

DORA

2024



Google Cloud

Accelerate State of DevOps

Gold Sponsors

 catchpoint  chronosphere  DATADOG

 Deloitte.  Excella  Gearset

 liatrion  Middleware  OPSERA

10

A decade with DORA

Contents

Executive summary	3	Final thoughts	83
Software delivery performance	9	Acknowledgements	85
Artificial intelligence: Adoption and attitudes	17	Authors	87
Exploring the downstream impact of AI	27	Demographics and firmographics	91
Platform engineering	47	Methodology	99
Developer experience	57	Models	113
Leading transformations	69	Recommended reading	117
A decade with DORA	77		

Executive summary

DORA has been investigating the capabilities, practices, and measures of high-performing technology-driven teams and organizations for over a decade. This is our tenth DORA report. We have heard from more than 39,000 professionals working at organizations of every size and across many different industries globally. Thank you for joining us along this journey and being an important part of the research!

DORA collects data through an annual, worldwide survey of professionals working in technical and adjacent roles. The survey includes questions related to ways of working and accomplishments that are relevant across an organization and to the people working in that organization.

DORA uses rigorous statistical evaluation methodology to understand the relationships between these factors and how they each contribute to the success of teams and organizations.

This year, we augmented our survey with in-depth interviews of professionals as a way to get deeper insights, triangulate, and provide additional context for our findings. See the [Methodology](#) chapter for more details.

The key accomplishments and outcomes we investigated this year are:

Reducing burnout

Burnout is a state of emotional, physical, and mental exhaustion caused by prolonged or excessive stress, often characterized by feelings of cynicism, detachment, and a lack of accomplishment.

Flow

Flow measures how much focus a person tends to achieve during development tasks.

Job satisfaction

Job satisfaction measures someone's overall feeling about their job.

Organizational performance

This measures an organization's performance in areas including profitability, market share, total customers, operating efficiency, customer satisfaction, quality of products and services, and its ability to achieve goals.

Product performance

This measures the usability, functionality, value, availability, performance (for example, latency), and security of a product.

Productivity

Productivity measures the extent to which an individual feels effective and efficient in their work, creating value and achieving tasks.

Team performance

This measures a team's ability to collaborate, innovate, work efficiently, rely on each other, and adapt.

Key findings

AI is having broad impact

AI is producing a paradigm shift in the field of software development. Early adoption is showing some promising results, tempered by caution.

AI adoption benefits:

- Flow
- Productivity
- Job satisfaction
- Code quality
- Internal documentation
- Review processes
- Team performance
- Organizational performance

However, AI adoption also brings some detrimental effects. We have observed reductions to software delivery performance, and the effect on product performance is uncertain. Additionally, individuals are reporting a decrease in the amount of time they spend doing valuable work as AI adoption increases, a curious finding that is explored more later in this report.

Teams should continue experimenting and learning more about the impact of increasing reliance on AI.

AI adoption increases as trust in AI increases

Using generative artificial intelligence (gen AI) makes developers feel more productive, and developers who trust gen AI use it more. There is room for improvement in this area: 39.2% of respondents reported having little or no trust in AI.

User-centricity drives performance

Organizations that prioritize the end user experience produce higher quality products, with developers who are more productive, satisfied, and less likely to experience burnout.

Transformational leadership matters

Transformational leadership improves employee productivity, job satisfaction, team performance, product performance, and organizational performance while also helping decrease employee burnout.

Stable priorities boost productivity and well-being

Unstable organizational priorities lead to meaningful decreases in productivity and substantial increases in burnout, even when organizations have strong leaders, good internal documents, and a user-centric approach to software development.

Platform engineering can boost productivity

Platform engineering has a positive impact on productivity and organizational performance, but there are some cautionary signals for software delivery performance.

Cloud enables infrastructure flexibility

Flexible infrastructure can increase organizational performance. However, moving to the cloud without adopting the flexibility that cloud has to offer may be more harmful than remaining in the data center. Transforming approaches, processes, and technologies is required for a successful migration.

High-levels of software delivery performance are achievable

The highest performing teams excel across all four software delivery metrics (change lead time, deployment frequency, change fail percentage, and failed deployment recovery time) while the lowest performers perform poorly across all four. We see teams from every industry vertical in each of the performance clusters.

Applying insights from DORA

Driving team and organizational improvements with DORA requires that you assess how you're doing today, identify areas to invest in and make improvements, and have feedback loops to tell you how you're progressing. Teams that adopt a mindset and practice of continuous improvement are likely to see the most benefits. Invest in building the organizational muscles required to repeat this over time.

Findings from our research can help inform your own experiments and hypotheses. It's important to experiment and measure the impact of your changes to see what works best for your team and organization. Doing so will help you validate our findings. Expect your results to differ and please share your progress so that we all may learn from your experience.

We recommend taking an experimental approach to improvement.

