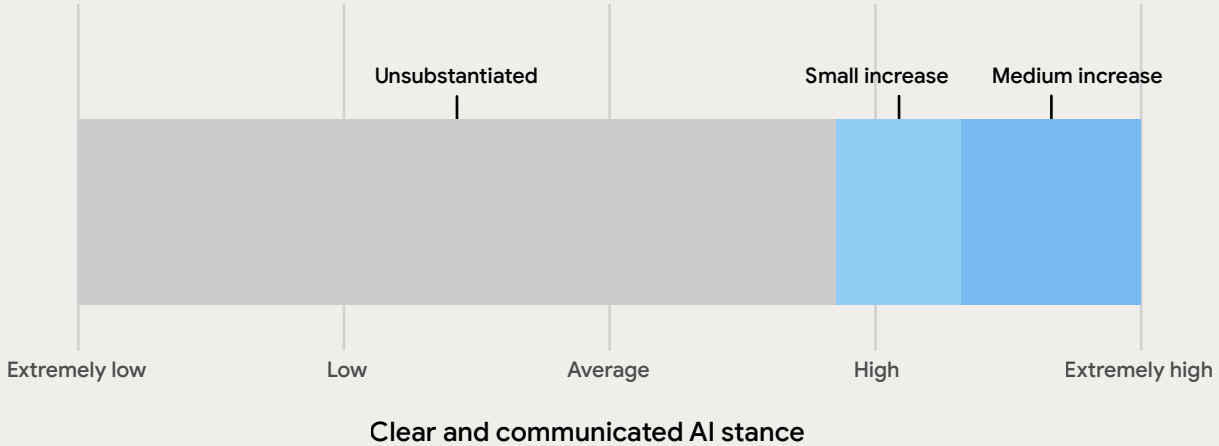


Figure 5: A clear and communicated AI stance moderates AI's impact on throughput

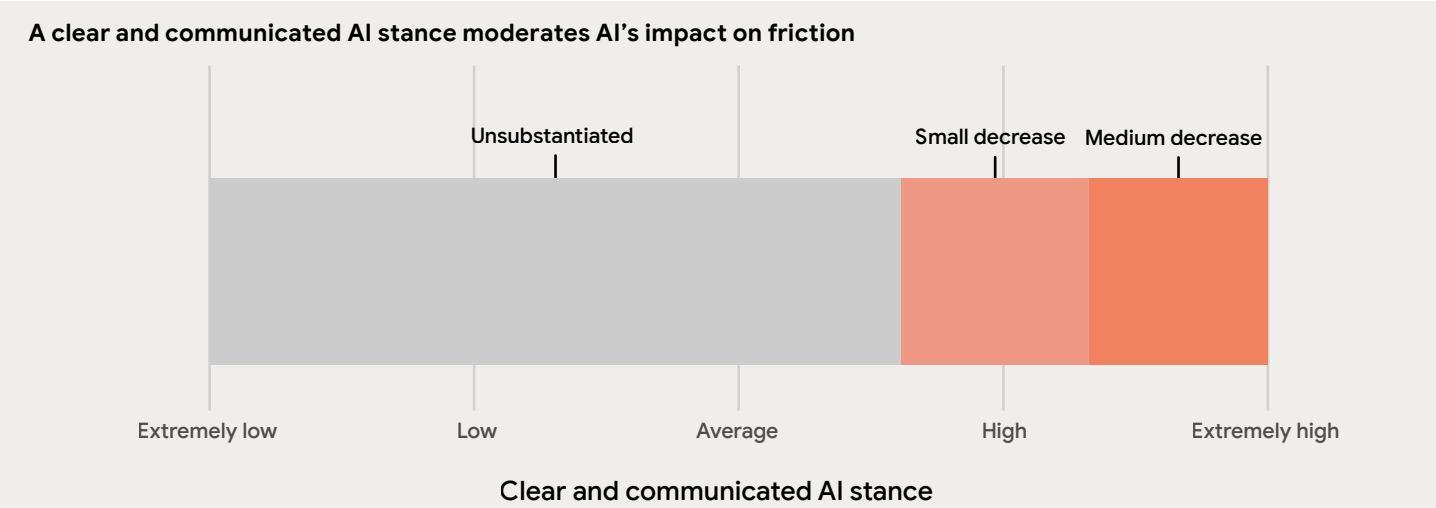


Figure 6: A clear and communicated AI stance moderates AI's impact on friction

How to improve your AI stance: From policy to daily use

Building a clear and communicated stance is a deliberate, cross-functional effort that moves from executive vision to daily use. It's a journey, not a one-time checklist.

It begins with executive sponsorship. Leadership must define and champion a clear AI mission, goals, and adoption plan. This signals the strategic importance of AI, dedicates the necessary resources to it, and provides the authority to enact a comprehensive policy.

That leadership vision, however, can't be implemented from a single silo. A policy created only by legal or security is unlikely to work in reality.

The stance should be authored by a cross-functional working group with representatives from engineering, legal, security, IT, and product leadership. A group of this kind is uniquely positioned to balance risk management with the practical realities of developer workflows.

This group should also define clear, long-term owners for the policy, who will be responsible for verifying changes and triaging suggestions from the feedback loop. To ensure the output is comprehensible, this group should adopt a simple style guide for the policy itself. This guide should emphasize the use of plain language, provide concrete examples of "do" and "don't," and avoid internal jargon and legalese as much as possible.

The primary output of this group shouldn't be a simple "yes/no" document, but a nuanced, risk-based framework. As a starting point, this guide on crafting an acceptable use policy for generative AI[4] offers practical advice on how to build a framework and balance stakeholder needs.

An effective model should serve as the documented set of critical use cases for AI, and use a "three-bucket" approach to categorize tools and data:

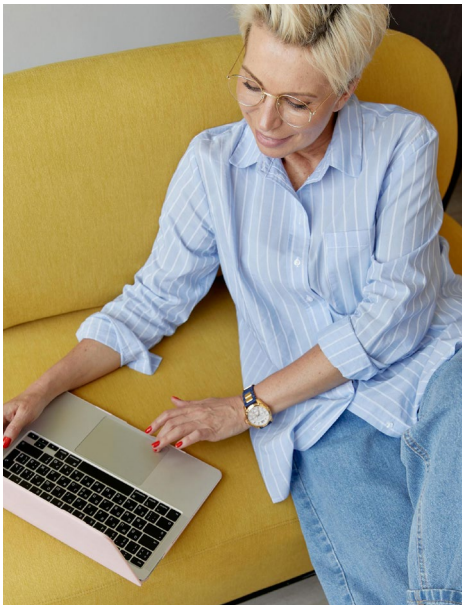
Prohibited: High-risk, unacceptable uses (for example, inputting any customer's personally identifiable information or trade secrets into a public AI model).

Permitted with guardrails: Permissible only with specific controls (for example, using proprietary source code only with an approved, enterprise-grade AI tool; requiring human-in-the-loop review for all AI-generated code).

Allowed: Low-risk, high-value activities that are actively encouraged (for example, generating boilerplate code, brainstorming ideas using no proprietary data).

This framework should be published as a "living document" in a central, easily searchable hub, such as an internal developer portal or wiki. This hub should host the policy, the official list of approved tools, and evolving frequently asked questions.

Finally, a policy is only useful if it's known. Its launch must be socialized through "town halls," team meetings, and formal training. Most importantly, this communication must establish a clear feedback loop for developers to ask questions and suggest updates, allowing the policy to evolve as the technology and business needs change.



Common obstacles to a clear and communicated AI stance

Treating the policy as a static document

The technology and workflows that enable AI assistance to have an impact on important outcomes are changing rapidly. Organizations should use lessons learned from experimenting with AI to inform regular updates to their AI stance and acceptable use policies.

Changing too frequently

Policy changes take time to be fully incorporated by teams. A policy that's always changing may be worse than having no policy at all. Ensure there is time and incentive for teams to adapt to changes in the AI stance.

A myopic view of the policy

An AI stance should balance the needs of all stakeholders, not just one group.

How to measure your AI stance

The most direct way to measure this capability is by surveying your teams to gauge developer perception and clarity.

DORA survey questions

DORA research measured this capability based on responses to these questions:[5]

- To what extent do you feel that the use of AI at work is mandatory?
- To what extent does your organization support you with experimenting with AI?
- To what extent is it clear how you're allowed and not allowed to use AI at work?
- To what extent does your organization's AI policy directly apply to your work?
- To what extent is it clear which AI tools you're allowed and not allowed to use at work?

Additional measures

You might also measure policy awareness by tracking page views for the policy and the number of clarifying questions about the AI policy in public channels.

Conclusion

A clear and communicated AI stance is the essential foundation for successful AI integration. Its effectiveness hinges on clarity, which reduces friction and gives developers the confidence to innovate safely. This capability works in concert with the other six capabilities in the DORA AI Capabilities Model to achieve the full, positive impact of AI.

The next capability focuses on creating a high-quality, reliable source of information for AI to consume, which requires building healthy data ecosystems.

1. "Fostering developers' trust in generative artificial intelligence." https://dora.dev/research/ai/trust-in-ai
2. "Concerns beyond the accuracy of AI output." https://dora.dev/research/ai/concerns-beyond-accuracy-of-ai-output
3. "Helping developers adopt generative AI: Four practical strategies for organizations." https://dora.dev/research/ai/adopt-gen-ai
4. "How to craft an Acceptable Use Policy for gen AI (and look smart doing it)." https://cloud.google.com/transform/how-to-craft-an-acceptable-use-policy-for-gen-ai-and-look-smart-doing-it
5. "Survey questions: AI policy." https://dora.dev/ai/capabilities-model/questions/#clear-and-communicated-ai-stance

Healthy data ecosystems

Ameer Abbas

Product Manager, Google Cloud

Lucia Subatin

Developer Advocate, Google Cloud

A healthy data ecosystem amplifies AI adoption's positive influence on organizational performance.

Why healthy data ecosystems matter for AI

When organizations invest in creating and maintaining high-quality, accessible, unified data ecosystems, they can yield even higher benefits for their organizational performance than with AI adoption alone.

A healthy data ecosystem refers to the overall quality and utility of an organization's internal data systems. Our research into AI-assisted development identifies healthy ecosystems as a foundational capability, defined by the extent to which internal data is high-quality, easily accessible, and unified.

In an organization with a healthy data ecosystem:

- Internal data is robust and trustworthy, ensuring its suitability for use as context for AI tools.
- Data is highly accessible to all necessary teams and systems, including developers and AI tools.
- Data is not siloed or divided, promoting a holistic, unified view across the organization.

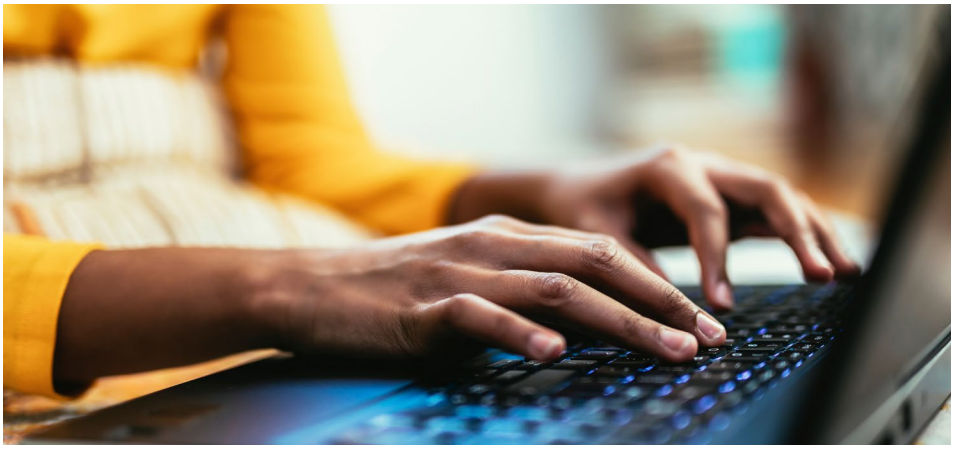
Systems are in place to ensure the integrity and continuous availability of data, making it a reliable resource for both human decision-makers and advanced AI-assisted systems.

The value of AI adoption is conditional; it's unlocked not just by the tools themselves, but by the surrounding technical and cultural environment. Our research confirms that AI functions as an amplifier,[1] magnifying the strengths—or dysfunctions—of existing organizational systems.

A healthy data ecosystem is a crucial prerequisite that determines the overall success of AI adoption, particularly in maximizing returns on investment.

Amplified organizational performance

When organizations invest in creating and maintaining high-quality, accessible, and unified data ecosystems, the positive influence of AI adoption on organizational performance is significantly amplified.



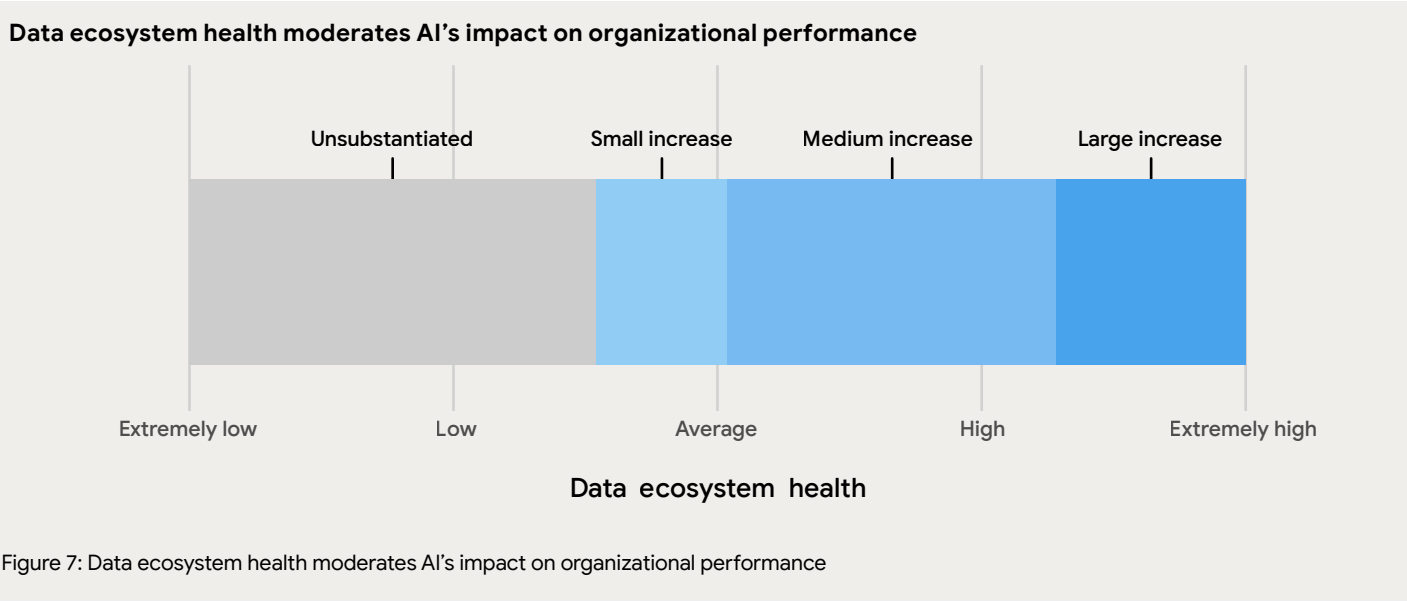


Figure 7: Data ecosystem health moderates AI's impact on organizational performance

Enabling contextually relevant assistance

A healthy data ecosystem is essential for developing sophisticated, context-aware AI assistance. It's necessary for tasks such as fine-tuning foundational models on proprietary knowledge, or implementing effective retrieval-augmented generation (RAG) systems that require high-quality data retrieval.

Reducing downstream chaos

When data environments are low quality, AI risks creating localized productivity gains that are "absorbed by downstream bottlenecks," resulting in little real value for the organization as a whole. Investing in data health ensures individual productivity gains translate into systemic organizational advantages.

Common obstacles to a healthy data ecosystem

Data as a by-product

Cultural shifts are required to create healthy data ecosystems. Organizational leaders should sponsor transitioning from treating data as a by-product of transactions to seeing it as a product with consumers that needs a cohesive, long-term strategy.

The tool is the silo

A mature data platform will be able to manage different sources and their specific approaches to data storage and modeling, including different ingestion tools and schemas. While some tools are more opinionated than others, part of breaking the data silos is breaking with the opinionated approach from specific tools when necessary to meet a common ground.

Prioritizing the right tool for the job and the platform, as well as having flexible integrations and properly documented schemas, becomes especially relevant when internal data can be complemented by external sources as preparation for RAG.

How to improve your data ecosystem's health

Improving data ecosystems requires a strategic organizational investment supported by continuous improvement principles, akin to improving core DORA technical capabilities like continuous integration and continuous delivery.

Obtain sponsorship from leaders
Building and continuously improving foundational capabilities requires engaged leaders who possess the authority and budget to make large-scale changes and provide visible support for the initiative.[2] Leaders play a pivotal role in promoting the necessary cultural and technical shifts.

Invest in data governance
Organizations should establish clear owners and stewards for critical data domains. For example, leaders should appoint a specific team to own "customer account data" and be responsible for its accuracy, metadata, and access policies.

Clarity around ownership and processes is fundamental to maintaining documentation quality[3] and, by extension, data integrity. Clear governance ensures accountability and defines responsibilities for data maintenance, updates, and verification.

Effective governance enables data access. The goal is to create a "paved road" of secure, well-documented, and trusted data pathways, balancing security controls with the need to make data discoverable.

Prioritize a "single source of truth"
Many organizations bury information in disparate systems and informal knowledge channels. Teams can address the issue of siloed and divided data by identifying key data sources, and working to consolidate or federate them to create a unified view accessible to business users.

Creating a unified view may require building workflows that automatically tag and surface the data's context during development and review, reducing the time developers spend searching and improving the quality of AI output. These workflows reduce friction for developers and AI systems attempting to access necessary context, and enable business users to provide feedback on the same data to avoid discrepancies downstream.

Implement data quality frameworks
To align with the principle of building quality in,[4] we recommend introducing automated checks and monitoring to ensure data accuracy, completeness, and timeliness. Automated validation practices mirror the necessity of automatic, continuous testing throughout the software development lifecycle, ensuring data is reliable and fit for purpose.

Democratize data access
Use tools and platforms that make it easy for teams to discover and access the data they need, coupled with security controls. Increasing accessibility directly supports the creation of AI-accessible internal data, allowing AI tools to move beyond generic assistance by providing the company-specific context necessary to improve code quality and individual effectiveness.

Document and share data locally
Teams can begin by documenting the key datasets they produce and consume. For example, a team may agree to add and maintain a "data" section to an application's README.md that includes information about what data is created or used by this application.

Ideally, this documentation and metadata (for example, table and column descriptions) are treated as a code artifact. They are versioned, live alongside the data, and are equally available to all types of consuming applications and across environments.

How to measure your data ecosystem's health

Progress in establishing healthy data ecosystems should be assessed through a combination of perceptual measures and objective metrics, guiding the organization toward continuous improvement.

DORA survey questions

DORA research measured this capability by asking these questions:[5]

- How easily can you access and analyze the internal data sources you need to complete your work?
- How often are you unable to use important internal data because the data is siloed?
- How would you rate the overall quality of the data you typically rely on for your work?
- If you need a particular data point, how likely is it that you can get a definitive answer within 1 hour of searching?

Additional measures

Additional signals that may help measure the health of a data ecosystem include:

Factor to test	What to measure
Timeliness of data	Elapsed time it takes for a developer to acquire access to a necessary dataset for a new project This metric serves as a key indicator of flow, similar to measuring lead time for changes, helping identify and remove constraints in data provisioning and access workflows
Data incidents	The number of bugs, production incidents, or customer-reported defects that are conclusively traced back to poor data quality This can act as a proxy for unplanned work and rework, demonstrating the direct business impact of data dysfunction on software delivery instability and operational performance
Data quality	Pass rates of automated checks, including accuracy, completeness, and timeliness
Data freshness	Metadata from data source (for example, "last updated" timestamps)

Conclusion

Ultimately, a healthy data ecosystem is not an optional investment but a foundational amplifier for organizational performance. The success of AI adoption is conditional on the quality, accessibility, and unity of this data.

Healthy data ecosystems provide the reliable, context-aware foundation for the entire DORA AI Capabilities Model. However, high-quality data is only useful if AI tools can securely connect to it. This requires the next capability: making internal data AI accessible.

1. State of AI-assisted Software Development. https://dora.dev/dora-report-2025
2. 2017 State of DevOps Report. https://dora.dev/dora-report-2017
3. "DORA Capabilities: Documentation quality." https://dora.dev/capabilities/documentation-quality
4. 2016 State of the DevOps Report. https://dora.dev/dora-report-2016
5. "Survey questions: Data ecosystem." https://dora.dev/ai/capabilities-model/questions/#healthy-data-ecosystems

AI-accessible internal data

Rob Edwards
Application Delivery Lead, Google Cloud

AI having access to internal data amplifies AI adoption's positive influence on:

- Individual effectiveness
- Code quality

Why healthy data ecosystems matter for AI

For AI tools to be useful, they should be more than mere assistants: they need to become specialists that understand your organization. To make that leap, they must be securely connected to your internal data.

This connection is what delivers real value. Our research confirms that giving teams AI tools that can access this internal data directly improves their effectiveness and code quality.

Think of it as the complete context a developer needs to achieve an objective, which includes proprietary company data, such as codebases, wikis, architectural diagrams, documentation, and style guides—all of which can be managed through manual and automated context engineering.

TIP: While not strictly internal data, it's important to make current external documentation for internal tools AI accessible. An AI's training data is a static snapshot; providing up-to-date documents for your team's frameworks and languages is essential to prevent it from generating outdated code.

Defining the new discipline: A system, not a string

Many people are familiar with prompt engineering, which is the art of writing a specific command for an AI. To unlock real value, we must move to context engineering to ensure the model produces desirable, reliable, and controllable outputs.[1]

A simple prompt is like giving your AI assistant a single order. Context engineering, on the other hand, is like giving it a complete briefing packet before they start work.

This packet should contain all the context they need to do the job right, including:

Company data: "Here is our internal codebase and our coding style guide."

Latest information: "Here are the most up-to-date application programming interface (API) docs for the library you're using."

Tools and rules: "Here are the policies you must follow and the tools you can use to achieve or check your work."[2]

Instead of just a one-off string, context engineering is the system that automatically gathers this relevant information and then uses it to manage an intelligent workflow. This system enables iterative refinement loops, where the AI is instructed to generate a first draft, critique its own work against the context, and improve its answer.[3] It enables the AI to move from a generic "chatbot" to a specialized, expert assistant that understands your company.

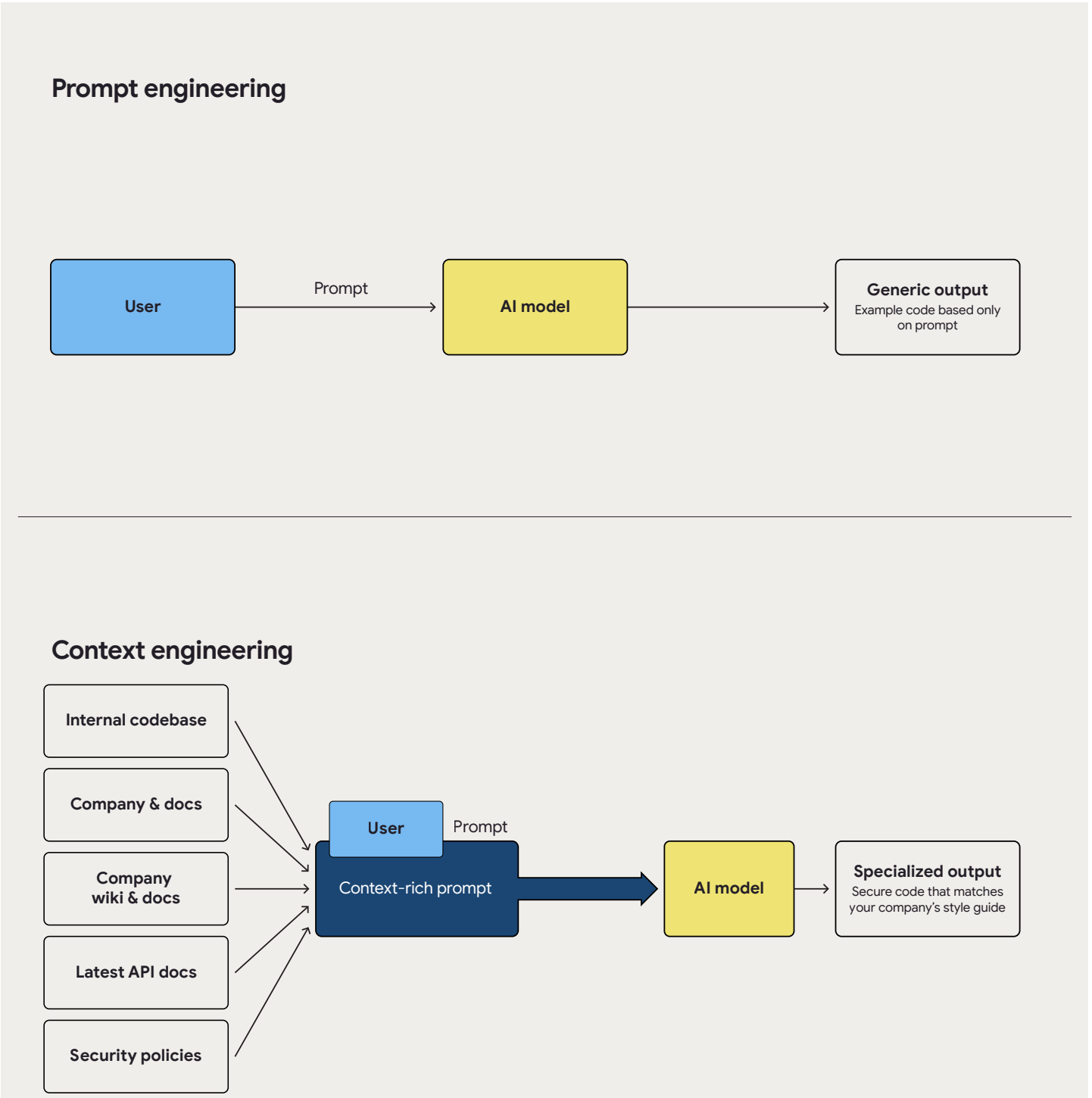


Figure 8: Example comparison of prompt engineering vs context engineering

Why AI-accessible internal data matters

Connecting AI to a complete context can elevate the AI from a general-purpose coding assistant to a specialized expert with deep knowledge of your organization's specific architecture, patterns, and business logic. It can help:

Boost developer efficiency: reduce the time developers spend on context gathering, including reading documentation, deciphering existing code, and interrupting senior engineers, by providing instant, context-aware answers.

Improve code quality and consistency: AI suggestions begin to align with internal coding conventions and best practices, leading to generated code that is more likely to be correct and performant within your specific systems.

Mitigate technical debt: By working from the latest framework and language documentation, the AI avoids suggesting deprecated functions and outdated patterns, helping teams build more maintainable and secure code from the start.

Accelerate onboarding and knowledge sharing: make institutional knowledge more accessible. New developers can get up to speed faster by asking the AI questions about the codebase, reducing interruptions to tenured team members.

These capabilities are powerful amplifiers for the benefits of AI adoption. DORA research shows with a high degree of certainty that AI-accessible internal data magnifies the positive impact of AI on key outcomes.

Specifically, it amplifies individual effectiveness and code quality. The AI's ability to provide answers tailored to your organization's unique context is a key differentiator between teams that see a marginal benefit from AI and those that achieve a step-change in performance.

AI-accessible internal data moderates AI's impact on individual effectiveness

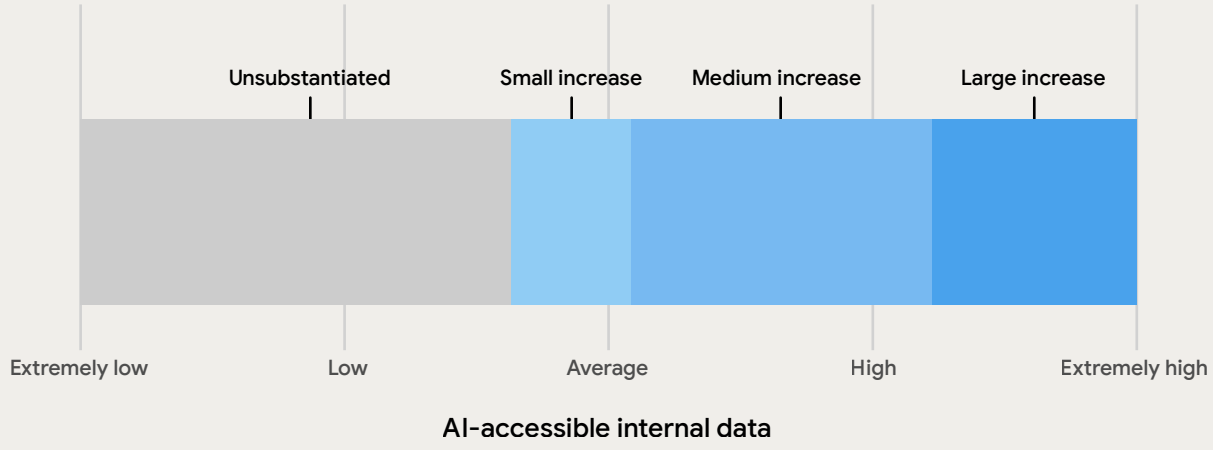


Figure 9: AI-accessible internal data moderates AI's impact on individual effectiveness

AI-accessible internal data moderates AI's impact on code quality

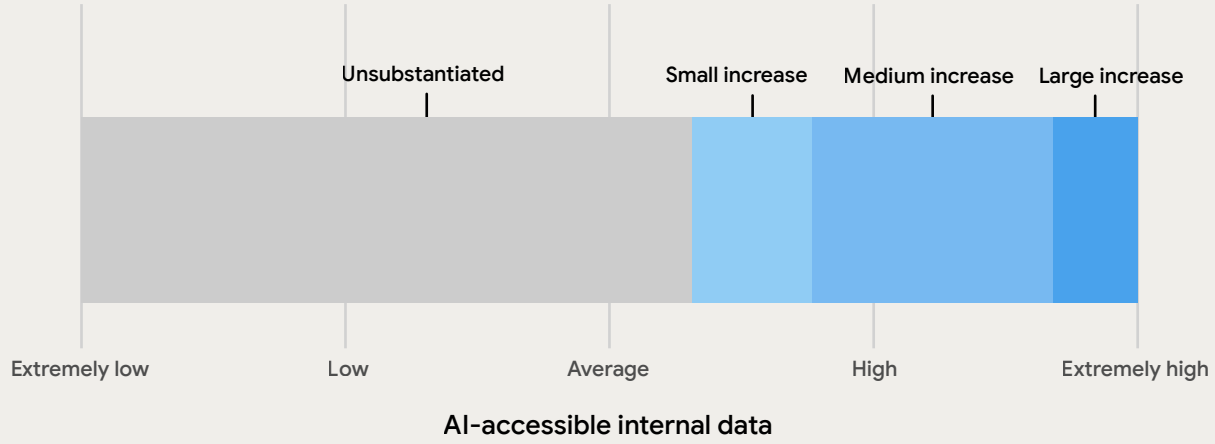


Figure 10: AI-accessible internal data moderates AI's impact on code quality

How to improve internal data accessibility

Improving accessibility to internal data involves a phased approach, starting with manual techniques and progressing toward a more systematic, automated solution.

Phase 1: Foundational and manual

Start by evolving your prompt engineering skills into manual context engineering. Teams should be empowered to move beyond just writing a query and toward assembling the context the AI needs to answer it.

Provide space for teams to learn how to manually find and attach the most relevant information (like specific API docs, style guides, and code snippets) to their requests. Create a shared, version-controlled library of these reusable context templates or briefing packets for common tasks.

Most importantly, reliable context depends entirely on accessible and accurate internal data. You can't connect a model or AI application to information that doesn't exist, and you shouldn't connect it to information that is wrong. Work with teams to identify and improve the most critical internal documentation first.

Phase 2: Automated pilot

Determine whether to build a custom solution or procure a commercial tool that can be securely integrated. Once a tool is selected, implement an automated retrieval solution for a single, high-impact use case.

This step highlights the importance of context engineering. While feeding large, raw documents into an AI's context window is tempting, it is often counterproductive. This approach frequently results in hallucinations: outputs that sound confident but are factually false.

Instead, effective solutions focus on providing only the most specific context for the current request. Two primary patterns are emerging for this:

- **Retrieval-augmented generation (RAG):**[4] A more complex but precise method for finding and providing only the most relevant, up-to-date information, which can be centralized.
- **A model context protocol (MCP) server:**[5] An approach that intelligently selects, structures, and feeds only the most relevant context to the AI, rather than entire raw documents.

This phase moves the needle from manual input to automated retrieval, which is a key step in maturing this capability.

Phase 3: Obtain sponsorship, scale, and operationalize

Use the pilot's success to obtain sponsorship from leaders for broader strategic investment. This investment is critical for scaling the capability, which often involves addressing foundational data challenges and building robust, secure internal APIs to make more data available to your AI systems. For documentation, this could involve integrating with a service that provides up-to-date documentation programmatically.

Example: Onboarding a new engineer

Imagine a new engineer joining a team responsible for a complex billing microservice.

Without AI-accessible data, their onboarding involves days of reading stale documentation, interrupting senior engineers with questions, and slowly piecing together a mental model of the system.

With AI-accessible data, they can ask the AI assistant directly: "What's the process for issuing a refund, and which services are involved?" The AI, using RAG to access the latest API specs, architectural diagrams, and the codebase itself, provides a step-by-step answer with links to the exact functions and documents.

This doesn't just accelerate onboarding; it empowers the new engineer to contribute value faster and more confidently.

Example: An AI-assisted code review

Consider a developer submitting a pull request (PR) for a new feature.

Without AI-accessible data, the human reviewer has to manually check for adherence to the team's specific style guide. They might also have a vague memory of a similar debate on a previous PR but can't find the link, leading to inconsistent feedback or repeated discussions.

With AI-accessible data, a code review agent can be configured to use the internal style guide and a database of past code review discussions as context. It could automatically comment:

"This is a great start. I noticed the error-handling pattern here is different from the one we established for this service in PR #1234. To maintain consistency, let's stick to that approach. You can see the discussion and reasoning here: [link to PR]."

The AI-accessible data can help offload tedious consistency checks from human reviewers. It can also adapt the AI's feedback to the organization's established norms, allowing the team to focus on the architectural and logical soundness of the change.

Common obstacles to data accessibility

Obstacle: Poor quality or "messy" internal data

Many organizations struggle with internal data that is outdated, inaccurate, and spread across multiple, disconnected systems. One of the most common challenges we see is that an AI connected to bad data will produce bad answers.

As one research participant noted, many companies "aren't even at a stage where they have their data properly organized." An AI connected to bad data will only produce bad answers.

Solution: Don't boil the ocean. A critical first step in your "data quality first" journey is to assign a team to a pilot project bound to a specific application or service. The team should seek to improve both the accuracy and the focus of the information available.

"Messy" data includes documentation that is simply too long. For example, a 20-page standards document is often too much for a model to pay attention to while also trying to solve a user's query. A better pattern is to:

- Identify the single most valuable and reasonably well-maintained data source (for example, the API documentation for one critical service).
- Use AI as a starting point to distill this information, then have a human subject-matter expert validate, edit, and approve this distilled summary (for example, `GEMINI.md`).[6]
- Focus on making that high-quality, distilled source accessible to your AI pilot first.
- Measure the improvement in AI response quality for that specific service, creating a business case to scale the effort.
- Prioritize declarative approaches and frameworks that allow for iterations, automated deployment pipelines, and testing including data quality assertions.

Obstacle: Risk of polluting the AI with bad examples

A common impulse is to index all code repositories to maximize the available context. However, this can be counterproductive.

If your indexes include deprecated projects, experimental branches, and code that violates current best practices, the AI will learn from this "bad" code just as easily as the "good" code. This leads to poor-quality suggestions that replicate outdated patterns and technical debt.

Solution: Curate your code context. Be selective and strategic about what you index. Start by indexing only the repositories and branches (like `main`) that represent your organization's gold standard for code style, architecture, and current patterns.

Curating your code context ensures the AI learns from your best examples, amplifying good practices rather than bad ones. You can gradually expand the index as you validate the quality of other code sources, treating your indexed code as a "curriculum" for your AI assistant.

Obstacle: Context rot

Many teams assume that "more data is better" and try to provide as much context as possible. However, large language models have finite context windows.

Overwhelming the model with excessive, irrelevant, or low-density information can be as detrimental as providing too little. The key, relevant signal can get "lost," leading to unfocused, incorrect, and generic responses that ignore the crucial details.

Solution: Focus on relevance, not volume. This is where precision and experiential understanding can help with context engineering. Instead of large data dumps, implement precise retrieval mechanisms like RAG.

RAG is designed to search a vast corpus (your entire wiki, all your code) but retrieve only the specific "chunks" of information that are most relevant to the user's immediate query. It ensures the prompt is packed with high-signal, relevant data, rather than being bloated with noise. This focus on relevance is leading to an emerging discipline of context harvesting, the active process of filtering for relevance, not volume.

Obstacle: Security and access control concerns

A primary concern is ensuring the AI tool respects existing permissions and access controls on sensitive data. A related major obstacle is controlling which AI services (like an approved MCP server) a user can connect to, and ensuring those services themselves are secure.

Solution: Implement a layered, "least privilege" security model. This strategic approach aligns with principles outlined in the Secure AI Framework (SAIF),[7] an industry model that provides comprehensive guidance on managing AI risks.

- For data access: When using automated systems like RAG, ensure the retrieval mechanism operates with the user's own credentials. This guarantees the system can only "see" and retrieve documents that the user is already authorized to access.
- For service access: Don't allow ad-hoc connections to unvetted, external AI services. Instead, channel requests through an approved, centralized proxy or MCP server that is vetted, monitored, and managed by IT. This can help mitigate data exfiltration to unauthorized models and ensures all AI interactions adhere to company security policy.

How to measure internal data accessibility

DORA survey questions

DORA research measured this capability by asking these questions:[8]

- How often have you inputted or provided internal company information or data (for example, documents, code, spreadsheets, text) to an AI tool?
- Over the last month at work, how frequently have you used AI in your organization to help you retrieve and use information from internal data sources (for example, for answering questions, generating reports, or improving workflows)?
- To what extent do the models have access to internal data sources (for example, databases or codebases) and applications (for example, wikis or work management systems)?
- When using an AI tool for your work, how often have its responses or outputs seemed to you to have utilized internal company information or data as context (for example, by referencing specific internal projects, data, or terminology in its explanations)?
- When you have asked an AI tool about internal company matters (for example, specific project details, internal reports), how often did its response lead you to believe it had accessed internal company information or data to generate that response?

Additional measures

There may be other factors that can help you assess the extent to which internal data is accessible to AI and how that is helping improve performance.

Factor to test	What to measure
Retrieval event frequency	The number of times the AI system successfully executes a RAG query
Data source access rate	Which data sources are being accessed
AI retrieval latency (time-to-context)	Elapsed time from when an AI system identifies a need for context to when it successfully retrieves that data
Context-rich prompts	The proportion of prompts that are "context-rich" versus "simple prompts"
New developer onboarding	Time to Nth change delivered to production. This might be the developer's first, tenth, or other change that gets delivered

It is reasonable to expect that access to internal data also increases the impact of AI on other factors such as cognitive load. To assess this you might include survey questions that use a scale with responses ranging from "strongly disagree" to "strongly agree." For example:

- Context-aware AI responses have reduced my efforts in finding information.
- Context-aware AI responses have improved my ability to understand our codebase.
- Context-aware AI responses require less effort to verify.
- Using context-aware AI assistants reduces the amount of time I spend seeking information.

Conclusion

Connecting AI to internal code and documentation, or "context engineering," transforms the AI from a generic assistant into a specialized expert.

AI-accessible internal data directly boosts individual effectiveness and code quality by providing relevant, internal context.

As this enhanced AI accelerates the volume and velocity of code generation, it creates new risks.

To manage this increased speed safely, teams must rely on the essential safety net provided by the next capability: strong version control practices.

1. "The rise of 'context engineering'." https://blog.langchain.com/the-rise-of-context-engineering
2. "Rules files for safer vibe coding." https://www.wiz.io/blog/safer-vibe-coding-rules-files
3. "Prompting techniques for secure code generation: A systematic investigation." https://arxiv.org/abs/2407.07064
4. "What is retrieval-augmented generation (RAG)?" https://cloud.google.com/use-cases/retrieval-augmented-generation
5. "What is the MCP and how does it work?" https://cloud.google.com/discover/what-is-model-context-protocol?e=48754805&hl=en
6. "Provide context with GEMINI.md files." https://geminicli.com/docs/cli/gemini-md/
7. "Cloud CISO Perspectives: AI as a strategic imperative to manage risk" https://cloud.google.com/blog/products/identity-security/cloud-ciso-perspectives-ai-strategic-imperative-to-manage-risk
8. "Survey questions: Internal data." https://dora.dev/ai/capabilities-model/questions/#ai-accessible-internal-data

Strong version control practices

Rob Edwards

Application Delivery Lead, Google Cloud

Dave Stanke

Specialist Customer Engineer, Google Cloud

Strong version control practices amplify AI adoption's positive influence on:

- Individual effectiveness
- Team performance

A foundational capability for modern software development

Version control systems are the bedrock of modern software development, providing a systematic way to manage and track changes to files over time. Systems like Git,[1] Apache Subversion,[2] Mercurial,[3] and Jujutsu[4] allow development teams to coordinate work, track a project's history, and maintain a definitive source of truth for all project assets.

As a cornerstone of high-performing technology organizations, version control is about the comprehensive use of a version control system for all production artifacts.

As far back as the 2014 State of DevOps Report,[5] DORA has emphasized that comprehensive use includes not just application code, but also system configurations, test and deployment scripts, and infrastructure definitions.

In the age of generative AI, where the volume and velocity of code generation are dramatically increasing, our research shows that mature version control habits are more crucial than ever for maximizing AI's benefits while mitigating its risks.

Version control provides several key benefits:

Recovery and stability: It helps reduce failed deployment recovery time. When an error occurs, version control makes it easier to pinpoint the cause of failure and roll back to the last known good state quickly and reliably.

Speed and throughput: It enables the easy recreation of environments for testing and troubleshooting, which boosts throughput.