1. Identify an area or outcome you would like to improve
2. Measure your baseline or current state
3. Develop a set of hypotheses about what might get you closer to your desired state
4. Agree and commit to a plan for improvement
5. Do the work
6. Measure the progress you've made
7. Repeat the process.
Improvement work is achieved iteratively and incrementally

DORA COMMUNITY



You cannot improve alone!

We can learn from each other's experience; an excellent forum for sharing and learning about improvement initiatives is the DORA Community <https://dora.community>.

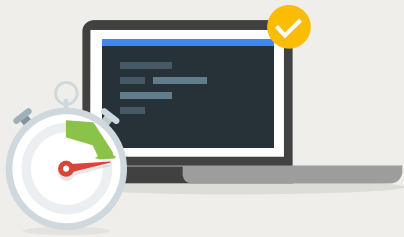
Software delivery performance

Technology-driven teams need ways to measure performance so that they can assess how they're doing today, prioritize improvements, and validate their progress. DORA has repeatedly validated four software delivery metrics — the four keys — that provide an effective way of measuring the outcomes of the software delivery process.



The four keys

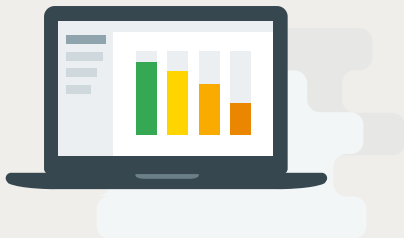
DORA's four keys have been used to measure the throughput and stability of software changes. This includes changes of any kind, including changes to configuration and changes to code.



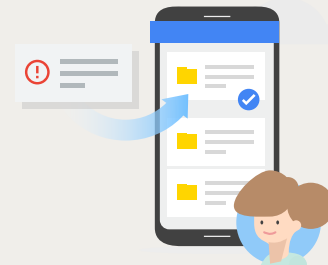
Change lead time:
the time it takes for a code commit or change to be successfully deployed to production.



Deployment frequency:
how often application changes are deployed to production.



Change fail rate:
the percentage of deployments that cause failures in production,¹ requiring hotfixes or rollbacks.



Failed deployment recovery time:
the time it takes to recover from a failed deployment.

We've observed that these metrics typically move together, the best performers do well on all four while the lowest performers do poorly.

Evolving the measures of software delivery performance

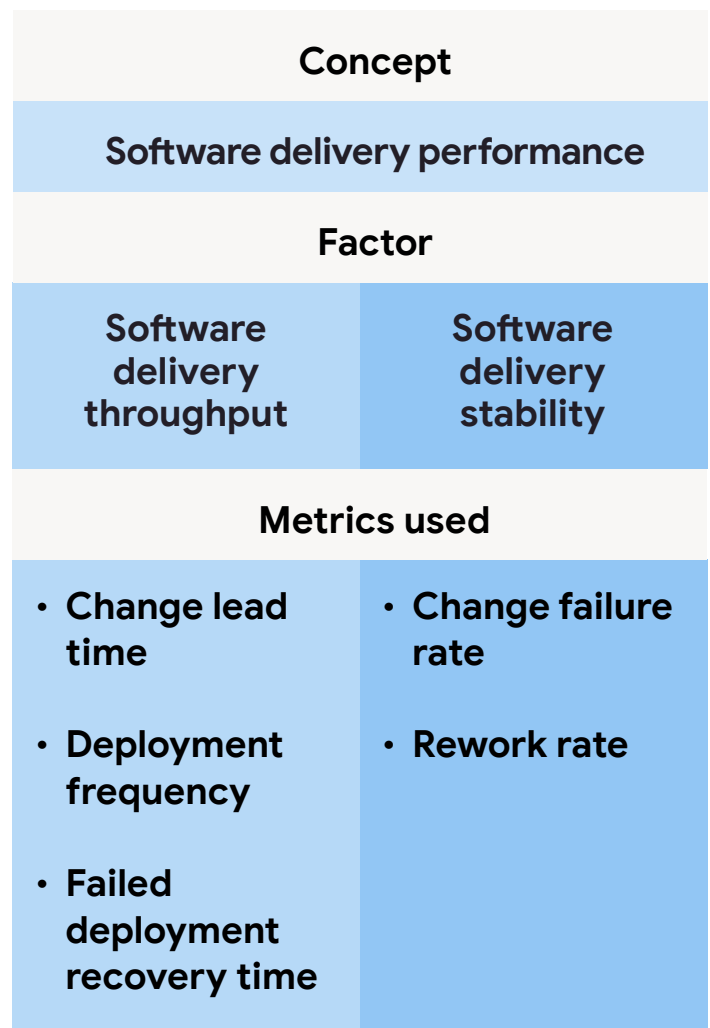
The analysis of the four key metrics has long had an outlier: change failure rate.² Change failure rate is strongly correlated with the other three metrics but statistical tests and methods prevent us from combining all four into one factor. A change to the way respondents answered the change failure rate question improved the connection but we felt there might be something else happening.

We have a longstanding hypothesis that the change failure rate metric works as a proxy for the amount of rework a team is asked to do. When a delivery fails, this requires the team to fix the change, likely by introducing another change.

To test this theory, we added another question this year about the rework rate for an application: "For the primary application or service you work on, approximately how many deployments in the last six months were not planned but were performed to address a user-facing bug in the application?"

Our data analysis confirmed our hypothesis that rework rate and change failure rate are related. Together, these two metrics create a reliable factor of software delivery stability.

This appears in the analysis of software performance levels, too. More than half of the teams in our study this year show differences in software throughput and software stability. These differences have led us to consider software delivery performance through two different factors:



Our analysis throughout this report utilizes the software delivery performance concept and both factors at various times. All five metrics are considered for describing software delivery performance.

Change lead time, deployment frequency, and failed deployment recovery time are used when we describe software delivery throughput. This factor measures the speed of making updates of any kind, normal changes and changes in response to a failure.

Change failure rate and rework rate are used when we describe software delivery stability. This factor measures the likelihood deployments unintentionally lead to immediate, additional work.



Performance levels

Each year we ask survey respondents about the software delivery performance of the primary application or service they work on. We analyze their answers using cluster analysis, which is a statistical method that identifies responses that are similar to one another but distinct from other groups of responses.

We performed the cluster analysis on the original four software delivery metrics to remain consistent with previous years' cluster analyses.

In our analysis of software delivery performance, four clusters of responses emerged. We do not set these levels in advance, rather we let them emerge from the survey responses. This gives us a way to see a snapshot of software delivery performance across all respondents each year.

Four distinct clusters emerged from the data this year, as shown below.

Performance level	Change lead time	Deployment frequency	Change fail rate	Failed deployment recovery time	Percentage of respondents*
Elite	Less than one day	On demand (multiple deploys per day)	5%	Less than one hour	19% (18-20%)
High	Between one day and one week	Between once per day and once per week	20%	Less than one day	22% (21-23%)
Medium	Between one week and one month	Between once per week and once per month	10%	Less than one day	35% (33-36%)
Low	Between one month and six months	Between once per month and once every six months	40%	Between one week and one month	25% (23-26%)

*89% uncertainty interval

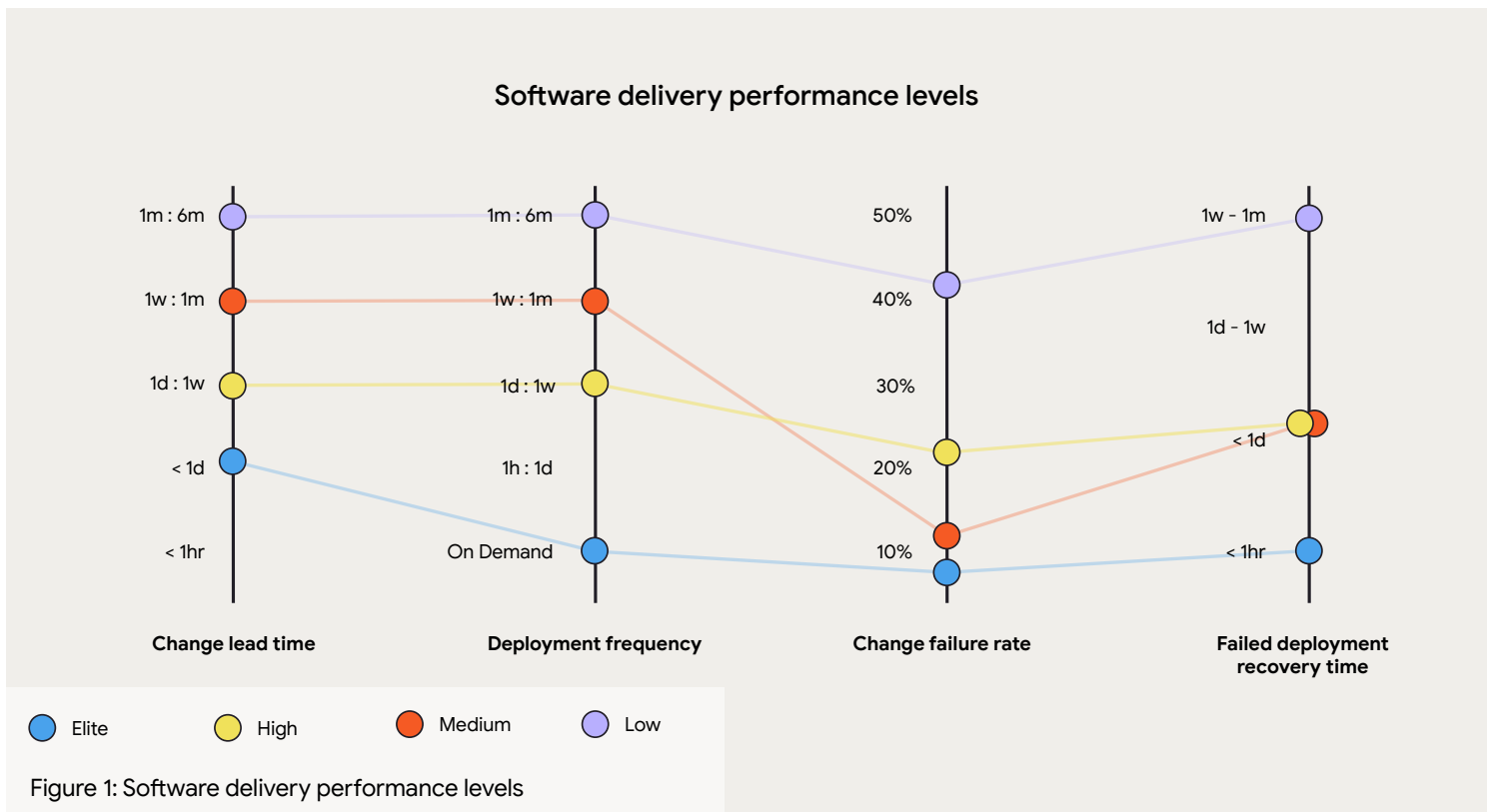
Throughput or stability?