Disaster recovery and auditability: It ensures teams can reproduce their production services quickly and predictably, providing complete traceability of every change for compliance purposes.

Psychological safety: Frequent check-ins and a reliance on rollback capabilities create a crucial psychological safety net. This safety net allows development teams to experiment and innovate with confidence, knowing they can easily revert to a stable state if a mistake is made.

Why version control matters for AI

The growing use of generative AI amplifies the importance of strong version control practices. AI-assisted coding can increase throughput, but our research shows it is also associated with increased software delivery instability. Strong version control practices are the essential safety net that allows teams to experiment with AI-generated code confidently.

AI transforms the principle of frequently committing small changes from a best practice into a critical safeguard. As development moves further into an agentic world—where AI assistants and agents generate and apply code changes autonomously—this discipline becomes paramount. Each change, whether made by a human or an AI, should be committed as a distinct, atomic unit. This creates a clear auditable trail and allows for immediate and precise rollbacks as necessary.

Version control hygiene (the consistent practice of making small, frequent, and well-organized commits) is critical to both the "inner loop" (a developer's local work crafting a change proposal) and the "outer loop" (the team's review and merge process to ultimately deliver it).

Practical observation shows that developers often treat the inner loop with less rigor, waiting to create clean commits before opening a pull request. More disciplined version control practices typically appear in the outer loop.

AI-assisted coding fundamentally changes how teams can enact version control best practices. AI tools can augment a developer by writing code in new ways, introducing powerful nondeterminism directly into the developer's workflow.

AI suggestions for documenting version control can be delightful breakthroughs or flawed detours. This "trial and error" cycle makes the old "commit when ready" model insufficient.

To effectively harness AI, developers must now bring the discipline of the outer loop into their inner loop. This means creating small, frequent commits at every clean, functional state. When an AI-generated change is undesirable, it's simple to discard it and revert to a known good state.

In the outer loop, version control provides the scaffolding for human-in-the-loop review of changes before they are merged. A change proposal may reflect the work of any number of human and AI agents, each with a unique set of knowledge and preferences.

While divergence here can be an asset, as a vector for innovation, the convergent process of code review and automated testing is necessary to ensure that changes conform to organizational standards and that they are in service of the organization's goals. Consistent use of version control allows every change to be thoroughly vetted.

With a high degree of certainty, we found that AI adoption's positive benefits depend on respondents' frequency of version control commits. Specifically, in the presence of frequent commits, AI's positive influence on individual effectiveness is amplified.

Version control commit frequency moderates AI's impact on individual effectiveness

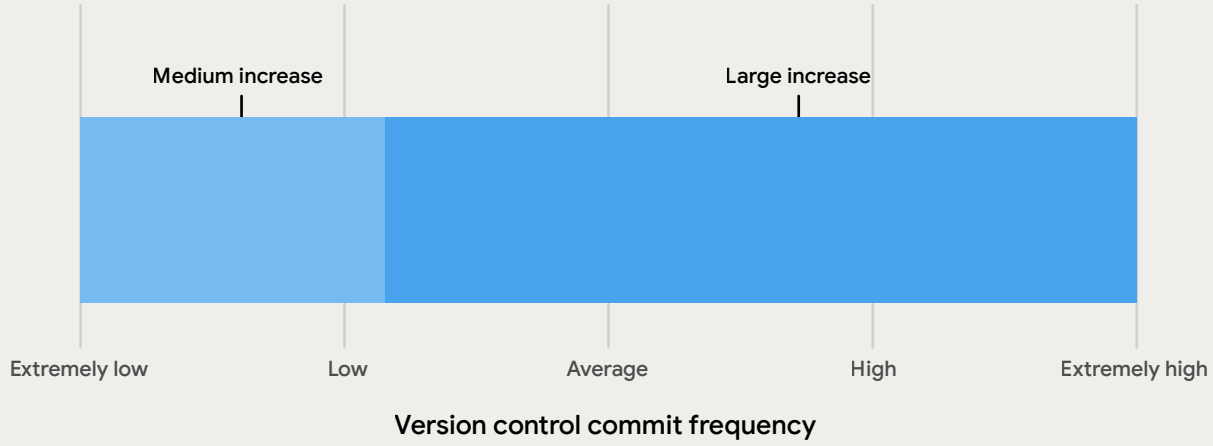


Figure 11: Version control commit frequency moderates AI's impact on individual effectiveness

Additionally, we found that AI adoption's positive benefits depend on respondents' frequency of use of their version control systems' "rollback" features to undo or revert changes. Specifically, in the presence of more frequent rollbacks, AI's positive influence on team performance is amplified.

This reliance on rollbacks for team performance likely stems from the challenge of managing AI-generated code.

As our research suggests, AI can increase software delivery instability, often by creating large, hard-to-review batches of code. The ability to quickly revert these changes becomes a critical factor in maintaining team performance.

We found that about 21% of our survey respondents are beginning to store AI prompts in version control. Prompts contain valuable context about a task. By versioning them, teams can share knowledge, refine their ability to get value from AI, and create an audit trail for agentic workflows.

Ability to rollback moderates AI's impact on team performance

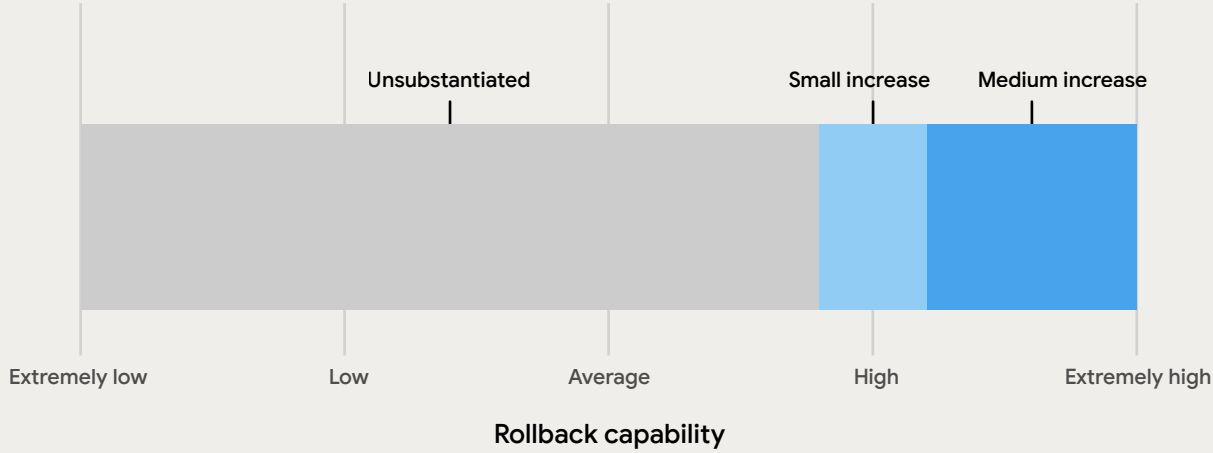


Figure 12: Ability to rollback moderates AI's impact on team performance

How to improve version control practices

How teams use version control is just as important as whether they use it. We found that a few key practices help teams unlock more value from their version control systems:

Version everything: Store all production artifacts in version control. This includes application code, system configurations, test scripts, deployment scripts, container orchestration files, and cloud configuration files. The goal is to be able to reproduce testing and production environments in a fully automated fashion using only information from version control.

Promote trunk-based development:[6] Encourage a branching strategy that minimizes long-lived branches and promotes frequent integration. This helps mitigate the risk of frequent or complex merge conflicts.

Make small, frequent commits: Committing small, related changes frequently, whether they are human- or AI-generated, makes it easier to understand progress and track the history of a feature. This practice reduces the risk of complex merge conflicts and makes it simpler to revert to a known good state if a change introduces a problem.

Write clear commit messages: Commit messages tell the story of a codebase. By clearly explaining the "why" behind a change, not just the "what", teams create an invaluable historical record. Adopting a standard like Conventional Commits[7] can further improve clarity and enable automation based on commit history.

Store AI prompts: Prompts contain valuable context about a task. By versioning them, teams can share knowledge, refine their ability to get value from AI, and create an audit trail for agentic workflows.

Store agent configuration files: Agent configuration files, such as `GEMINI.md` or `CLAUDE.md`, provide context and instructions for AI agents. These files can be an important source of team norms and practices and can provide important guardrails for AI assistants to follow.

Bringing AI onto the team means you'll need to rethink your version control playbook. This is the perfect time to get on the same page and set some new ground rules. The first one you have to tackle is the biggest:

The core philosophy: First, agree as a team: Are we letting an AI commit changes directly, or does a human always have to review and commit the code locally?

Once you have an answer for that, the other questions start to fall into place:

Commit history: AI-assisted development can dramatically increase the volume and velocity of commits. How will the team manage this expanded history? This is a good time to define and align on clear, team-wide rules for using `rebase`, `squash`, and `merge`.

Commit messages: What is the team's standard for a "good" commit message? To ensure all changes are understandable, agree on a consistent format. For example, teams should decide if they want to include the prompt when describing the reasoning for a change.

Batch size: The practice of working in small batches is a safeguard. Teams should consider how they will ensure that all contributions are small, logical changes instead of one giant, hard-to-review code dump.

Writing these new rules down is critical. It gives your team a clear guide and even builds a knowledge base, which can then be used by the AI-accessible internal data capability.

How to improve version control practices

As teams develop their version control practices, two significant obstacles often emerge:

Limited use: One of the most common pitfalls is storing only application source code in version control. Teams should be able to reproduce testing and production environments in a fully automated fashion with the requisite infrastructure and data along with the scripts, source code, and configuration information from version control.

Merge conflicts: Conflicts arise when a version control system can't automatically resolve competing changes to the same lines of code. While unavoidable at times, frequent or complex merge conflicts are often a sign of process issues. Teams can mitigate this risk by ensuring that development branches are short-lived and by adopting foundational practices like continuous integration[8] and trunk-based development.[9]

How to measure version control practices

DORA survey questions

DORA research measured this capability by asking these questions:[10]

- For the primary application or service you work on, which of the following assets are managed in a version control system?
  - Application code
  - Application configurations
  - Code for automating build and configuration
  - Prompts for AI systems
  - System configurations
- Thinking about the most recent change that was committed to the primary application or service you work on, approximately how many lines were changed (added, edited, or removed) as part of that commit?
- When actively changing code or configuration for the primary application or service you work on, approximately how frequently do you commit changes to version control?
- When actively changing code or configuration for the primary application or service you work on, approximately how much do you rely on the ability to undo, revert, or rollback changes?

Additional measures

Additional signals that may help measure a team's version control practices include:

Factor to test	What to measure
Time to rebuild an environment	Practice rebuilding an environment using only the required infrastructure, data, and version control system
Active branches on the application's code repository	Measure how many active branches you have on your application repositories' version control systems, and make this number visible to all teams. Then track the incremental progress toward the goal state
Frequency of merging branches and forks to trunk	Measure either a binary (yes/no) value for each branch that's merged, or measure a percentage of branches and forks that are merged every day
Merge conflict rate	Track the number or percentage of merge operations (for example, merging a feature branch to main) that result in a conflict requiring manual resolution

Conclusion

Version control is more than just a tool for storing code; it's a foundational capability that underpins many other advanced practices. It's a prerequisite for continuous delivery[11] and for creating the kind of flexible infrastructure[12] that allows organizations to adapt and thrive.

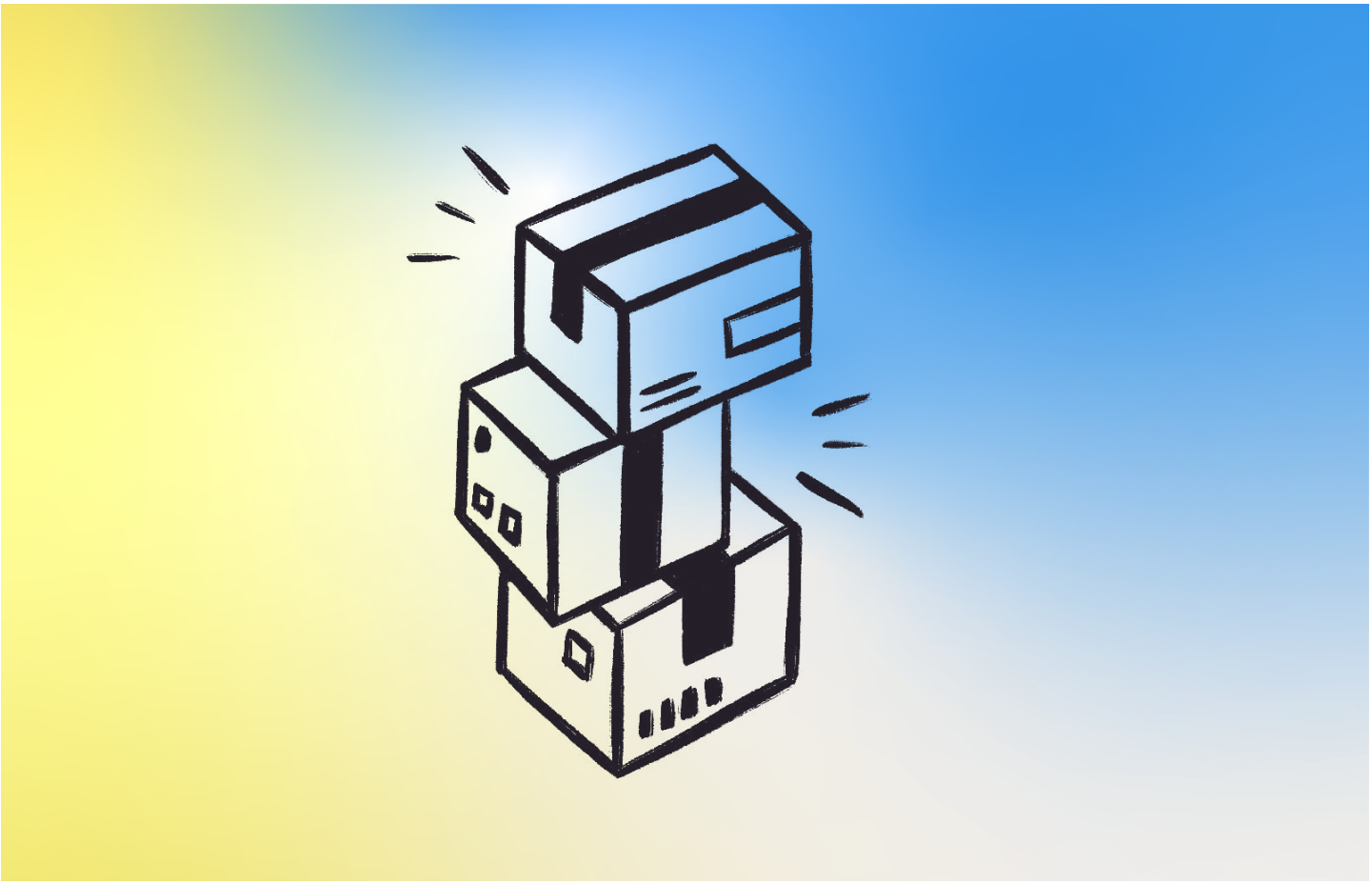
DORA has long asserted the need for strong version control practices. Today, that need is stronger than ever. As AI becomes integrated into all aspects of software development, code is being created more quickly—and less predictably—than ever before.

This dynamism unlocks tremendous creative capacity but also brings new forms of risk. Version control is the essential safety mechanism that facilitates experimentation: in the software that teams build, as well as in the ways they build it.

As development moves further into an agentic world—where AI agents generate and apply changes autonomously—this discipline becomes paramount. Each change, whether from a human, an AI, or a collaboration between multiple parties, must be a distinct, atomic unit with a clear message, creating an auditable trail that allows for immediate and precise rollbacks.

This safety net is most effective when combined with the specific discipline of working in small batches.

1. Git. https://git-scm.com
2. Apache® Subversion®. https://subversion.apache.org
3. Mercurial. https://www.mercurial-scm.org
4. Jujutsu—a version control system. https://github.com/jj-vcs/jj
5. 2014 State of DevOps Report. https://dora.dev/research/2014
6. "Trunk-based development." https://dora.dev/capabilities/trunk-based-development
7. "Conventional Commits." https://www.conventionalcommits.org
8. "Continuous integration." https://dora.dev/capabilities/continuous-integration
9. "Trunk-based development." https://dora.dev/capabilities/trunk-based-development
10. "Survey questions: Version control." https://dora.dev/ai/capabilities-model/questions/#strong-version-control-practices
11. "Continuous delivery." https://dora.dev/capabilities/continuous-delivery
12. "Flexible infrastructure." https://dora.dev/capabilities/flexible-infrastructure



Working in small batches

Rob Edwards

Application Delivery Lead, Google Cloud

Working in small batches amplifies AI adoption's positive influence on product performance.

Working in small batches decreases

- Friction
- Individual effectiveness

One change at a time: How small batches unlock performance gains

Building a successful application is a game of improving feedback on the units of work a team delivers. The team needs to know if the change has the correct syntax, if it integrates with the rest of the codebase, if it meets operational requirements, if it accomplishes its goal, and, ultimately, if users are happy with the change.

Working in small batches is the degree to which teams break down their work into the smallest possible chunks that can be completed and delivered independently. Each batch of work should follow the INVEST principle[1] to ensure it's high-quality and valuable:

Independent: It can be developed and deployed without depending on other pieces.

Negotiable: It's not a strict contract; the details can be refined over time.

Valuable: It delivers real value to a user or stakeholder.

Estimable: You can get a rough idea of how long it will take to build.

Small: It can be completed in a short timeframe, ideally hours, not weeks.

Testable: You can verify that it works as expected.

The capability to work in small batches is a cornerstone of both lean product management[2] and continuous delivery.[3] It changes the economics of the software development process, making it cheap and easy to get changes out the door and into the hands of users.

Working in large, infrequent batches is risky, slow, and demotivating. It introduces significant risk to the deployment process, makes recovery from failure difficult, and creates long, painful integration phases that kill productivity.

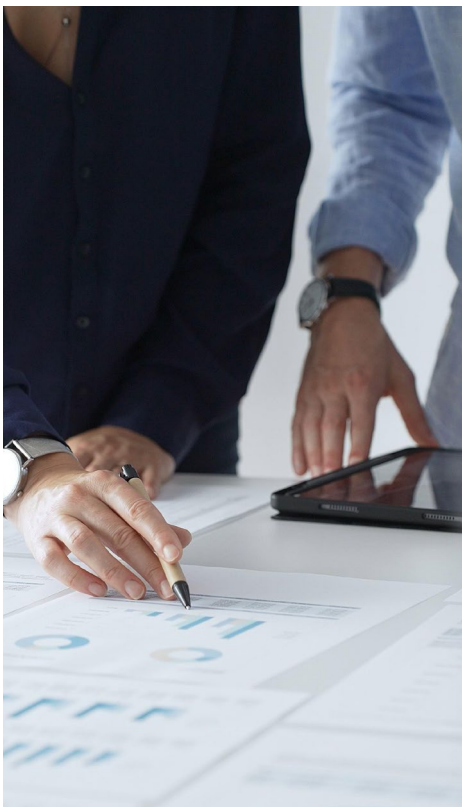
Working in small batches is a central tenet of continuous integration[4] and continuous delivery,[5] and it dramatically improves performance and well-being by:

Creating faster feedback loops
When you release a small change, you find out almost immediately if it works. This makes it easier to triage issues, fix bugs, and learn from your decisions.

Building quality in
The practice helps build quality into the process and minimizes the rework associated with fixing errors found late in the cycle. This is amplified when combined with strong test automation.

Increasing efficiency and motivation
Teams see their work go live quickly, which is a huge motivator. It also prevents the organization from succumbing to the sunk-cost fallacy.

Enabling lean product management
The practice allows for rapid release of release of minimum viable products and frequent and frequent gathering of user feedback through techniques like A/B testing.



Why working in small batches is critical for AI

Working in small batches is a critical countermeasure to the risks of AI-assisted development. While AI can generate large amounts of code quickly, large changes are difficult to review, test, and integrate safely.

Small-batch work can help prevent AI-accelerated development from leading to increased instability. It forces a disciplined approach where the focus shifts from raw code generation to thoughtful decomposition, prompting, and verification.

DORA research shows that, when paired with AI adoption, working in small batches:

Amplifies product performance
It has a clear, positive impact on the performance of the product.

Reduces friction and rework
It turns AI's otherwise neutral effect on organizational friction into a net positive, helping teams overcome downstream chaos and avoid disruptive deployments.

Balances individual effectiveness
AI tools excel at helping developers generate a lot of code quickly. The discipline of small batches puts a natural limit on this, which slightly reduces the perceived individual effectiveness gains from AI, but channels that productivity toward a more stable and sustainable delivery process.