Within all four clusters, throughput and stability are correlated. This correlation persists even in the medium performance cluster (orange), where throughput is lower and stability is higher than in the high performance cluster (yellow). This suggests that factors besides throughput and stability influence cluster performance. The medium performance cluster, for example, may benefit from shipping changes more frequently.

Which is better, more frequent deployments or fewer failures when deploying?

There may not be a universal answer to this question. It depends on the application or service being considered, the goals of the team working on that application, and most importantly the expectations of the application's users.

We made a decision to call the faster teams “high performers,” and the slower but more stable teams “medium performers.” This decision highlights one of the potential pitfalls of using these performance levels: Improving should be more important to a team than reaching a particular performance level. The best teams are those that achieve elite **improvement**, not necessarily elite **performance**.



When compared to low performers, elite performers realize

127x

faster lead time

182x

more deployments per year

8x

lower change failure rate

2293x

faster failed deployment recovery times

How to use the performance clusters

The performance clusters provide benchmark data that show the software delivery performance of this year's survey respondents. The clusters are intended to help inspire all that elite performance is achievable.

More important than reaching a particular performance level, we believe that teams should focus on improving performance overall. The best teams are those that achieve elite **improvement**, not necessarily elite **performance**.

Industry does not meaningfully affect performance levels

Our research rarely³ finds that industry is a predictor of software delivery performance; we see high-performing teams in every industry vertical. This isn't to suggest that there are no unique challenges across industries, but no one industry appears to be uniquely encumbered or uniquely capable when it comes to software delivery performance.

Using the software delivery performance metrics

Each application or service has its own unique context. This complexity makes it difficult to predict how any one change may affect the overall performance of the system. Beyond that, it is nearly impossible to change only one thing at a time in an organization. With this complexity in mind, how can we use the software delivery performance metrics to help guide our improvement efforts?

Start by identifying the primary application or service you would like to measure and improve. We then recommend gathering the cross-functional team responsible for this application to measure and agree on its current software delivery performance. The DORA Quick Check (<https://dora.dev/quickcheck>) can help guide a conversation and set this baseline measurement. Your team will need to understand what is preventing better performance.

One effective way to find these impediments is to complete a value stream mapping exercise⁴ with the team.

Next, identify and agree on a plan for improvement. This plan may focus on improving one of the many capabilities that DORA has researched⁵ or may be something else that is unique to your application or organization.

With this plan in-hand, it's now time to do the work! Dedicate capacity to this improvement work and pay attention to the lessons learned along the way.

After the change has had a chance to be implemented and take hold, it's now time to re-evaluate the four keys. How have they changed after the team implemented the change? What lessons have you learned?

Repeating this process will help the team build a practice of continuous improvement.

Remember: change does not happen overnight. An iterative approach that enables a climate for learning, fast flow, and fast feedback⁶ is required.

¹ We consider a deployment to be a change failure only if it causes an issue after landing in production, where it can be experienced by end users. In contrast, a change that is stopped on its way to production is a successful demonstration of the deployment process's ability to detect errors.

² Forsgren, Nicole, Jez Humble, and Gene Kim. 2018. Accelerate: The Science Behind DevOps : Building and Scaling High Performing Technology Organizations. IT Revolution Press. pp. 37-38

³ The 2019 Accelerate State of DevOps (p 32) report found that the retail industry saw significantly better software delivery performance. <https://dora.dev/research/2019/dora-report/2019-dora-accelerate-state-of-devops-report.pdf#page=32>

⁴ <https://dora.dev/guides/value-stream-management/>

⁵ <https://dora.dev/capabilities>

⁶ <https://dora.dev/research>

Artificial intelligence: Adoption and attitudes



Takeaways

The vast majority of organizations across all industries surveyed are shifting their priorities to more deeply incorporate AI into their applications and services. A corresponding majority of development professionals are relying on AI to help them perform their core role responsibilities — and reporting increases in productivity as a result. Development professionals’ perceptions that using AI is necessary for remaining competitive in today’s market is pervasive and appears to be an important driver of AI adoption for both organizations and individual development professionals.

Introduction

It would be difficult to ignore the significant impact that AI has had on the landscape of development work this year, given the proliferation of popular news articles outlining its effects, from good¹ to bad² to ugly.³ So, while AI was only discussed as one of many technical capabilities affecting performance in our 2023 Accelerate State of DevOps Report,⁴ this year we explore this topic more fully.

As the use of AI in professional development work moves rapidly from the fringes to ubiquity, we believe our 2024 Accelerate State of DevOps Report represents an important opportunity to assess the adoption, use, and attitudes of development professionals at a critical inflection point for the industry.

Findings

Adopting Artificial Intelligence

Findings on the adoption of AI suggest a growing awareness that AI is no longer “on the horizon,” but has fully arrived and is, quite likely, here to stay.

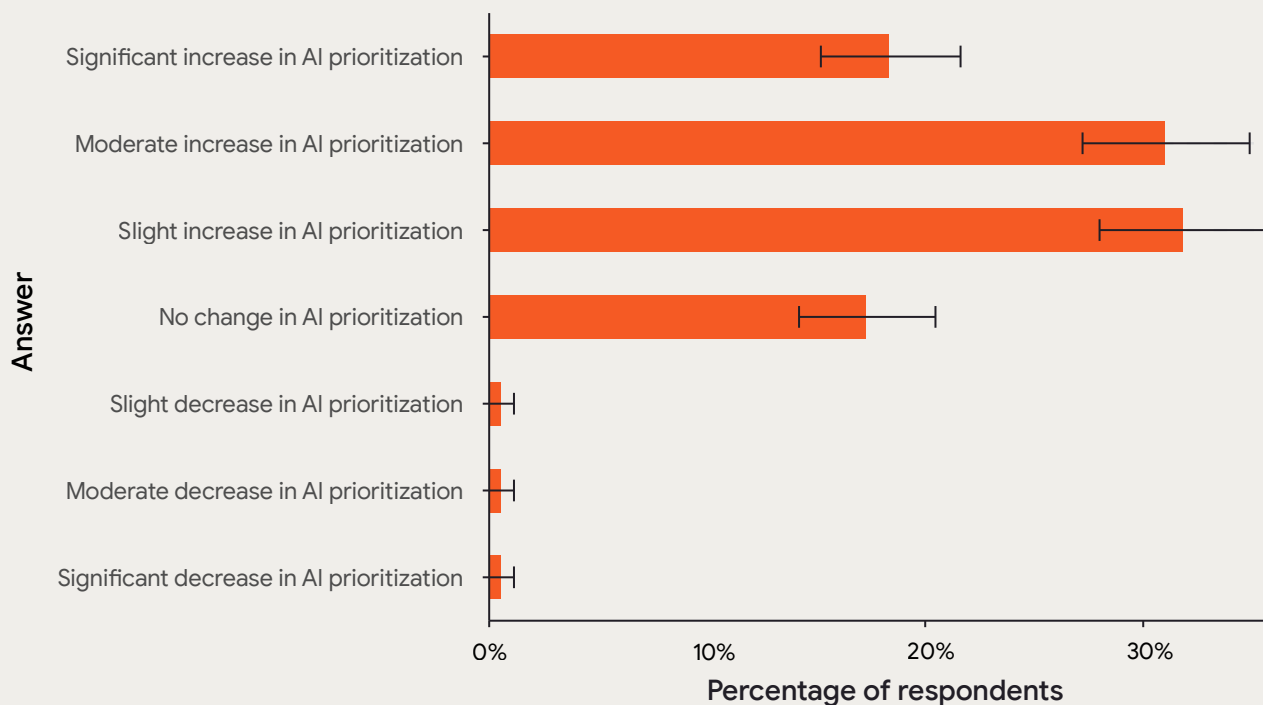
Organizational adoption of Artificial Intelligence

The vast majority of respondents (81%) reported that their organizations have shifted their priorities to increase their incorporation of AI into their applications

and services. 49.2% of respondents even described the magnitude of this shift as being either “moderate” or “significant.”