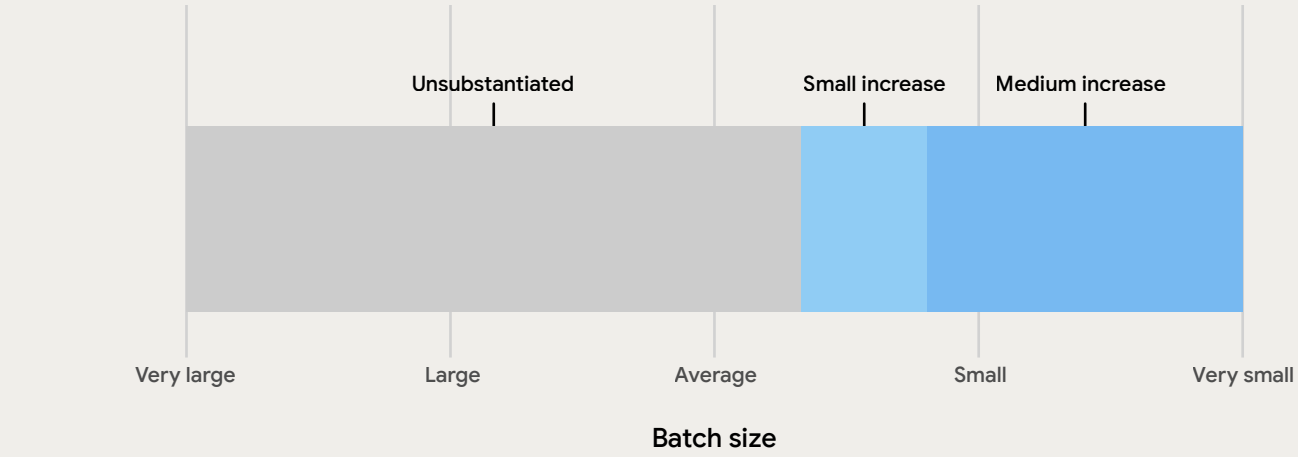


Figure 13: Batch size moderates AI's impact on product performance

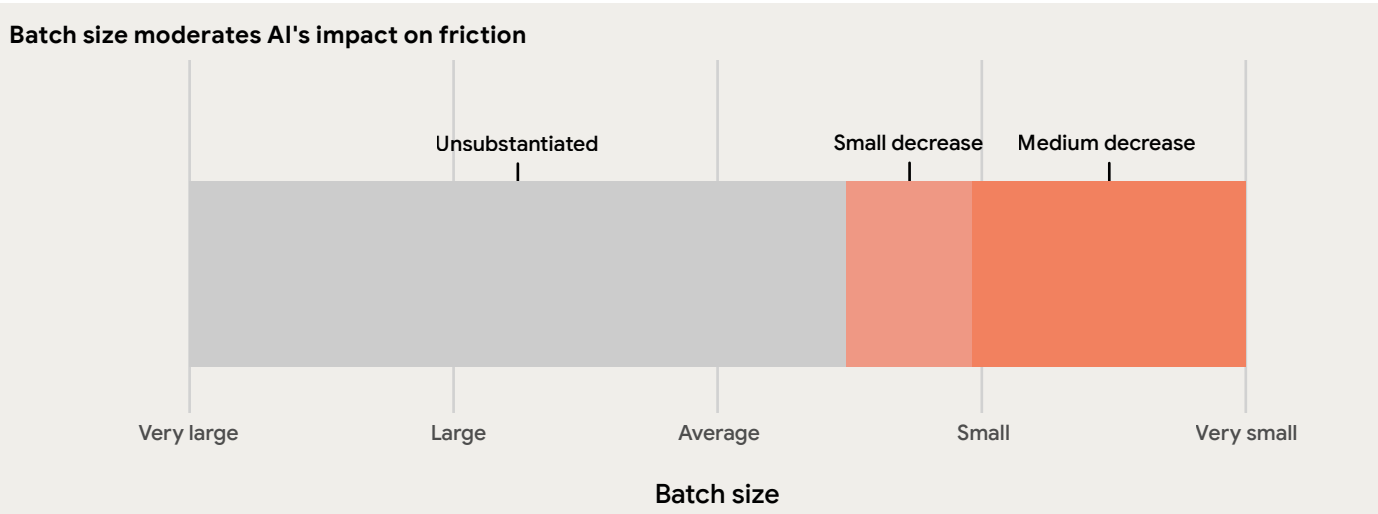


Figure 14: Batch size moderates AI's impact on friction

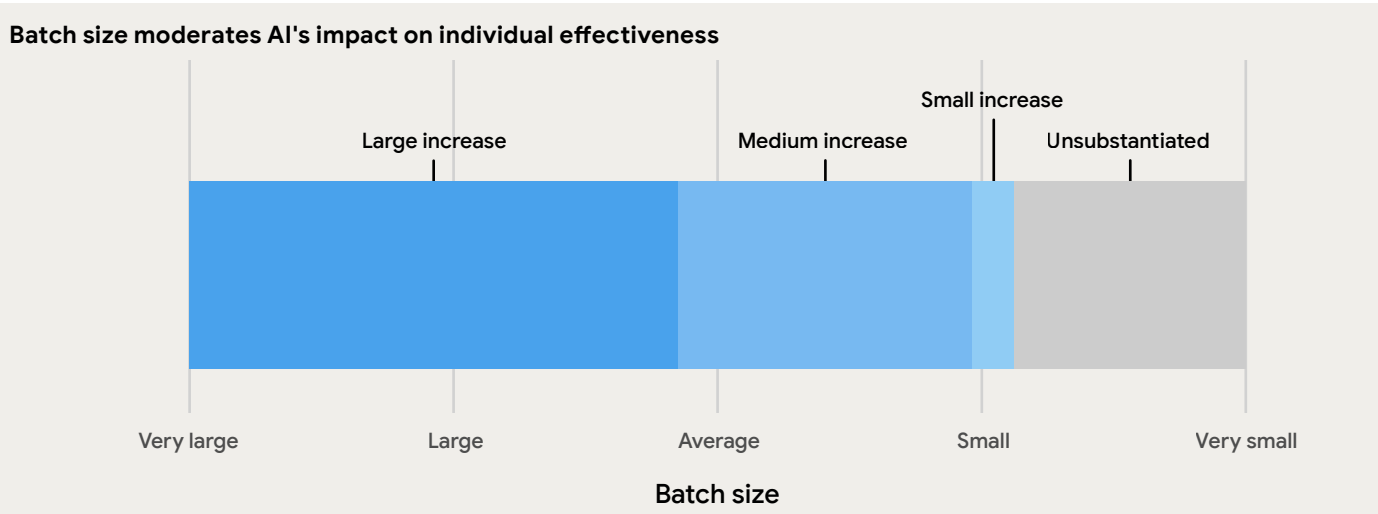


Figure 15: Batch size moderates AI's impact on individual effectiveness

Our research uncovered a curious finding: while AI has a positive impact on individual effectiveness, our data suggests that these benefits are slightly reduced in teams that are working in small batches. Why might this be? We have a few hypotheses:

Overhead hypothesis: AI assistants can be very effective at generating large blocks of code for a well-defined problem. The human effort required to decompose a larger problem into many small, discrete, and well-formed prompts may offset some of the raw code-generation speed. The developer's work shifts from writing code to decomposing, prompting, and verifying.

Review friction hypothesis: The cognitive load for a human to review and understand a small chunk of unfamiliar, machine-generated code may be higher per line than reviewing human-written code from a trusted colleague. This added friction at each small batch could slow the overall process, negating some of the productivity gains from rapid generation.

Tooling mismatch hypothesis: Today's AI development tools may be better optimized for generating larger, more complete features or files. The workflow of using these tools to create many small, discrete pull requests might be cumbersome, adding a layer of manual overhead that erodes productivity gains.

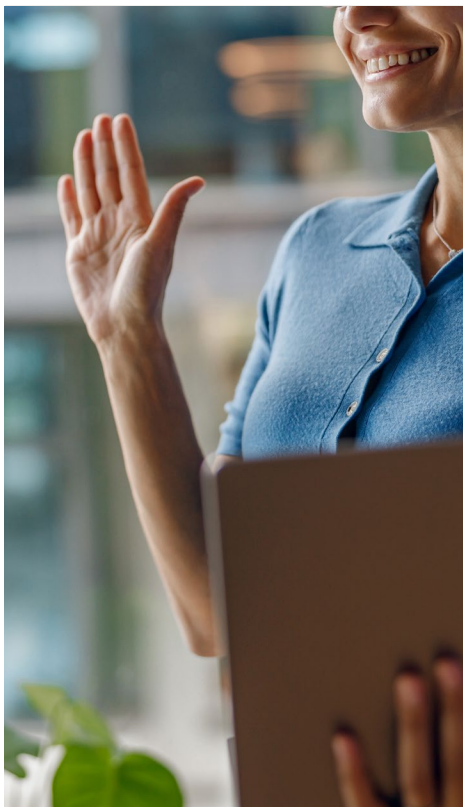
More importantly, we argue that individual effectiveness should not necessarily be pursued as a goal in and of itself. Rather, individual effectiveness is a means to realize greater organizational, team, and product performance, and improved developer well-being.

While this necessary shift from raw code generation to thoughtful decomposition and verification may feel like a loss of individual speed, it is precisely this discipline that can unlock sustainable team-level performance, and help prevent downstream chaos.

In this case, working in small batches increases reported product performance, while also decreasing perceived friction for AI-assisted teams. We think these benefits outweigh any potential harm to individual effectiveness from working in small batches—in addition to those benefits of working in small batches that have been long-proven as part of our DORA Core Model.[6]

These benefits are significant, but they rely on mastering the fundamentals. Whether your team is using AI-assisted tools or not, the core techniques for shrinking batch size remain the same.

How to work in smaller batches



The goal is for work units to be completed in a shorter timeframe, ideally lasting only hours to a couple of days. Any batch of code that takes longer than a week to complete is too big.

Refine story-slicing skills
Train product owners and teams on how to break large user stories into smaller, vertical slices of value.

Commit to trunk daily
Developers must commit and merge their work into the mainline (trunk) at least once per day. This is a necessary condition for both continuous integration[7] and trunk-based development.[8]

Embrace feature flags
Use feature flags (or other progressive delivery methods) to decouple deployment from release. This is a critical technical practice that allows incomplete features to be merged to the main branch and deployed to production without being visible to users.

Set work-in-process (WIP) limits
Use Kanban-style boards with explicit WIP limits to discourage multitasking and encourage the completion of work.

This makes bottlenecks visible and forces the team to focus on getting work to "done."

Example: Slicing a "save for later" feature

A product manager asks to add a "save for later" button to items on a product page. A large-batch approach would be to build the entire feature on a branch and release it when it's all done. A small-batch approach looks like this:

Batch 1 (Back-end): Create the new API endpoint (`POST/api/v1/items/{id}/save`) and the database change to store the saved item. This is deployed to production but is not yet called by any client.

Batch 2 (Back-end): Create the GET and DELETE endpoints for managing saved items. Also deployed but unused.

Batch 3 (Front-end): Add the "save for later" button to the user interface, but hide it behind a feature flag. The button's onClick handler calls the new API. This is deployed, but the feature is turned off for all users.

Batch 4 (Testing): The team enables the feature flag in a testing environment to verify the end-to-end flow.

Batch 5 (Release): The feature flag is enabled for a small percentage of users to validate its performance and usefulness before a full rollout.

Each batch is a small, independent, and deployable unit of work that can be completed in a day or less.

Common obstacles to working in small batches

As teams work to decrease batch sizes, they must avoid the trap of local optimization. A common pitfall is for a development team to optimize for completing their own work quickly and handing it off to the next team in the process, such as quality assurance (QA) or operations. This can lead to work piling up, creating a bottleneck that slows down the entire system.

A better approach is to take a full view of the system and prioritize the flow of small batches all the way through to completion. This often involves implementing WIP limits[9] and making work and bottlenecks visible,[10] both of which can support the team in addressing systemic issues.

As the saying goes, "Stop starting, start finishing." This favors completing a small number of tasks over starting many and juggling them simultaneously.

Obstacle: "This feature is too big to be broken down"
This is a common feeling, but it's rarely true. The key is to shift from thinking about horizontal layers (for example, "build the whole user interface, then the whole backend") to vertical slices. Start with the smallest possible piece of end-to-end value, even if it's not user-facing at first, like an API endpoint. The "branch by abstraction"[11] technique is a powerful pattern for making large-scale refactors incrementally.

Obstacle: "My product manager wants to see the whole feature before we release it"
This is a communication and trust issue. One solution is to use feature flags. Explain that the code is being released to production continuously, which reduces risk, but the feature will remain invisible to users until it's complete and ready for review. This decouples technical deployment from business release, giving product management full control over when the feature goes live.

How to measure batch size

DORA survey questions

DORA research measured this capability by asking these questions:[12]

- For the primary application or service that you work on, approximately how long does it take a developer to complete the work assigned in a single task (for example, card, ticket, or story)?
- For the primary application or service that you work on, approximately how many changes (for example, pull requests, merge requests, or change lists) are combined into a single release or deployment?
- Thinking about the most recent change that was committed to the primary application or service you work on, approximately how many lines were changed (added, edited, or removed) as part of that commit?

Questions for team assessment

Discussing batch size with your team can reveal opportunities for improvement in your workflow and architecture.

- How can we break our work down into smaller pieces that can be released independently?
- What is the largest source of delay in our deployment pipeline? How can we reduce it?
- Are our WIP limits surfacing obstacles? Are we addressing those obstacles, or just relaxing the limits?
- When a bug is found in production, how many separate changes do we typically have to investigate to find the cause?

Additional measures

Additional signals that may help measure working in small batches include:

Factor to test	What to measure
Lead time for changes	The time it takes for a code commit or change to be successfully deployed to production
Frequency of merging branches and forks to trunk	Measure either a binary (yes/no) value for each branch that's merged, or measure a percentage of branches and forks that are merged every day
Decoupling releases	Measure the number or percentage of changes deployed to production but not immediately available to all users

Conclusion

Working in small batches is an essential discipline that channels AI's generative speed into sustainable performance. By forcing large, AI-generated changes into small, reviewable, and testable units, small-batch work acts as a critical amplifier. It translates potential individual efficiency gains into real-world product performance and reduced friction.

When small-batch work is combined with version control, teams can move quickly and safely. The next step is to ensure they are moving in the right direction, which requires a strong user-centric focus.

1. "INVEST (mnemonic)." https://en.wikipedia.org/wiki/INVEST_(mnemonic)
2. 2017 State of DevOps Report. https://dora.dev/dora-report-2017
3. "Continuous delivery." https://dora.dev/capabilities/continuous-delivery
4. "Continuous integration." https://dora.dev/capabilities/continuous-integration
5. "Continuous delivery." https://dora.dev/capabilities/continuous-delivery
6. "DORA's Research Program." https://dora.dev/research
7. "Continuous integration." https://dora.dev/capabilities/continuous-integration
8. "Trunk-based development." https://dora.dev/capabilities/trunk-based-development
9. "Work in process limits." https://dora.dev/capabilities/wip-limits
10. "Visibility of work in the value stream." https://dora.dev/capabilities/work-visibility-in-value-stream
11. "Trunk based development: Branch by abstraction." https://trunkbaseddevelopment.com/branch-by-abstraction
12. "Survey questions: Working in small batches." https://dora.dev/ai/capabilities-model/questions/#working-in-small-batches

User-centric focus

Dave Stanke
Specialist Customer Engineer, Google Cloud

Amanda Lewis
Developer Relations Engineer, Google Cloud

User-centric focus moderates the impact of AI adoption on team performance.

- Low levels of user-centricity lead to decreased team performance
- High levels of user-centricity lead to increased team performance

All software is made for human users. The true worth of any technology, from a highly visible front-end interface to an unseen back-end process, is determined by its usefulness to the individual at the conclusion of the value chain. However, complex technology stacks, and the intricate supporting technologies used to build and operate them, can obscure the essential connection between the act of creating software and the benefit it eventually provides.

It's far too easy to fixate on the internal mechanisms of software development—debating framework choices, optimizing for corner cases, fighting organizational bureaucracy—and lose sight of the overarching purpose of that software.

While DORA's research has demonstrated that attention to intermediate outcomes (such as software delivery metrics) in the software development lifecycle is essential to overall performance, teams must also continuously align their priorities in service of the end user.

In addition to improving product quality[1] and predicting a 40% improvement in organizational performance,[2] user-centric focus can also improve the quality of life for developers, lifting job satisfaction and productivity while reducing burnout.

Why user-centric focus matters for AI

Prior to the advent of autonomous coding systems, a human (the developer) would be intimately involved in crafting each line of code, drawing on their human intuition and experience. Now, much of that code is generated by AI. Therefore, it's incumbent on the developer to be a proxy for the user—to continuously refocus AI tools toward user-centric objectives. Engineers should seek to improve their understanding of user needs and prioritize efforts to describe those needs thoroughly within the context provided to AI models.

DORA's 2025 research showed that, as teams with a strong user focus adopt AI tools, their effectiveness grows. But the converse also holds: a team with poor user focus that adopts AI is likely to see their team performance decrease. If a team's priorities are not aligned to the North Star of user needs, amplification with AI can propel them even further in the wrong direction.

User-centric focus moderates AI's impact on team performance

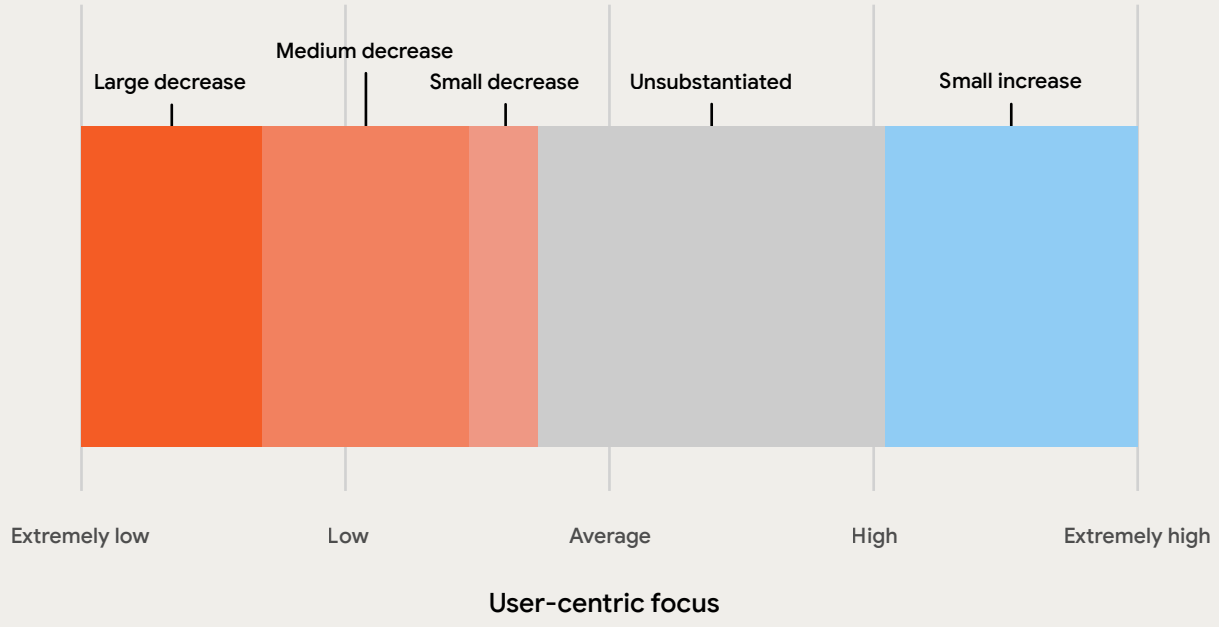


Figure 16: User-centric focus moderates AI's impact on team performance

How to improve user-centric focus

As AI models are integrated into teams' toolsets, those models must be provided with precise, current, granular end-user context, such as behaviors, pain points, and desired outcomes. A cultural shift is needed, making user stories, feedback, and analytics essential. Some techniques for developing a team's user-centric focus include:

Integrate user feedback loops

Actively solicit and make use of direct user feedback. Create channels for user feedback, such as integrating in-app, one-question surveys right after a user completes a critical task. Supplement quantitative data with qualitative insights from user interviews and structured beta testing programs. Establish a continuous, low-latency loop: feedback comes in, it is immediately available for the team, and the insights are used to refine AI prompts and feature development.

Make user metrics visible

A team's focus follows what it measures. If dashboards only show metrics like velocity and deployment frequency, the user is easily forgotten.

To shift to a user-centric focus, display user experience and engineering metrics prominently in team meetings, and directly in developer tools. Focus on success and satisfaction metrics, for example measuring customer satisfaction (CSAT), task completion rates for features, or time-on-task for critical user journeys. Consider product success metrics like those in the H.E.A.R.T. framework.[3]

By making the "why" and "for who" of the work visible alongside the technical "what" and "how," teams and the AI tools they use are grounded in the impact they are having on the user.

Involve engineering directly in user research

While product managers, user experience researchers, and other team members have a vital role to play in synthesizing and exposing user needs, it's important that these roles serve to facilitate connection, rather than enforcing distance, between engineers and users. Invite developers to observe user testing sessions and usability studies directly. Distilled user research findings can strip away nuance. Seeing a user struggle firsthand to complete a task creates a lasting connection to the problem. Direct exposure builds stronger user empathy and provides a better understanding of user pain points.

This firsthand experience informs how they design solutions, write tests, and engage with AI, ensuring it serves the user's needs.

Consider spec-driven development

An emerging development paradigm aims to provide a structure around which large language models (LLMs) may be oriented toward user needs: spec-driven development (SDD).[4] SDD breaks coding into phases, generating versionable outputs. In the initial phase, developers, possibly with AI, refine user needs and constraints into documentation, the "source of truth."

Only after locking in specifications (which, optionally, may be subjected to peer review), is AI tasked with writing actual code. By elevating specs to a position of primacy, SDD encourages developers to actively assert user needs as constraints. This promises to reduce the likelihood that AI-generated code will be misaligned to user value.

For a deep dive into spec-driven development, see Betsalel (Saul) Williamson's presentation to the DORA Community: Unlock LLM Potential with Spec-driven Development.[5]

Common obstacles to a user-centric focus

As teams adopt AI, several common obstacles can weaken the essential connection to the end user, undermining the value of AI-accelerated work:

The feature factory[6] mindset
Teams become focused on measuring output (features shipped, velocity) rather than outcomes (user value).

AI can amplify this dysfunction, leading teams to rapidly build features that don't solve real user problems, resulting in high activity but low impact.

A technology-centric approach
Teams fall into the trap of "solutionism" or "resume-driven development," adopting new technologies (including AI models) for their own sake, not to solve a specific user problem. This adds complexity and pulls focus from the user's actual needs.

Organizational and procedural silos
Developers are often systematically disconnected from their end users by policies, processes, or team structures. This creates a "gatekeeper" model where user feedback is filtered, robbing developers of the deep context and empathy required to build valuable solutions.

How to measure user-centric focus

DORA survey questions

The ultimate measure of user-centric focus is in the eye of the user, reflected in product metrics like adoption, retention, or customer satisfaction.

A team with a healthy culture, attentive to visible user metrics, will prioritize that end result. Beyond such extrinsic measures, you can also evaluate a team's attitudes and practices, according to the same rubric used in the 2025 DORA survey.

DORA research measured this capability based on responses to these questions:[7]

- Creating value for our users is our focus.
- The experience of our users is our top priority.
- We believe that focusing on the user is key to the success of the business.
- We have a clear understanding of what our users want to accomplish with our application or service.
- We leverage user feedback to continuously revisit and reprioritize features.

Additional measures

Additional signals that may help measure the user-centricity of a team include:

Factor to test	What to measure
Product success	Product metrics like: • Adoption rate • Retention rate • Customer satisfaction
Team alignment	Inclusion of the user or user's goals in objectives like project milestones
User representation in artifacts	Design specifications that prominently include the user
User focus in ad-hoc design	How often a user is depicted in whiteboard or design drawings
Team values and recognition	User-centric accomplishments that receive the most vociferous praise from peers and leadership

Conclusion

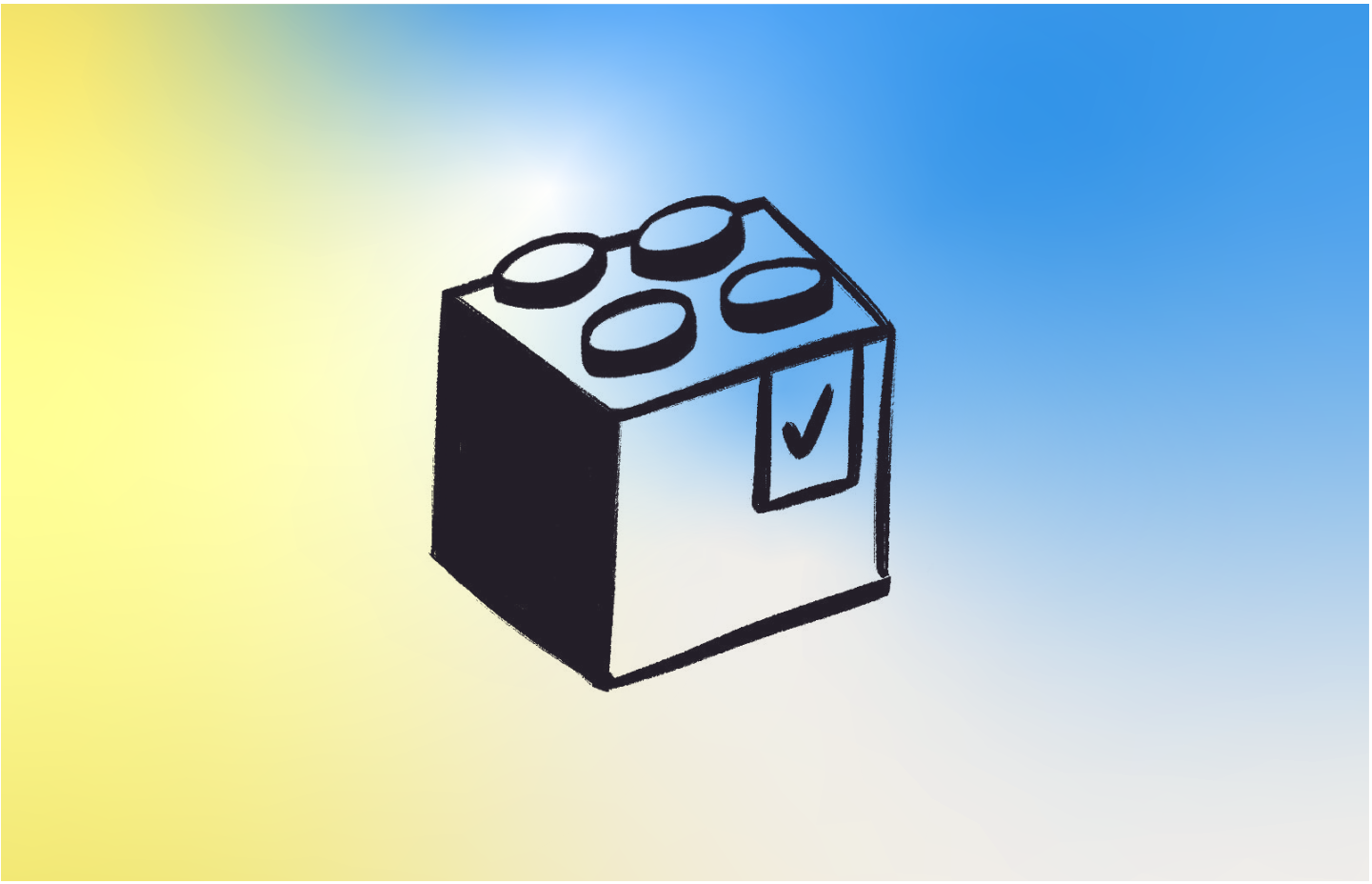
Gen AI tools are valued for their ability to process enormous amounts of heterogeneous data and quickly produce a single coherent output. They do this by making predictions based on context.

The developer is tasked with shaping this context, steering AI tools toward preferred outcomes.

Powerful tools like AI and attention to practices like continuous delivery[8] enable teams to produce high-quality software, quickly. But only teams with a user-centric focus will make software worth making.

Once teams are aligned in this user-focused direction, they need a scalable, secure, and automated way to deliver that value, which is the role of a quality internal platform.

1. 2024 Accelerate State of DevOps. 59–60. https://dora.dev/dora-report-2024
2. Accelerate State of DevOps Report 2023. 17. https://dora.dev/dora-report-2023
3. "Unlocking product success by combining DORA and H.E.A.R.T." https://cloud.google.com/transform/unlocking-product-success-by-combining-dora-and-heart
4. "Understanding spec-driven-development: Kiro, spec-kit, and Tessl." https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html
5. "DORA Community Demo & Discussion: Unlock LLM potential with spec-driven development." https://www.youtube.com/watch?v=9Goq80lgxSY
6. "12 signs you're working in a feature factory." https://medium.com/@johnpcutler/12-signs-youre-working-in-a-feature-factory-44a5b938d6a2
7. "Survey questions: User centricity." https://dora.dev/ai/capabilities-model/questions/#user-centric-focus
8. "Continuous delivery." https://dora.dev/capabilities/continuous-delivery



Quality internal platforms

Eric Maxwell

10x Technology Lead, Google Cloud

Quality internal platforms amplify AI adoption's positive influence on organizational performance.

The central theme of our 2025 research is that AI is an amplifier. It magnifies your organization's existing strengths — but also its dysfunctions. While many organizations are racing to equip individual developers with AI tools, our data reveals a crucial gap: these individual productivity boosts are often lost to downstream disorder, swallowed by bottlenecks in testing, security reviews, and complex deployment processes.

This is where platform engineering becomes the critical enabler. Our research found that 90% of organizations have adopted internal platforms and 76% now have dedicated platform teams to manage them, making platforms an essential foundation for modern software delivery.

A high-quality internal platform provides the automated, standardized, and secure pathways necessary to turn AI's potential into systemic, organizational improvements. It is the foundational capability that allows you to harness the speed of AI without sacrificing stability, amplifying its positive impact on organizational performance.

What high-quality internal platforms are

A quality internal platform is the set of shared, high-quality tools, services, documentation, and "paved roads" (or "golden paths") that make it easy for development teams to build, test, and deploy their applications securely, reliably, and in a compliant way.

This platform is best understood as an internal product designed for your developers. It abstracts away underlying complexity, allowing teams to focus on delivering user value rather than navigating infrastructure, security, and operational hurdles.

Our research shows that a platform's success is not just about its technical features, but about the holistic developer experience (DevEx) it provides.

Why internal platforms matter for AI

An internal platform acts as the essential distribution and governance layer for AI.

Our 2025 research found that the positive effect of AI adoption on organizational performance depends on the quality of the internal platform.

When platform quality is low, the effect of AI adoption on organizational performance is negligible. But when platform quality is high, that effect becomes strong and positive.

A high-quality platform provides the standardized, automated pathways needed to manage the increased volume and velocity of AI-assisted development. It ensures that AI-generated code is consistently tested, secured, and deployed, turning individual productivity gains into systemic, organizational improvements. It also serves as a critical risk mitigator, creating a safe space for experimentation by making failure cheap and recovery fast.

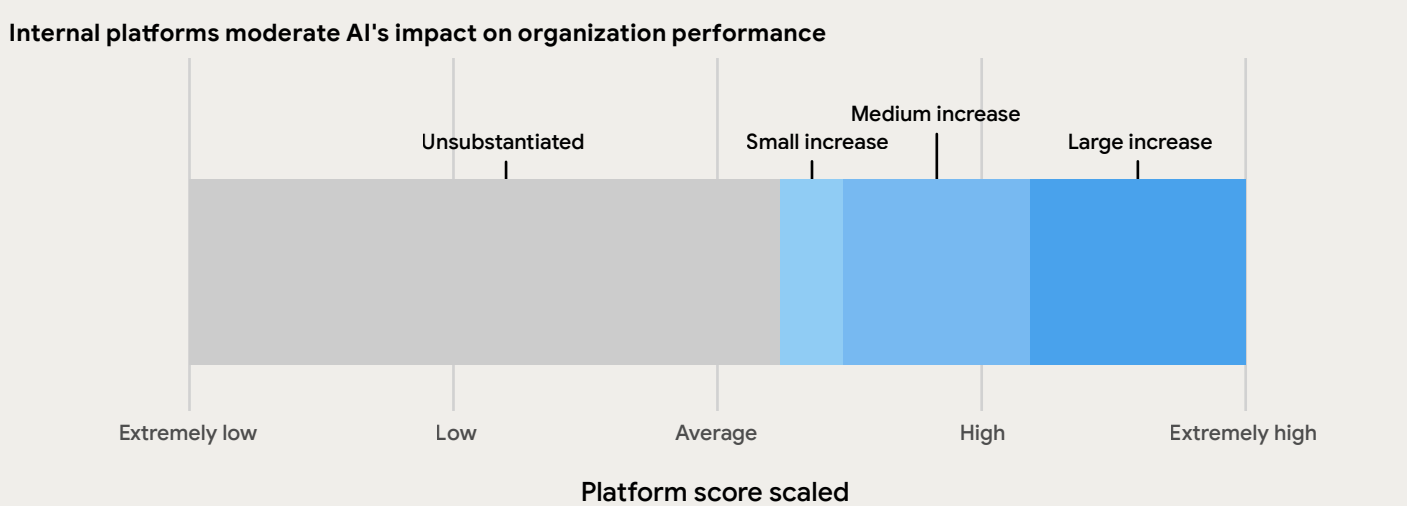


Figure 17: Internal platforms moderate AI's impact on organization performance

How to improve your internal platform

Our 2025 research shows that a platform's value is tied to its overall holistic experience, not just its individual features. Developers perceive the platform as a single entity, and their entire journey colors their perception of its success.

Improving your platform means moving beyond just technology and adopting a product-centric and user-centric approach.

Adopt a product management mindset

A platform is an internal product, and your developers are your customers. This requires assigning a product manager to focus on the DevEx, deeply understanding developer needs, and creating a strategic roadmap. This work should be driven by mapping out critical user journeys (for example, "spinning up a new service" or "debugging a production issue") to identify and eliminate the most significant points of friction.

Proactively "shift down" cognitive load

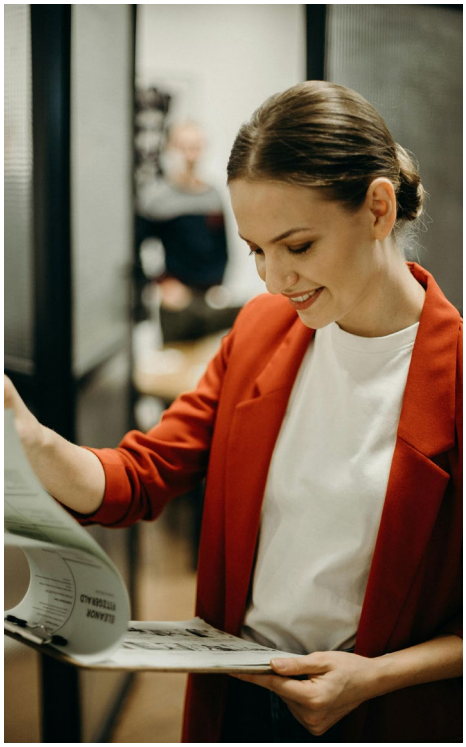
A platform's primary goal is to reduce the cognitive load on developers by abstracting away underlying complexity. Instead of forcing developers to become experts in Kubernetes, cloud networking, and security policies, you "shift down" this complexity into the platform. By providing simple, self-service workflows and golden paths, the platform enables developers to focus on what they do best: building features and delivering value to users. This concept is a key tenet of a successful platform engineering strategy (for more, see "How Google does it: Your guide to platform engineering"[1] and Google Cloud's platform engineering solution).[2]

Start with a "minimum viable platform"

Instead of trying to build a comprehensive platform all at once, focus on solving one high-value problem for a specific set of users. Identify the golden path for the most common workflow and build just enough to make that journey demonstrably better. This allows you to deliver value quickly, get crucial feedback, and build momentum for future iteration.

Design for extensibility and contribution

Your central platform team can't—and shouldn't—build everything. A successful platform is extensible, allowing other teams to contribute their own tools and services in a standardized way. Build your platform with clear APIs, good documentation, and a well-defined contribution model. This "open" approach scales the platform's value, leverages domain expertise from other teams, and prevents the platform team from becoming a bottleneck.



Prioritize clear feedback and observability

How do you know your platform is working well? Our 2025 data showed that the platform capability most correlated with a positive user experience is "gives me clear feedback on the outcome of my tasks." Developers need to easily understand what is happening when they use the platform, especially when something fails. Provide clear, actionable feedback, logs, and diagnostics to empower developers to self-serve and troubleshoot independently.

Secure ongoing investment by demonstrating value

A platform is not a "one-and-done" project; it is a living product that requires sustained investment to evolve with developer and business needs. Use the measurement strategies in the next section (like DORA metrics, H.E.A.R.T., and platform scorecards) to connect your platform's improvements directly to developer productivity and, ultimately, to organizational performance. This creates a virtuous cycle of investment and improvement.

Common obstacles and antipatterns

As organizations invest in platform engineering, several common pitfalls can undermine the platform's value, frustrate developers, and prevent the organization from realizing the benefits of its investment. Recognizing these antipatterns is the first step to avoiding them.

The "build it and they will come" antipattern

This is a primary symptom of treating the platform as a technical project, not a product. A team builds a platform based on what they think developers need, without doing any user research, interviews, or validation. They focus entirely on the technology and engineering, assuming its value will be self-evident. This platform ends up being a ghost town because it either doesn't solve real, painful problems for developers, or it doesn't fit their existing workflows.

The "ivory tower" platform

This obstacle arises when a central platform team dictates architecture and tools from on high, enforcing rigid standards without collaboration or a feedback loop. They act as gatekeepers of technology rather than enablers of developers. This approach leaves developers feeling disempowered and often creates "shadow IT" or unofficial workarounds to bypass the platform's constraints, defeating its purpose.

The “ticket-ops trap”

Here, the platform team operates like a vending machine for infrastructure, rather than an enabler of self-service. Their work is entirely reactive and driven by an endless queue of tickets from developers (for example, “provision me a database,” “set up a CI/CD pipeline”).

This creates a bottleneck and adds work for both the platform team and the developers. The team spends all its time on one-off requests and never has the capacity to build the cohesive, self-service capabilities that provide the platform’s real value.

The “big bang” approach

Some organizations attempt to build a comprehensive, all-encompassing platform that solves every conceivable problem before releasing any part of it. This strategy is high-risk, expensive, and delays valuable feedback. By the time the “perfect” platform is finally launched, developer needs have often changed. A product-centric approach, in contrast, delivers value incrementally and iterates based on real-world use.

A “one-size-fits-all” mentality

In an attempt to maximize standardization, platform teams may create a single, rigid “golden cage” that doesn’t account for the diverse needs of different development teams. The needs of a data science team, for example, are very different from those of a front-end mobile team.

A successful platform provides enabling constraints and golden paths while still offering the flexibility for teams to use the right tools for their specific job.

Failing to secure executive sponsorship

A platform initiative is a significant organizational and technical investment that requires bridging existing silos. Without clear, visible, and long-term executive sponsorship, the platform team can be starved of resources, struggle to get buy-in from other departments, and lack the authority to drive the necessary cross-functional changes.

How to measure an internal platform's quality

DORA survey questions

DORA research measured this capability by asking these questions:[3]

- In the last three months, did you use a platform at work?
- To what extent does your platform demonstrate the following characteristics?
  - The platform behaves in a way I would expect.
  - The platform effectively abstracts away the complexity of underlying infrastructure.
  - The platform gives me clear feedback on the outcome of my tasks.
  - The platform helps me build and run reliable applications and services.
  - The platform helps me build and run secure applications and services.
  - The platform helps me follow required processes (for example, code reviews, security sign-offs).
  - The platform is easy to use.
  - The platform provides the tools and information I need to work independently.
  - The platform team acts on the feedback I provide.
  - The platform's user interface is straightforward and clean.
  - The tasks I perform on the platform are well-automated.
- Does your organization have a dedicated platform team?

H.E.A.R.T. framework measures

The H.E.A.R.T. framework provides additional signals that may help measure the platform engineering practice.

Factor to test	What to measure
Happiness (**H**.E.A.R.T.)	Developer sentiment and satisfaction with the platform
Engagement (H.**E**.A.R.T.)	How often and how deeply developers use platform features
Adoption (H.E.**A**.R.T.)	The rate of new teams and services onboarding to the platform
Retention (H.E.A.**R**.T.)	The rate at which teams continue to use the platform
Task success (H.E.A.R.**T**.)	The efficiency and effectiveness of developers completing key workflows

Additional measures

Additionally, you may want to measure the impact of your platform engineering efforts through software delivery performance metrics and developer satisfaction.

Use DORA's software delivery performance metrics (internally and externally)

DORA's five software delivery performance metrics (lead time for changes, deployment frequency, failed deployment recovery time, change failure rate, and deployment rework rate) benefit platform and application teams.

1. **For the platform team:** Your platform team should use these metrics to measure its own performance, helping you improve the speed and stability of changes to the platform.
2. **For the application teams:** The platform should provide these metrics to its users. By automatically instrumenting and exposing these metrics for every service, you empower developers with the high-level insights they need to understand their own delivery performance and see the direct impact of the platform on their work.

Track developer satisfaction

Regularly survey developers (the platform's customers) on their satisfaction with the platform. Use simple measures like customer satisfaction (CSAT) or net promoter score (NPS), combined with qualitative feedback, to track sentiment over time and identify areas for improvement.