Notably, 3% of respondents reported that their organizations are decreasing focus on AI — within the margin of error of our survey. 78% of respondents reported that they trusted their organizations to be transparent about how they plan on using AI as a result of these priority shifts. This data is visualized in Figure 2.

Changes in organizational priorities concerning AI



Error bar represents 89% uncertainty interval

Figure 2: Respondents’ perceptions of their organizations’ shifts in priorities toward or away from incorporation of AI into their applications and services.

Participants from all surveyed industries reported statistically identical levels of reliance on AI in their daily work, which suggests that this rapid adoption of AI is unfolding uniformly across all industry sectors. This was somewhat surprising to us. Individual industries can vary widely with respect to their levels of regulatory constraints and historical pace of innovation, each of which can impact rates of technology adoption.

However, we did find that respondents working in larger organizations report less reliance on AI in their daily work than respondents working in smaller organizations, which is consistent with prior literature indicating larger firms more slowly adapt to technological change because of their higher organizational complexities and coordination costs.⁵



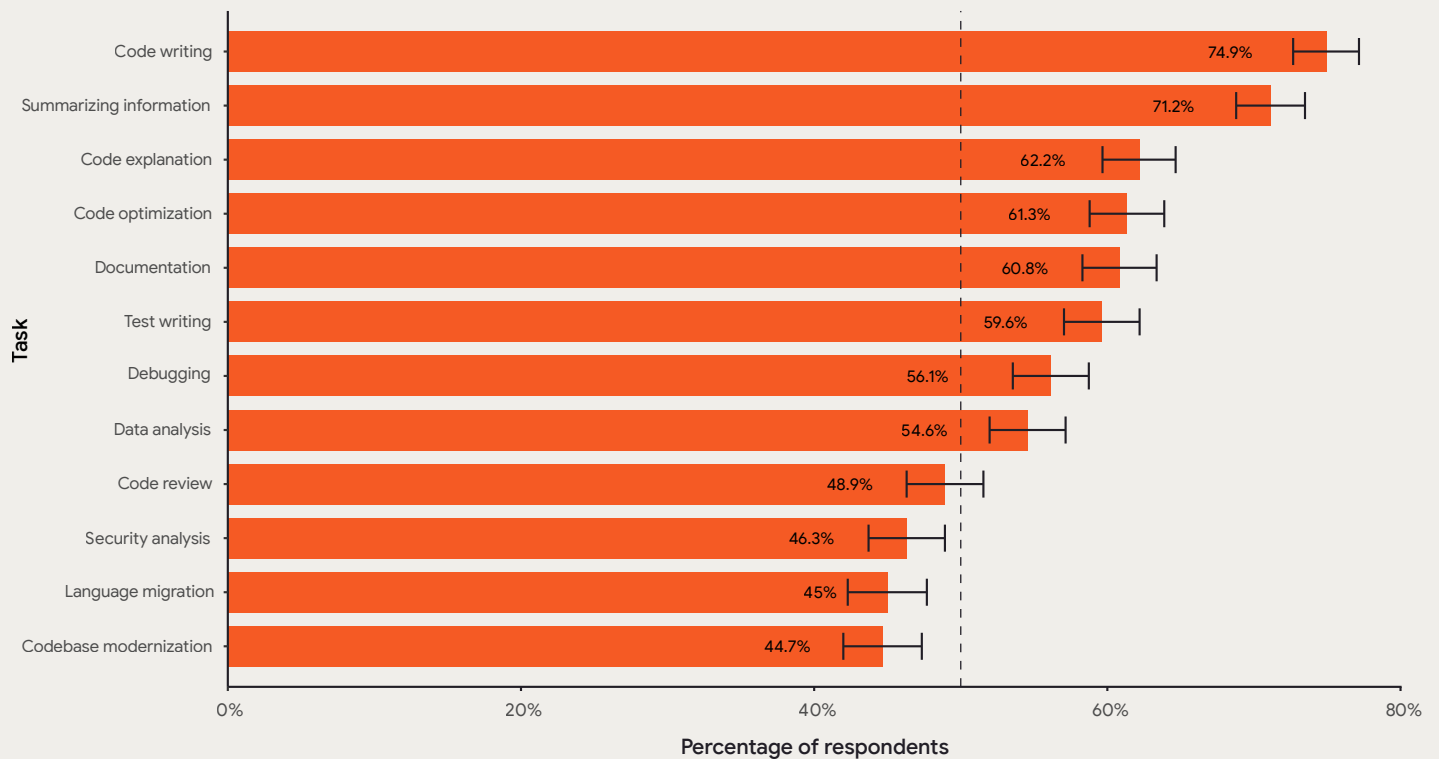
Individual adoption of artificial intelligence

At the individual level, we found that 75.9% of respondents are relying, at least in part, on AI in one or more of their daily professional responsibilities. Among those whose job responsibilities include the following tasks, a majority of respondents relied on AI for:

1. Writing code
2. Summarizing information
3. Explaining unfamiliar code
4. Optimizing code
5. Documenting code
6. Writing tests
7. Debugging code
8. Data analysis

Of all tasks included in our survey responses, the most common use cases for AI in software development work were writing code and summarizing information, with 74.9% and 71.2% of respondents whose job responsibilities include these tasks relying on AI to perform them — at least in part. This data is visualized in Figure 3.