Conclusion

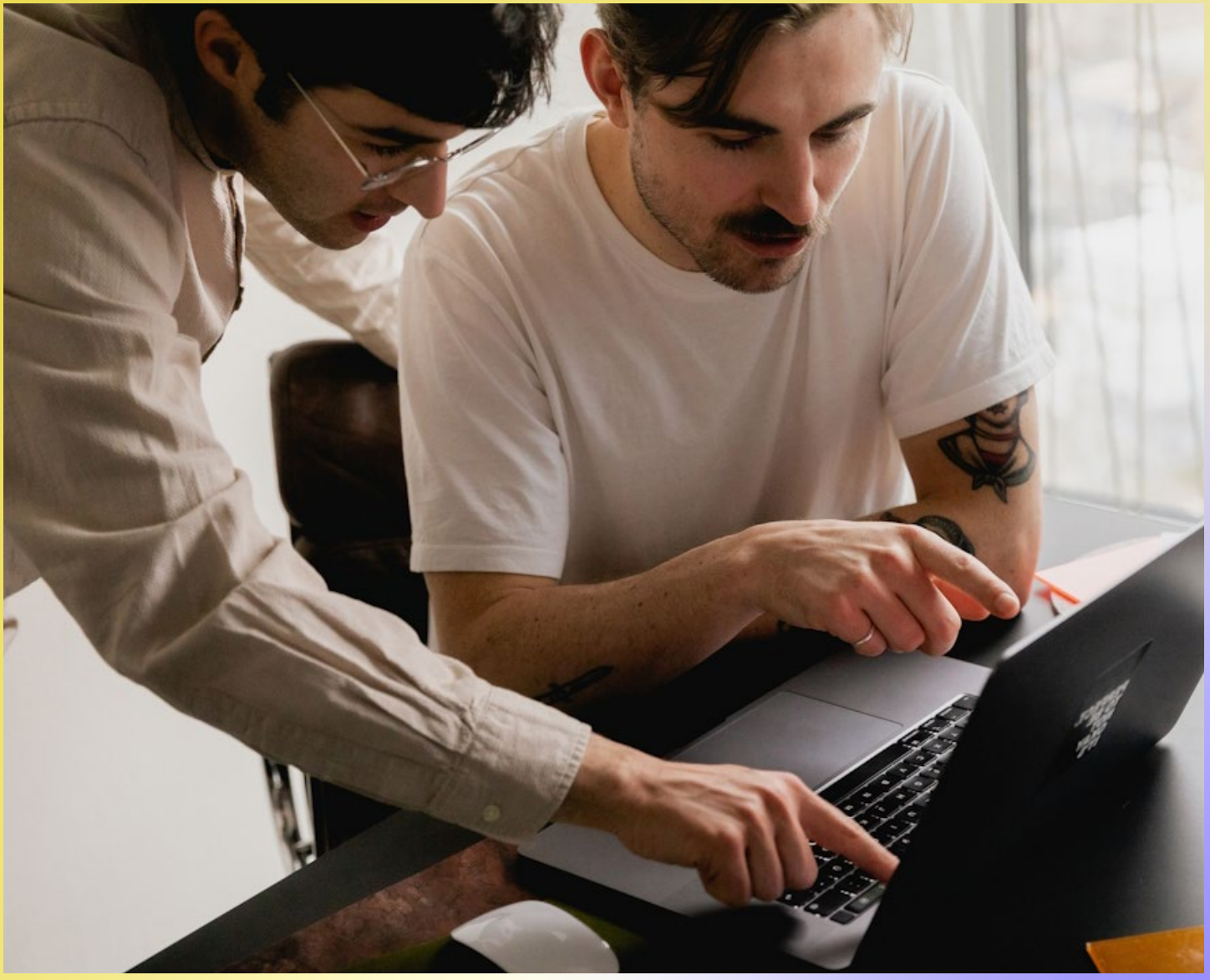
A quality internal platform is the ultimate amplifier in the DORA AI Capabilities Model. It acts as the "paved road" that scales the benefits of all other capabilities—from AI-assisted workflows to user-focused features—securely and reliably across the organization.

By abstracting complexity and automating delivery, the platform ensures that individual productivity gains from AI are not lost to downstream disorder. It is the foundational system that translates the potential of AI into measurable, systemic, and sustainable organizational performance.

1. "How Google does it: Your guide to platform engineering." https://cloud.google.com/blog/products/application-modernization/a-guide-to-platform-engineering
2. "Shift down with platform engineering on Google Cloud." https://cloud.google.com/solutions/platform-engineering
3. "Survey questions: Platform engineering." https://dora.dev/ai/capabilities-model/questions/#quality-internal-platform

Assessing and prioritizing your AI capabilities

Nathen Harvey
DORA Lead, Google Cloud



Identifying your team profile

Using the data collected from nearly 5,000 technology professionals from around the world, we conducted a cluster analysis to understand the human and systemic factors that drive performance. We identified common patterns, team archetypes, and team profiles. Our statistical clustering approach included the following factors:

Team performance

This factor measures the perceived effectiveness and collaborative strength of an individual's immediate team.

Product performance

This factor measures the success and quality of the products or services the team is building based on characteristics like helping users accomplish important tasks and keeping information safe, and performance metrics such as latency.

Software delivery throughput

This represents the speed and efficiency of the software delivery process.

Software delivery instability

This captures the quality and reliability of the software delivery process.

Individual effectiveness

This factor captures an individual's self-assessed effectiveness and sense of accomplishment at work.

Valuable work

This measures the self-assessed amount of time an individual spends doing work they feel is valuable and worthwhile.

Friction

This measures the extent to which friction hinders an individual's work. Lower amounts of friction are generally considered to be a positive outcome.

Burnout

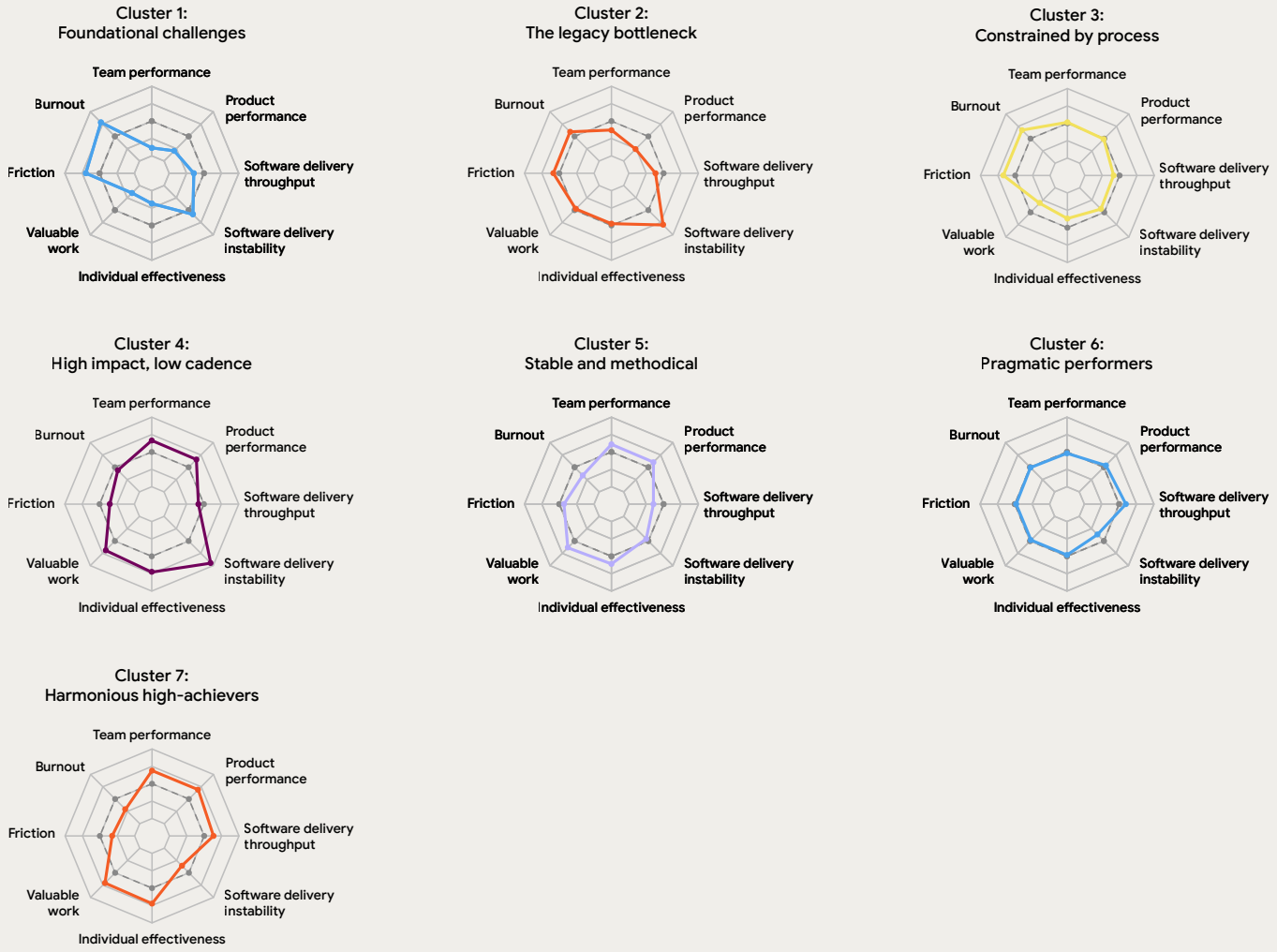
This measures feelings of exhaustion and cynicism related to one's work. Lower amounts of burnout are generally considered to be a positive outcome.

Organizations, teams, and individuals usually strive to increase team performance, product performance, software delivery throughput, individual effectiveness, and valuable work. They aim to reduce software delivery instability, friction, and burnout.

Our analysis revealed seven distinct team archetypes, ranging from those excelling in healthy, sustainable environments (harmonious high-achievers) to those trapped by technical debt (legacy bottleneck) or inefficient processes (constrained by process).

Figure 18 shows the performance levels of the seven team archetypes. The dotted line represents the average of all respondents, while the colored line shows the performance of each cluster. Remember that teams typically strive to increase their score for some factors, such as individual effectiveness and software delivery throughput, while reducing the score for other factors, such as burnout and friction.

Performance levels of seven team archetypes



The names and descriptions for each of these clusters are an interpretation of the data. Your team may see similar performance levels as a given cluster but may not feel the cluster name or description describe your team well.

Figure 18: Performance levels of seven team archetypes

Cluster 1: Foundational challenges

These teams are stuck in survival mode, facing significant challenges with fundamental gaps in their processes, environment, and outcomes.

Percentage of respondents:
10% of survey respondents are in cluster 1.

Performance indicators:
Team output, product delivery, and value creation are consistently low.

Team well-being:
The data shows high reported levels of burnout and significant friction.

System stability:
There are notable challenges with the stability of the software and operational environment.

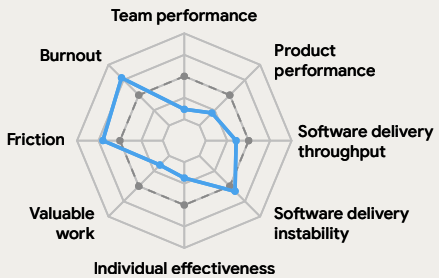


Figure 19: Cluster 1: Foundational challenges

Cluster 2: The legacy bottleneck

Teams in this cluster are in a constant state of reaction, where unstable systems dictate their work and undermine their morale.

Percentage of respondents:
11% of survey respondents are in cluster 2.

Performance indicators:
Product performance metrics are low. While the team delivers regular updates, the value realized is diminished by ongoing quality issues.

Team well-being:
The data indicates a demanding work environment. Team members report elevated levels of friction and burnout.

System stability:
There are significant and frequent challenges with the stability of the software and its operational environment, leading to a high volume of unplanned, reactive work.

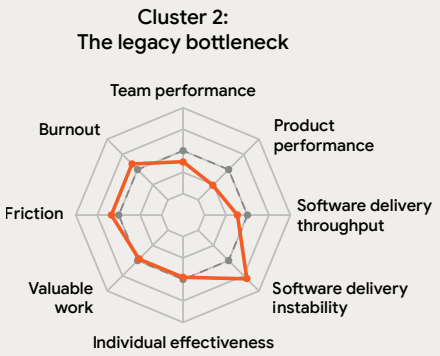


Figure 20: Cluster 2: The legacy bottleneck

Cluster 3: Constrained by process

These teams are running on a treadmill. Despite working on stable systems, their effort is consumed by inefficient processes, leading to high burnout and low impact.

Percentage of respondents:
17% of survey respondents are in cluster 3.

Performance indicators:
These teams experience low effectiveness and the creation of limited customer or business value.

Team well-being:
The data shows high reported levels of both burnout and friction. This suggests that current workflows and processes are creating a challenging and unsustainable work environment for the team.

System stability:
The team's software and operational environments are stable and reliable. This indicates that technical instability is not a primary contributor to the challenges in performance and well-being.

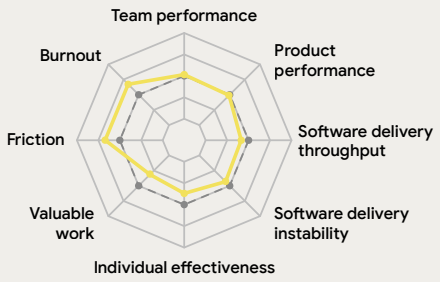


Figure 21: Cluster 3: Constrained by process

Cluster 4: High impact, low cadence

These teams produce high-impact work, reflected in strong product performance and high individual effectiveness. However, this is coupled with a low-cadence delivery model characterized by low software delivery throughput and high instability.

Percentage of respondents:
7% of survey respondents are in cluster 4.

Performance indicators:
The team consistently achieves top-tier levels of productivity. Both effectiveness and product performance metrics are strong.

Team well-being:
The data indicates a low-friction environment, suggesting that team processes are efficient and collaborative.

System stability:
The operational environment is characterized by a high degree of instability. This level of volatility represents a significant risk to service reliability and long-term sustainability.

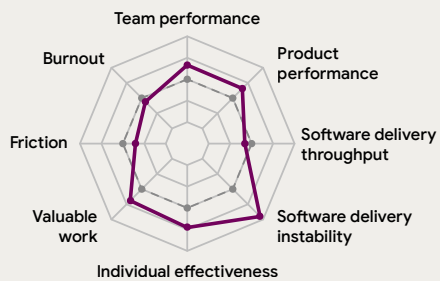


Figure 22: Cluster 4: High impact, low cadence

Cluster 5: Stable and methodical

These teams are the steady artisans of the software world, delivering high-quality, valuable work at a deliberate and sustainable pace.

Percentage of respondents:
15% of survey respondents are in cluster 5.

Performance indicators:
Metrics for product quality and value creation are consistently positive. However, the team's software delivery throughput is in a lower percentile, indicating a more deliberate pace of work.

Team well-being:
The data shows low reported levels of burnout and friction, which points to a healthy and sustainable team environment.

System stability:
The team's software and operational environments are characterized by high stability and reliability.

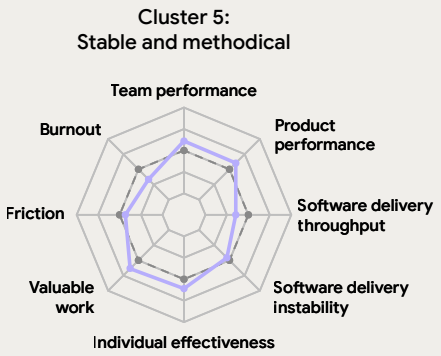


Figure 23: Cluster 5: Stable and methodical

Cluster 6: Pragmatic performers

These teams consistently deliver work with impressive speed and stability, even if their work environment hasn't reached a state of peak engagement.

Percentage of respondents:
20% of survey respondents are in cluster 6.

Performance indicators:
Software delivery performance is strong, with better-than-average throughput and low instability. The team maintains a steady cadence of valuable output, reliably meeting expectations.

Team well-being:
The data shows average levels of reported burnout and friction. This indicates a work environment that is functional and sustainable but may lack strong engagement drivers.

System stability:
The team's software and operational environments are stable and reliable, providing the solid foundation required for their high performance.

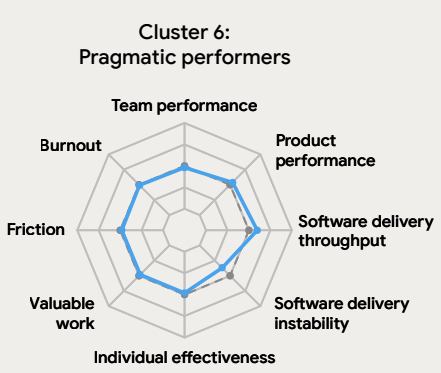


Figure 24: Cluster 6: Pragmatic performers

Cluster 7: Harmonious high-achievers

This is what excellence looks like—a virtuous cycle where a stable, low-friction environment empowers teams to deliver high-quality work sustainably and without burnout.

Percentage of respondents:
20% of survey respondents are in cluster 7.

Performance indicators:
The team shows positive metrics across multiple areas, including team well-being, product outcomes, and software delivery.

Team well-being:
The work environment is characterized by low reported levels of burnout and friction.

System stability:
The team operates on a stable technical foundation that supports both the speed and quality of their work.

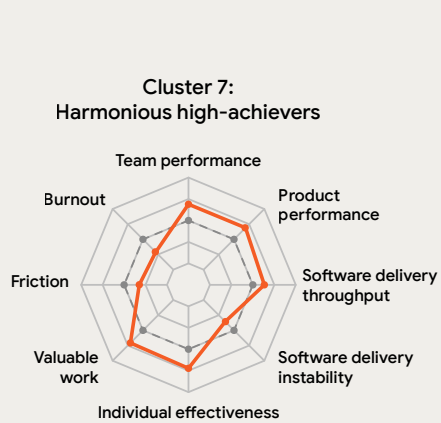


Figure 25: Cluster 7: Harmonious high-achievers

Where is your team today?

Improving performance starts with understanding the current conditions for a team.

However, a precise answer on which cluster best describes the team today isn't critical. It's more important to identify the area that would benefit the most from improvement efforts.

There are a few different ways to assess a team's current performance to determine which cluster best describes how you are performing today.

Conversation

Gather the cross-functional team that is responsible for a single application or service and have a conversation about how the team is performing. The team can use the seven clusters as a guide and agree on which one best describes how things are going.

Here's one way to structure the conversation.

Review the team profiles:

Review this section's content covering the seven team profiles.

1. Build consensus:
   - List the seven team profiles on a whiteboard or screen.
   - Give each participant three slips of paper for silent, anonymous voting.
   - Vote: Each participant must cast votes for the profile they feel best matches the current conditions of the team. Each participant must cast three votes but can distribute them as they see fit. For example, if a participant feels very strongly that the team maps best to cluster six, that participant would cast all three votes for that profile.
   - Tally the results.
2. Reveal the votes and discuss:

   Reveal the results to the team and take note of where the votes clustered. Have a brief, timeboxed discussion. The goal isn't to find one "perfect" label but to build a general consensus on the team's current challenges. Some helpful questions to ask:
   - Looking at where our votes clustered, what's the one or two challenges from that profile that feel the most real in your day-to-day work?
   - Did you expect a different profile to get more votes, and if so, why?
   - What are the common themes or frustrations that less popular profiles share with our team?

Assessment

Run a survey that collects individual responses for each of the eight factors used in our cluster analysis. Doing so will give you insights into the current experience of the team. Using this information you can map your team to one of the seven clusters.

The questions used in the 2025 DORA survey are available on the DORA website.[1]

Understanding how a team is performing can help uncover which areas would benefit most from improvement, and subsequently inform AI capabilities to prioritize.

For example, teams facing "foundational challenges" might want to focus their improvement efforts on team performance, supported by AI capabilities that enhance collaboration. High impact, low cadence teams might want to focus on reducing software delivery instability, perhaps by using AI for automated testing and analysis.

1. "Survey questions." https://dora.dev/research/2025/team-profiles/questions

Digging deeper with value stream mapping

Rob Edwards

Application Delivery Lead, Google Cloud

Dave Stanke

Specialist Customer Engineer, Google Cloud

As DORA research has repeatedly demonstrated, organizational performance depends on the rapid delivery of high-quality software to end users. But how does that software really get made? Software engineering is a team activity, with many human participants and automated processes, each with a unique perspective.

To improve the software delivery flow, we first need a shared, high-level understanding of its current state. Value stream mapping (VSM) offers a detailed, data-driven view of how work actually flows through your systems. VSM is the practice of visualizing and analyzing every step in your process—from idea to customer—to identify the real friction points, bottlenecks, and delays.

It's important to clarify a common point of confusion: the difference between value stream mapping and value stream management.

Value stream mapping: This is the tactical exercise we describe in this chapter, a tool for visualizing and analyzing a specific workflow, often on a whiteboard, to build a shared understanding.

Value stream management: This is the strategic, holistic discipline of continuously managing and improving the entire end-to-end flow of value, from initial idea to customer, across interconnected teams and systems.

Value stream management as a discipline is a vast topic. For this chapter, we are intentionally focusing on the mapping exercise and using the software delivery process (from "commit-to-prod," in line with DORA software delivery performance measures) as a practical starting point to get teams thinking about their flow.

Why VSM is critical for prioritization

Metrics and experience can tell you what you might be experiencing (for example, "Change lead time is becoming longer"), but VSM shows you where and why it's happening (for example, "Our code review process has a three-day average wait time and our security scan takes eight hours to run").

In an era of rapid AI adoption, the greatest risk is pouring massive investment into chaotic activity that doesn't move the needle. DORA research shows that AI acts as an amplifier of positive and negative behaviors and outcomes, so it's essential to identify and address dysfunction in the flow of value. VSM is what separates disorganized activity from focused improvement, allowing you to target the most impactful capability.

VSM: An AI force multiplier

DORA research has validated that VSM is a force multiplier that turns AI investment into a true competitive advantage. The findings are clear: AI adoption alone has only a modest impact on organizational performance. However, this effect is increasingly amplified when paired with strong VSM practices.