Task reliance on AI



Error bar represents 89% credibility interval

Figure 3: Percentage of respondents relying on AI, at least in part, to perform twelve common development tasks

Chatbots were the most common interface through which respondents interacted with AI in their daily work (78.2%), followed by external web interfaces (73.9%), and AI tools embedded within their IDEs (72.9%). Respondents were less likely to use AI through internal web interfaces (58.1%) and as part of automated CI/CD pipelines (50.2%).

However, we acknowledge that respondents' awareness of AI used in their CI/CD pipelines and internal platforms likely depends on the frequency with which they interface with

those technologies. So, these numbers might be artificially low.

We found that data scientists and machine learning specialists were more likely than respondents holding all other job roles to rely on AI. Conversely, hardware engineers were less likely than respondents holding all other job roles to rely on AI, which might be explained by the responsibilities of hardware engineers differing from the above tasks for which AI is commonly used.



Drivers of adoption of artificial intelligence

Interview participants frequently linked the decision to adopt AI to competitive pressures and a need to keep up with industry standards for both organizations and developers, which are increasingly recognized to include proficiency with AI.

For several participants' organizations, using AI at all was seen as "a big marketing point" (P3)⁶ that could help differentiate their firm from competitors. Awareness that competitors are beginning to adopt AI in their own processes even prompted one firm to forgo the typical "huge bureaucracy" involved in adopting new technology because they felt an urgency to adopt AI, questioning "what if our competitor takes those actions before us?" (P11).

At the individual level, many participants linked their adoption of AI to the sentiment that proficiency with using AI in software development is "kind of, like, the new bar for entry as an engineer" (P9). Several participants suggested fellow developers should rapidly adopt AI in their development workflow, because "there's so much happening in this space, you can barely keep up... I think, if you don't use it, you will be left behind quite soon" (P4).

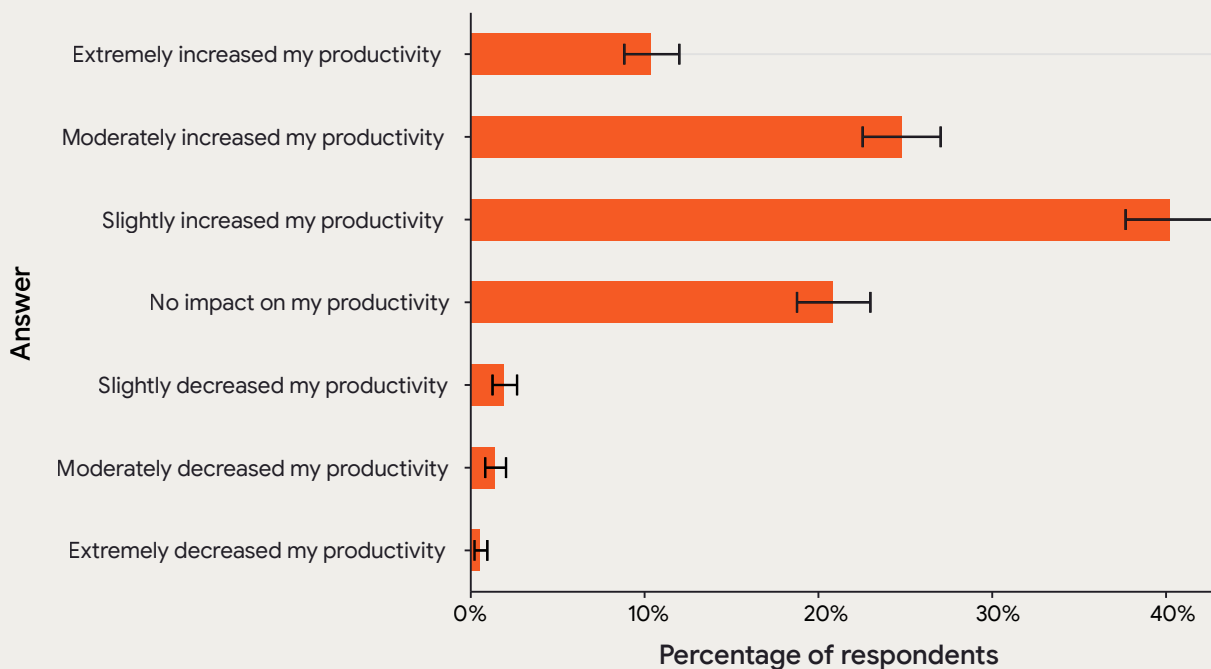
Perceptions of artificial intelligence

Performance improvements from artificial intelligence

For the large number of organizations and developers who are adopting it, the benefits of using AI in development work appear to be quite high. Seventy-five percent of respondents reported positive productivity gains from AI in the three months preceding our survey, which was fielded in early 2024.