VSM moderates AI's impact on organizational performance

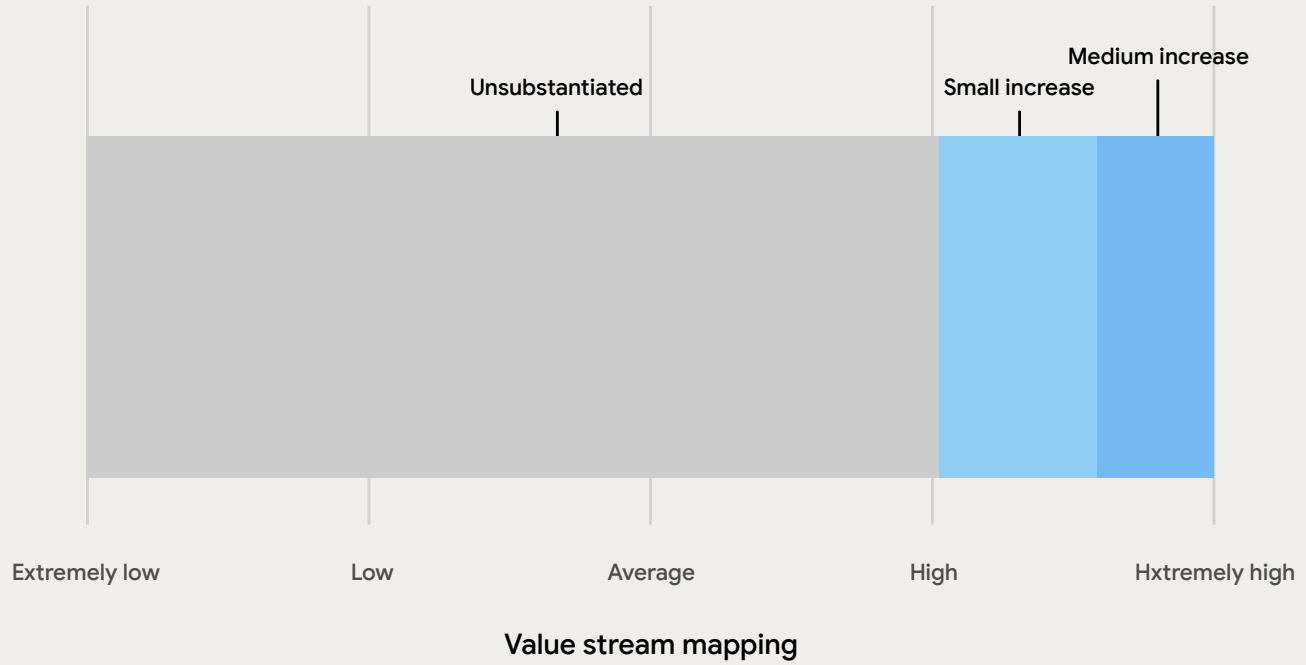


Figure 26: VSM moderates AI's impact on organizational performance

Successful AI adoption requires more than adopting new tools; it's a transformation of systems and processes. VSM ensures that the efficiency gains from AI are channeled toward solving system-level constraints. Without VSM, AI risks creating local pockets of productivity that are simply lost to downstream chaos and bottlenecks.

For example, if your VSM exercise reveals that code review is a major bottleneck, VSM guides you to investigate where to improve, including experiments to apply AI to the review process. This is far more effective than using AI to simply generate more code, which would only pile up and worsen the bottleneck.

Teams that practice VSM see higher team performance, and spend significantly more of their time on valuable work and achieve better product outcomes.

Core VSM principles

To achieve this focused improvement we have five core principles.

Prioritize the act of mapping over the artifact:

VSM is more than just a map. The most valuable outcome of VSM is the collaborative process of creating it. The goal is to engage all participants in the value stream and solicit their diverse perspectives. The shared truths uncovered and the cross-functional commitment built during this act of mapping are what truly drive improvement.

Move from mental mess to shared map:

VSM is an exercise in getting the process details out of everyone's head and into a shared space, like a whiteboard. This shared map allows the team to develop a collective understanding of the workflow and easily spot hidden bottlenecks and inefficiencies. A key test is to ask: "Can you draw your software delivery value stream on a whiteboard?"

Focus on flow, not just speed:

The primary objective is to make work flow smoothly and predictably. This requires a shift from optimizing isolated steps to optimizing the overall system. To do this, you must measure key metrics across each transition between steps.

Create a culture of continuous improvement:

Because VSM is an ongoing cycle, teams should revisit their value stream map regularly. Teams should be empowered to experiment, learn, and adapt without fear of reprisals, and then share those lessons across the organization.

Build on a foundation of technical excellence:

A fast, smooth flow is impossible without technical excellence, which is typically one of the things provided by a well-designed internal platform. This platform should offer developers "paved roads" for capabilities like testing and delivery, which abstracts complexity and makes high-performance work scalable.

Running a lightweight VSM exercise

Here is a simple way to get started:

Define outcomes
Before mapping, the team must clearly define the business or customer outcomes they are trying to improve. This ensures the VSM effort is focused on solving a real business problem, not just optimizing a process.

This step answers, "Why are we doing this?" (for example, "reduce our lead time") and "What does 'better' look like?" (for example, "from seven days to two days").[1]

Gather the team
Get everyone who contributes to the product in a room (virtual or physical). The cross-functional team involved will probably consist of separate discrete organizational "teams"; it is important that all the people involved in the area you are focusing on are involved in the exercise.

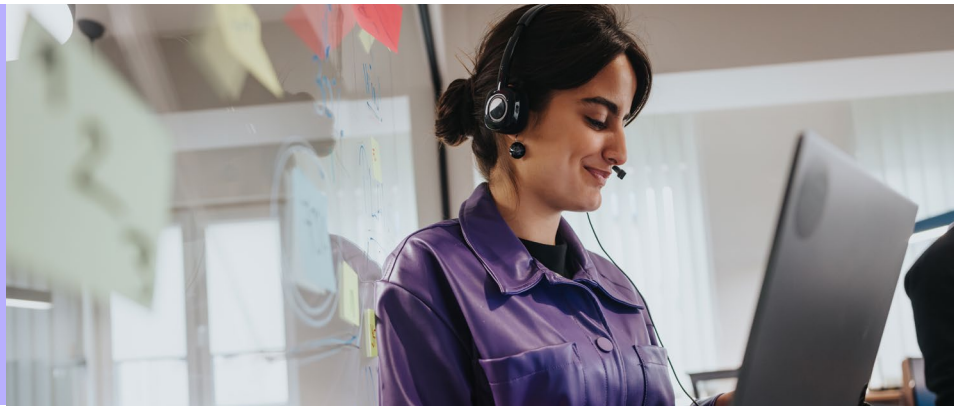
Map the flow
On a whiteboard, map the high-level steps.

TIP:

A value stream doesn't stop at your team's boundaries. To see the real flow, you must include representatives from key upstream (such as Product and Design) and downstream (such as Security, Ops, and Legal) teams. The most painful bottlenecks and longest wait states are almost always found in the handoffs between teams.

TIP:

For a fresh perspective, tackle your task in reverse. By starting at the end and working your way back to the start, you disrupt your normal thought patterns, which often reveals new insights.



It's crucial to map both the "happy" path (the ideal flow) and the process for recovering from an incident (the "recovery value stream"), as this often reveals hidden friction and delays. While the full map goes from idea to customer, a powerful starting point is often the scope from code commit to running in production, as teams have the most agency here. Once the current-state flow is mapped, the team must analyze it to identify constraints and gather data:

Identify wait states:
For each step, identify where work is waiting for someone or something else. These are your bottlenecks.

Hypothesize causes:
Discuss the potential reasons for the delays at each wait state. This is where you'll connect the bottleneck to a specific capability gap.

Gather the metrics:
For each step, estimate or measure the key metrics:

- Process time: The time it takes to actively work on a task.
- Wait time: The time a task spends in a queue waiting for the next step.
- Percent complete & accurate (%C&A): The percentage of work that the downstream step can use without needing corrections. For example, if six out of 10 pull requests sent to QA are accepted without rework, the %C&A is 60%.

When you first run this exercise on a whiteboard, your metrics will likely be estimates (for example, "Code review usually takes roughly two days"). This is a valuable starting point for building a shared understanding.

While modern VSM can eventually pull real, live data from your toolchains, be careful not to make metric precision the main objective. The true value lies in using these numbers—whether they're rough estimates or precise observations—to have better conversations and find ways to improve the flow of work.

Figure 27 illustrates an example of a value stream map. The total active process time (PT) is only about one day (~24.5 hours), while the total wait time is nearly four days (~92.5 hours). This results in a total end-to-end lead time of roughly 117 hours and a flow efficiency of ~21%. This highlights that work is sitting in a wait state for nearly four-fifths of the time.

Map the future stats: After identifying constraints and gathering metrics on your current state, design an ideal "future state" map. This new map represents your goal, showing what the process should look like with bottlenecks and wait times removed. The gaps between your current and future states become your specific, actionable improvement goals.

VSM: The code review bottleneck

Process:	Commit Code is committed to repo	Pull request /Coding WT:24h	Code review	Deploy to QA env	CI build	Functional tests	Perf regression tests	Release review	Deploy to pre-prod	Deploy to prod
Responsible team:	Developer	Developer	Technical lead	QA Team	Perf regression tests	QA Team	Performance Eng	Change approval board (CAB)	Release eng	App ops
Metrics	PT: - WT: - %C&A: -	PT:12h WT: 24h %C&A: 80%	PT: 2h WT: 20h %C&A: 70%	PT: 0.5h WT: 1h %C&A: 95%	PT: 0.5h WT: 2h %C&A: 90%	PT: 4h WT: 16h %C&A: 85%	PT: 12h WT: 24h %C&A: 80%	PT: 1h WT: 4h %C&A: 100%	PT: 0.5h WT: 0.5h %C&A: 95%	PT: 1h WT: 1h %C&A: 99%

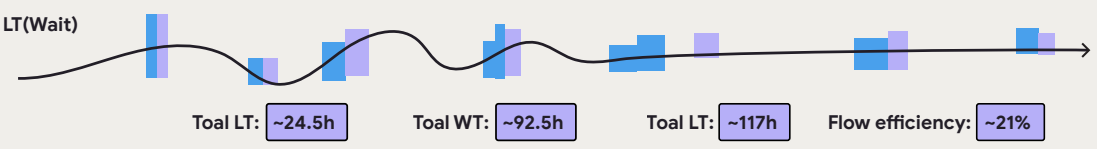


Figure 27: VSM: The code review bottleneck

Connecting VSM insights to AI capabilities

Use the findings from your VSM exercise to target specific AI capabilities for maximum impact.

If you find...	Prioritize...
Inconsistent code reviews	"AI-accessible internal data" to create context-aware code review agents and "strong version control practices" to ensure changes are small and easy to review
Developers spend a large amount of process time searching for information or understanding existing code	"AI-accessible internal data" to provide instant, context-aware answers
A high rate of rework and bugs discovered late in the process	"Comprehensive test automation" where AI can help generate, execute, and curate more thorough test cases
Code piling up waiting for review, making code review a bottleneck	Using AI to improve the review process itself, instead of using AI to generate more code that just piles up. Teams with faster code reviews have 50% higher software delivery performance.[2]

Example: Using VSM for platform engineering

The VSM framework is not limited to your application's delivery pipeline; it is an incredibly effective tool for platform engineering teams to improve their own internal products.

In this model, you treat your platform as a product and your developers as your customers. This shifts the entire perspective:

Your user: The internal developer or development team.

Your value stream: The critical developer journeys required to build, test, and deliver software using your platform.

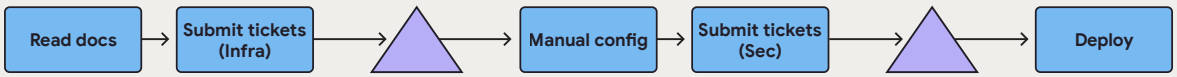
Your goal: To map, measure, and improve the developer experience, removing their bottlenecks, cognitive load, and wait times.

You can use the exact same lightweight VSM exercise outlined above, but you will map these developer journeys instead.

Key developer user journeys to map:

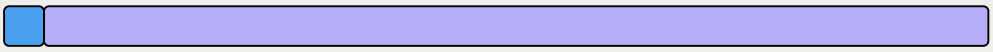
- The **golden path:** Scaffolding a brand new, production-ready service.
- The **inner loop:** A developer making a code change and seeing it running in a development environment.
- The **outer loop**: A developer merging a feature and getting it deployed to production.
- The **incident**: A developer being paged and trying to debug a production issue.
- The **onboarding**: A new developer's journey from getting their laptop to merging their first pull/merge request.

VSM: "New microservice" journey (current state)



Process time:	2h	2h	0	2h	10m	0	30m
Wait time:	0	0	**2 days**	0	0	**3 days**	0
%C&A:	50%	50%	100%	60%	95%	100%	90%

Lead time breakdown



Process time: 4.8 hours

Wait time: 5 days

Total lead time: 5 days, 6 hours and 40 minutes
Total process time: 6 hours 40 minutes
Flow efficiency: 5.3%

Figure 28: VSM of new service journey (current state)

Example VSM: The new microservice journey

Imagine mapping the golden path for scaffolding a new service.

1. **Map the flow (current state):**
   - **Discovery:** Developer navigates multiple wiki pages.
   - **Ticket:** Submits repository request to infrastructure. (Wait: Two days)
   - **Manual work:** Copies boilerplate code; attempts pipeline configuration.
   - **Ticket:** Submits security scan request. (Wait: Three days)
   - **Result:** A basic service is finally deployed.
2. **Gather the metrics:**
   - **Total lead time:** More than five days
   - **Developer toil:** Four to six hours of frustrating, low-value work.
   - **Risk profile:** High. Manual work leads to errors and configuration drift.
   - **Core friction:** Manual "ticket-ops," handoffs, and high cognitive load.
3. **Map the future state and connect to AI:**
   - **The future state:** A developer runs a single command and has a secure, production-ready service deployed in five minutes.
   - **Connecting to AI:** This analysis presents the clear business case: the future state is an AI-powered agent. Trained on your internal docs, security policies, and best practices, this agent is invoked by that single command. It orchestrates the entire journey—provisioning, configuration, and AI-powered code generation—turning a five-day bottleneck into a five-minute, paved road.

Figure 29: VSM of new service journey (future state)

By applying VSM to your internal developer journeys, you transform platform engineering from a cost center (managing tickets) into a value driver (accelerating all teams). You use the same VSM principles to find the real constraints on developer productivity and apply AI to solve them at a system level.

Conclusion

We believe that this research is more than observations on VSM; it's a challenge to improve team processes.

The first step to escaping the cycle of current activity may not be easy, but it is simple. Ask your team: "Can we draw our software delivery value stream on a whiteboard?"

If the answer is no, or if the drawing reveals more questions than answers, you have found your starting point. If the answer is yes, then you have a map and can immediately move to the analysis phase: measuring the process time, lead time, and %C&A for each step to find the hidden wait states. That single conversation is the beginning of getting better at getting better.

1. "How to use value stream mapping to improve software delivery: A guide to value stream mapping." https://dora.dev/guides/value-stream-management
2. Accelerate: State of DevOps Report 2023. https://dora.dev/dora-report-2023

Facilitating a team prioritization workshop

Cedric Yao

Solution Engineer, AI Application Innovation,
Google Cloud

An assessment is just data without an actionable plan to follow. Taking the first step to drive change based on assessments turns those insights into focused, collective action.

We've created the following 90-minute workshop to separate divergent and convergent thinking stages.

The goal is to bring the whole team to a shared understanding of your primary constraints, reach consensus on the highest impact improvements, and find agreement on a prioritized list of what to do next.

Before you begin: Prerequisites

This 90-minute workshop is for acting on data, not gathering it. You must have a few things ready to make the most of your time together.

A facilitator: This person guides the discussion, watches the clock, and stays neutral. It can be a team member, but it's often easier if they aren't directly invested in the outcomes. They can also ask clarifying questions in their role as an outsider to foster conversation when needed.

The whole team: The concept of software delivery as a team sport has echoed through the halls of the software industry for decades. Your team is the group of people you work with on a day-to-day basis to build, maintain, or support the same application or service. This workshop is designed to bring empathy to other parts of the team and requires representatives from all aspects of delivery. Some roles that are sometimes forgotten:

1. Product manager
2. QA
3. Executive sponsor
4. Architect
5. DevOps

The completed value stream map: You should have a value stream map that clearly identifies the top one or two bottlenecks or constraints in your current workflow. This workshop isn't for creating the value stream map, but for acting on it.

Workshop supplies: You'll need a large whiteboard (or a virtual equivalent like Miro), sticky notes, and markers. You'll also need a template for the impact/effort matrix, shown in step 3 below.

(Optional) Identified team profile: Have the identified team profile ready to display. See Identifying your team profile for additional guidance.

Executive sponsor's role in the workshop

While having an executive sponsor participate in the full 90-minute workshop can effectively signal buy-in, their presence can also be a blocker to open, honest conversation. The power dynamics in some cases hinder the psychological safety required for effective brainstorming and prioritization.

The executive sponsor should:

Kick off the workshop:
Join for the first 10 minutes to explain why the exercise is a priority and explicitly state their sponsorship for the improvements the team identifies.

Leave the room:
After the introduction, the sponsor should leave, allowing the team to proceed with the workshop's divergent and convergent thinking stages freely.

Rejoin for the final 15 minutes:
Return to the workshop as the team presents their "quick wins" and the prioritized "first step." The sponsor should reinforce their commitment and offer to help clear any potential blockers for the team's plan.

Workshop agenda (90 mins)

Step 1: Context setting and problem identification (10 mins)

The goal of this step is to ensure everyone is starting from the same, shared reality. The most effective workshops do this by democratizing sharing and decision-making. Use activities like anonymous voting and timeboxing to safeguard against strong voices biasing the conversations.

1. **(Optional) Review the team profile (5 mins):**
   Ask a team lead to present the identified team profile.
   - Review any points of relevant conversation that arose during the team identification activity. The goal is to provide empathy and shared context around the team's findings.
2. **Review value stream map findings (5 mins):**
   Present the team's value stream map; review any points that need clarity. Point directly to the one or two biggest friction points that were previously identified (for example, "wait time for code review" or "manual testing handoff"). State clearly: "For this workshop, we are going to focus on solving this specific bottleneck."

Step 2: Brainstorming initiatives (20–40 mins)

The next step of effective workshops is to generate as many ideas as possible. Your goal as a facilitator is to collect as many ideas as you can because sometimes the best ones come from the least expected places.

1. **Idea generation (5 mins each bottleneck):**
   For each of those selected bottlenecks state:
   "I want you to think of specific, actionable things we could do to reduce this bottleneck. Write each idea on a single sticky note."

- Encourage all ideas. This is about quantity, not quality, at this stage.
- Write each idea on its own sticky note.
- Example:
  - **Identified constraint:** Code review is a bottleneck.
  - **Brainstormed ideas:** "Implement a user story AI agent reviewer," "Implement a PR size checker," "Hold a story-slicing training session," "Set a team norm of 4-hour review turnarounds," "Try pairing on all complex changes."

2. **Review and group (10–15 mins each bottleneck):** Going around the room, ask one team member at a time to read one sticky (15 seconds). Ask the team if they have a similar idea. Try to group all similar ideas into a single sticky.

Step 3: Ruthlessly prioritize through impact/effort mapping (30 mins)

Chances are you have too many ideas with no consensus on where to start. In order to keep the team focused, you need to ruthlessly prioritize all the ideas and let the best rise to the top. You will build that shared understanding and alignment among the team with a convergent 2x2 prioritization activity.

1. **Set up the exercise:** Create a single axis and label as shown in Figure 30.

Effort axis

Note: The effort axis looks reversed, but this is on purpose as we want to prioritize low effort changes.
Figure 30: Effort axis

2. **Start somewhere:** Randomly pick an idea and place it in the direct center.
3. **Prioritize and discuss the effort for each idea:** Pick up one sticky note at a time and, as a team, decide where to position it relative to other stickies on the board. When discussing effort, sometimes it helps to think about this axis as complexity rather than estimates of time.

Effort axis with initial ideas

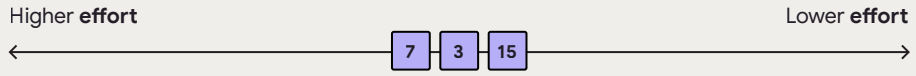
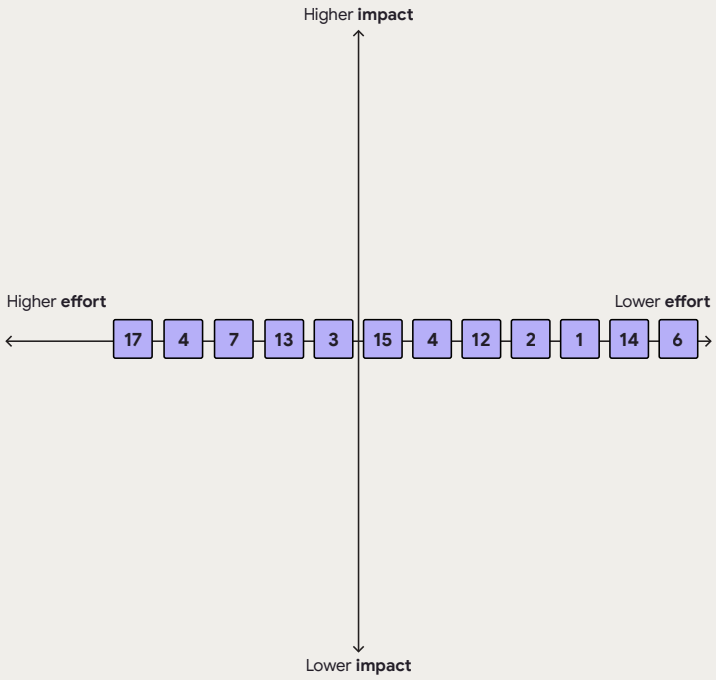


Figure 31: Effort axis with initial ideas

4. **Add impact axis:** Create a perpendicular axis and label as shown in Figure 32.

Add impact axis



Tip: Do not allow any sticky note to straddle the impact axis.
Figure 32: Add impact axis

5. **Focus on the impact:** Discuss the potential impact of each idea. Move each idea along the impact axis, positioning it relative to the others. Repeat until every sticky note is positioned on the grid.

All ideas mapped according to effort and impact

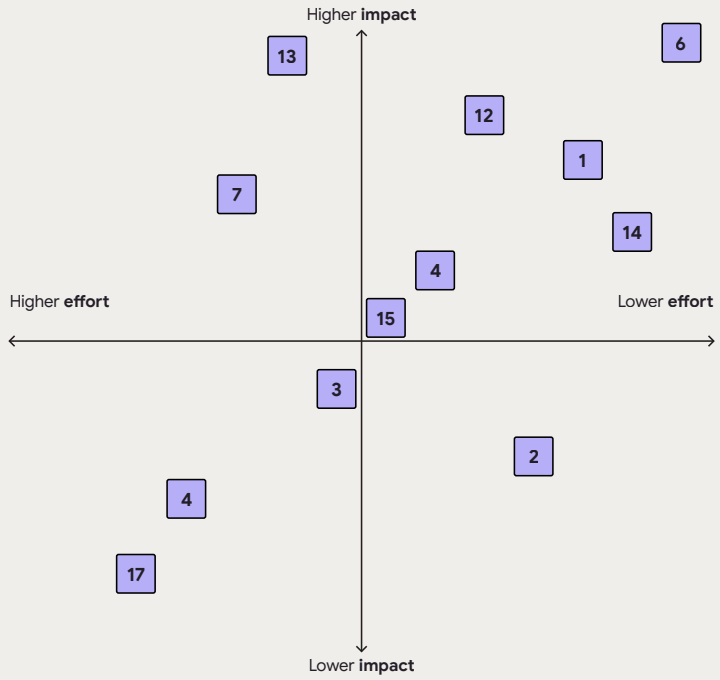


Figure 33: All ideas mapped according to effort and impact

6. **Assess your roadmap and identify your "quick wins":** One of the benefits of a 2x2 prioritization matrix is that you now have a strategic roadmap that has the agreement and consensus from the team based on the locations of the quadrants:
   - **Higher impact, lower effort:** This quadrant is your "quick wins"; do them now.
   - **Higher impact, higher effort:** This is the "do next" quadrant. It will take more time, effort, and thought to tackle these.
   - **Lower impact, lower effort:** This is the "do it later" quadrant.
   - **Lower impact, higher effort:** Don't do these. If an idea becomes a higher impact or lower effort, at some point it will move to a higher quadrant.

Prioritization order for the ideas

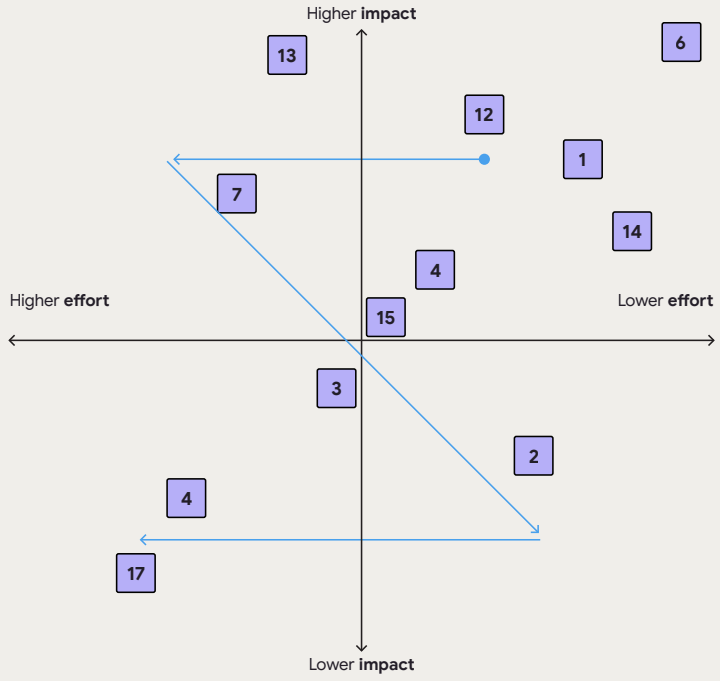


Figure 34: Prioritization order for the ideas

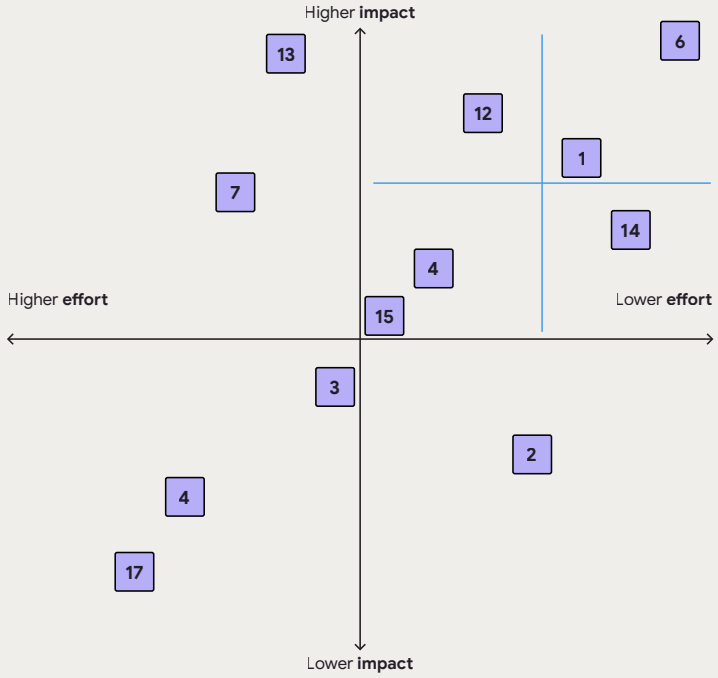
Step 4: Owning your "first step" (10 mins)

For this final step, identify the first one or two ideas and their owners.

1. Discuss and agree on just o**ne or two** initiatives. Avoid the trap of overcommitment; real change comes from completing one thing rather than having several things left unfinished.
2. It's key to identify someone on the team to "own" each idea. Keep in mind that the owner of an idea is responsible for shepherding the idea toward completion; they aren't required to execute the idea on their own.

The team should leave the workshop with a clear, actionable, and collectively owned plan for their next improvement. Each agreed action should have a clear owner and targets for completion and success criteria.

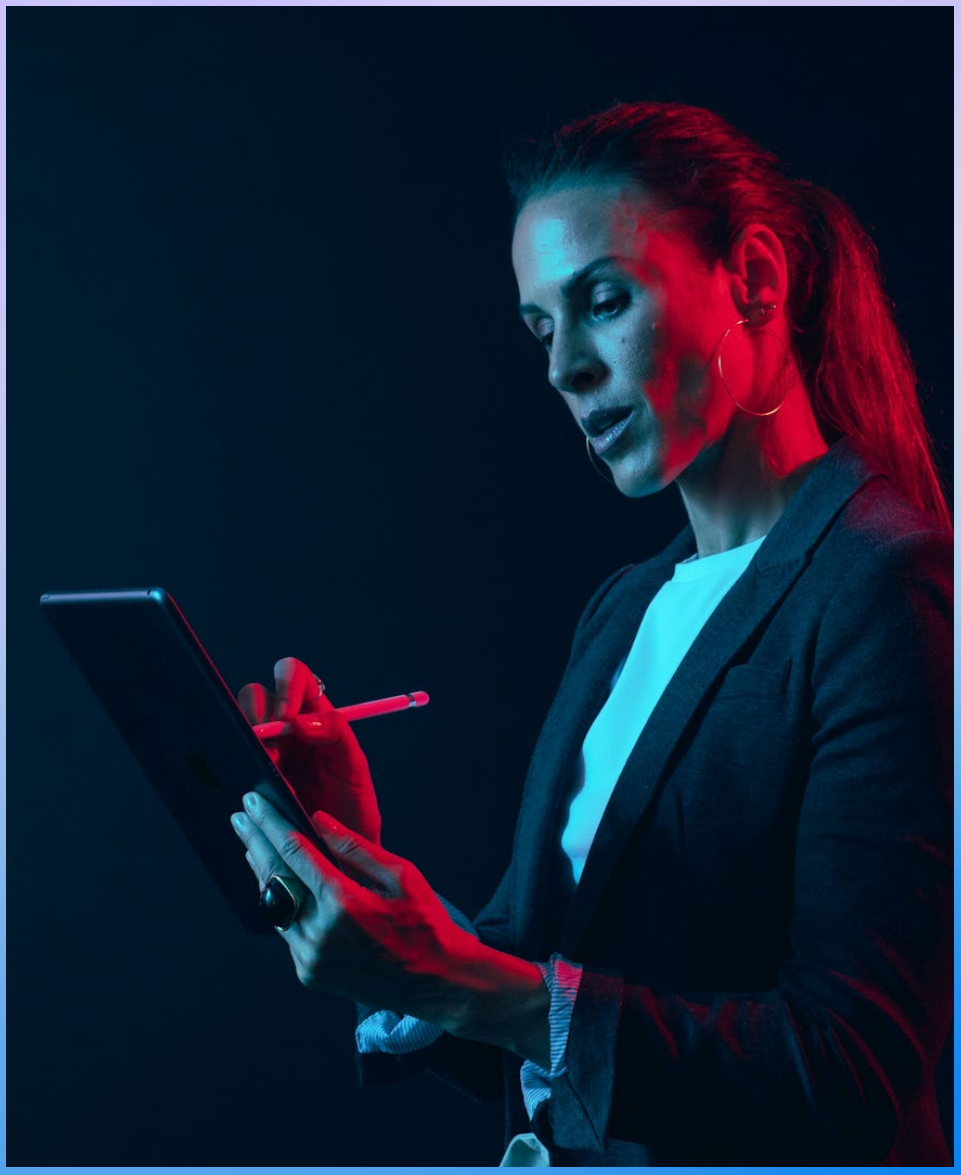
New axes drawn in the high impact, low effort quadrant.



Tip: Sometimes the team ends up with too many "quick wins." Consider moving the axes or drawing a new 2x2 in that quadrant to force a new prioritization.
Figure 35: New axes drawn in the high impact, low effort quadrant

Conclusion: Building your roadmap for AI-powered success

Dave Stanke
Specialist Customer Engineer, Google Cloud



For the overwhelming majority of technology organizations, the key questions surrounding the use of AI are no longer if or when—those have been answered: yes, and now. So we turn to the more nuanced issue of how. Research by DORA and others, plus the collective experience of many thousands of engineers, has shown that AI, in and of itself, is not assured to be beneficial. It must be adopted with care.

To achieve a return on investments made in acquiring and adapting to AI, teams should also attend to how they communicate, collaborate, and operate across their broader sociotechnical context.

DORA's research has revealed seven key predictors of success with AI, which are presented in the DORA AI Capabilities Model. This model works in concert with the established DORA Core Model.[1]

By investing in the core capabilities that drive all high-performing teams, and then focusing on these seven AI-specific capabilities, organizations can create the optimal environment to amplify AI's benefits.

Use this model as a starting point and reference as you continue on your unique journey of continuous improvement.

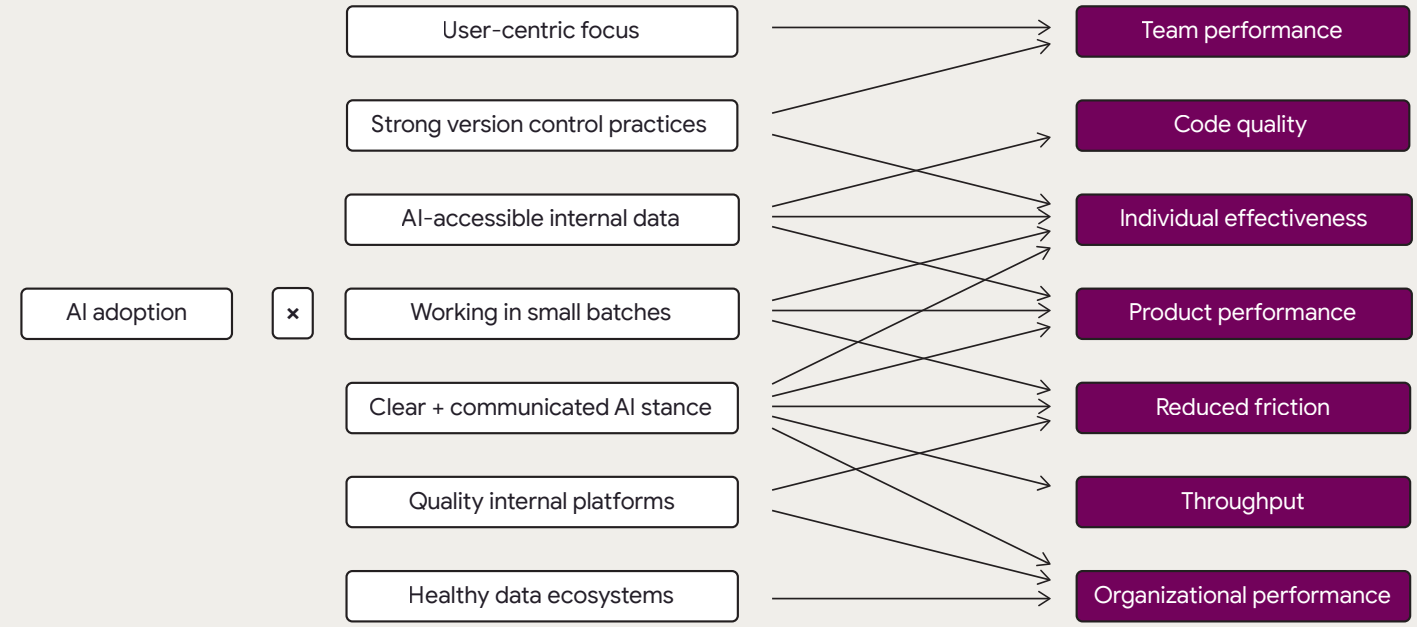


Figure 36: DORA AI Capabilities Model

Assessing your current state

A journey of improvement begins by establishing a baseline. DORA uses both quantitative and qualitative research methods, and you can leverage both within your organization. Using the DORA AI Capabilities Model, determine your current proficiency on each capability. You can conduct an internal survey with the same questions[2] used in the development of the model.

Additionally, the team profiles that emerged from DORA's survey response cluster analysis provide archetypes that describe most current teams.

Consult these profiles and find the one that most closely matches your experience. Conduct a value stream mapping (VSM) exercise to visualize your team's particular flow of value from ideation to end user, and discover opportunities for improvement.

Finally, don't overlook the simplest approach to gathering data: asking people. Qualitative data from interviews, observation, and open-ended survey questions can enrich your understanding not only of the team's status along known dimensions, but also to suggest previously unknown areas of inquiry: answers to questions you didn't think to ask.

Setting priorities

After establishing a baseline, set priorities for improvement. These form your roadmap, arranged in order—most crucially, the first item will be the first experiment you'll try. A multidimensional prioritization rubric is advised and considerations may include:

Predicted impact
Naturally, capabilities which are expected to have the biggest effect on key outcomes (such as productivity or user satisfaction) will be given precedence. However, these expectations must be balanced against other factors.

Current status
Each team has a distinct profile of performance across all of the AI capabilities. Consider starting in an area where the team is currently underperforming, as it may offer "low-lying fruit" and rapid improvement.

Locus of control
While teams should be encouraged to bridge organizational silos and influence change across team boundaries, it's also prudent to recognize limitations. It may be best to start with a local optimization, and demonstrate success, before attempting to effect broader change.

Subjective factors
There may be other dynamics to consider. For example, a particular project might have unique appeal to individuals on your team: someone's pet project. If that motivates them to invest in it, it's more likely to succeed.

Use survey data, metrics, and conversations to develop a complete picture of opportunities for improvement, and revise priorities continuously as you make changes and evaluate their impact.

Starting points

Your roadmap, and the projects it encompasses, will be unique to your organization. As a starting point, here is some practical advice based on the seven DORA AI capabilities:

Clear and communicated AI stance

Ambiguity around AI stifles adoption and creates risk. Establish and socialize a clear policy on permitted tools and usage to build developer trust. This clarity provides the psychological safety needed for effective experimentation, reducing friction and amplifying AI's positive impact on individual effectiveness and organizational performance.

Healthy data ecosystems

The benefits of AI on organizational performance are significantly amplified by a healthy data ecosystem. Invest in the quality, accessibility, and unification of your internal data sources. When your AI tools can learn from high-quality internal data, their value to your organization increases.

AI-accessible internal data

Connect your AI tools to your internal systems to move beyond generic assistance and unlock boosts in individual effectiveness and code quality. This means going beyond simply procuring licenses, and investing the engineering effort to give your AI tools secure access to internal documentation, codebases, and other data sources. This provides the company-specific context necessary for the tools to be maximally effective.

User-centric focus

Developers can experience large increases in their personal effectiveness when they adopt AI. However, if their users' needs aren't their focus, they may be moving quickly in the wrong direction. We found that adopting AI-assisted development tools can harm teams that don't have a user-centric focus. Conversely, keeping the users' needs as a product's North Star can guide AI-assisted developers toward appropriate goals and has an exceptionally strong positive effect on the performance of teams using AI.

Strong version control practices

AI-assisted coding can increase the volume and velocity of changes, which can also lead to more instability. Your version control system is a critical safety net. Encourage teams to become highly proficient in using rollback and revert features, as this practice is associated with better team performance in an AI-assisted environment.

Working in small batches

While AI can increase perceptions of individual effectiveness by generating large amounts of code, our findings show this isn't necessarily the most important metric. Instead, focus on outcomes. Enforce the discipline of working in small batches, which improves product performance and reduces friction for AI-assisted teams.

Quality internal platforms

A quality internal platform is a key enabler for magnifying the positive effects of AI on organizational performance. These platforms provide the necessary guardrails and shared capabilities that allow AI benefits to scale effectively and securely across the organization.

Developing a culture of continuous improvement

Continuous improvement is not an activity, it's a mindset. It's a deliberate practice of long-term growth and learning, through a program of iterative experimentation, always in service of the North Star of user value. Here are some techniques for developing this culture:

Celebrate progress, not attainment

Every team in your organization has a different starting point, and will encounter different obstacles. Progress will vary accordingly. Rather than declaring a single bar for success ("all batch sizes must be under 100 lines of code"), align on what "better" is ("prefer smaller batch sizes") and allow each team to improve on their current situation. Treat each increment of progress as a win and incentivize teams to always push for more. Struggling teams may still have a long way to go—a massive batch size, cut in half, is still large—but they can receive a share of praise for their efforts, which motivates ongoing improvement. And high-performing teams should be encouraged to continue to push the envelope.

Embrace failure

AI is shifting the role of a developer, from being primarily a direct author of code, to a higher-level strategist and orchestrator. In this role, curiosity, adaptation, and learning are more important than ever before. People learn by trying, and failing, and trying again. Facilitate innovation by enabling developers to fail, safely. This may include organizational constructs like hackathons, as well as technical systems like sandbox environments and access to new tools. Most importantly, make learning from failure something that is rewarded and incentivized: by embracing failure, developers are discouraged from hiding mistakes and encouraged to share new insights.

Use communities of practice

You can't do this alone, and you don't have to. While each team's journey is unique, the challenges they face, and strategies for overcoming them, are often universal. As part of a community of practice,[3] you can celebrate successes and failures, find common cause, and continue to build on the industry's collective knowledge. Use structures provided in your organization to establish an internal group to share information, and join DORA's global community[4] to connect with people from every industry, all over the world.

Conclusion

To realize the potential of AI, remember that its primary role is that of an amplifier, magnifying both strengths and dysfunctions within your organization. Therefore, focus on investing in the foundational technical and cultural environment and cultivating key capabilities like those in DORA's AI Capabilities Model.

By treating AI adoption as an organizational transformation and addressing systemic constraints, you can channel AI's power toward solving real customer problems and achieving sustainable success.

Get better at getting better.

1. "DORA's Research Program." https://dora.dev/research
2. "DORA Research: DORA AI Capabilities Model." https://dora.dev/ai/capabilities-model/questions
3. "How to transform your organization." https://dora.dev/guides/how-to-transform/#build-community-structures-to-spread-knowledge
4. "The DORA Community." https://dora.community

Acknowledgements

This report serves as a companion to the 2025 State of AI-assisted Software Development report.[1] We recommend you refer to that for more information about our methodology, models, and more.

Contributing authors and editors

Ameer Abbas

James Brookbank

Derek DeBellis

Rob Edwards

Nathen Harvey

Vivian Hu

Ben Jose

Amanda Lewis

Eric Maxwell

Allison Park

Jerome Simms

Dave Stanke

Kevin M. Storer, Ph.D.

Lucia Subatin

Seth Rosenblatt

Cedric Yao

Accuracy Matters[2]

Research and design partners

Apparent:[3] DORA branding

Human After All:[4] DORA report design

Prolific:[5] Research infrastructure support

User Research International:[6] Research infrastructure support

1. State of AI-assisted Software Development. https://dora.dev/dora-report-2025
2. Accuracy Matters. https://accuracymatters.co.uk
3. We Are Apparent. https://apparent.com.au
4. Human After All: Clarity through creativity. https://www.humanafterall.studio
5. Prolific | Easily collect high-quality data from real people. https://www.prolific.com
6. User Research for Product Development | User Research International. https://www.uriux.com

Check if you have the latest version of this report

Visit https://dora.dev/vc/aicm/?v=2025.1 to see if there's a newer version of this report available.

Get better at getting better

dora.dev

DORA