Notably, more than one-third of respondents described their observed productivity increases as either moderate (25%) or extreme (10%) in magnitude. Fewer than 10% of respondents reported negative impacts of even a slight degree on their productivity because of AI. This data is visualized in Figure 4.

Perceptions of productivity changes due to AI



Error bar represents 89% uncertainty interval

Figure 4: Respondents' perceptions of AI's impacts on their productivity.

Across roles, respondents who reported the largest productivity improvements from AI were security professionals, system administrators, and full-stack developers. Although they also reported positive productivity improvement, mobile developers, site reliability engineers, and project managers reported lower magnitudes of productivity benefits than all other named roles.

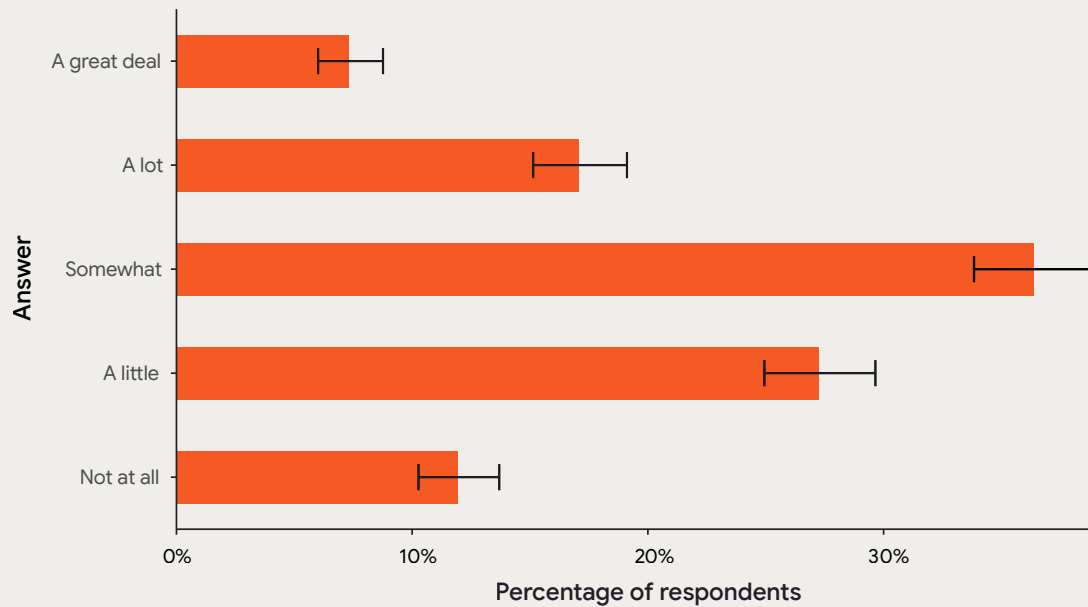
Although we suspected that the novelty of AI in development work, and corresponding learning curve, might inhibit developers' ability to write code, our findings did not support that hypothesis. Only 5% of respondents reported that AI had inhibited their ability to write code to any degree. In fact, 67% of respondents reported at least some improvement to their ability to write code as a result of AI-assisted coding tools, and about 10% have observed "extreme" improvements to their ability to write code because of AI.

Trust in AI-generated code

Participants' perceptions of the trustworthiness of AI-generated code used in development work were complex. While the vast majority of respondents (87.9%) reported some level of trust in the quality of AI-generated code, the degree to which respondents reported trusting the quality of AI-generated code was generally low, with 39.2% reporting little (27.3%) or no trust (11.9%) at all. This data is visualized in Figure 5.



Trust in quality of AI-generated code



Error bar represents 89% uncertainty interval

Figure 5: Respondents' reported trust in the quality of AI-generated code.

Given the evidence from the survey that developers are rapidly adopting AI, relying on it, and perceiving it as a positive performance contributor, we found the overall lack of trust in AI surprising. It's worth noting that during our interviews, many of our participants indicated that they were willing to, or expected to, tweak the outputs of the AI-generated code they used in their professional work.

One participant even likened the need to evaluate and modify the outputs of AI-generated code to “the early days of StackOverflow, [when] you always thought people on StackOverflow are really experienced, you know, that they will know exactly what to do. And then, you just copy and paste the stuff, and things explode” (P2).

Perhaps because this is not a new problem, participants like P3 felt that their companies are not “worried about, like, someone just copy-and-pasting code from Copilot or ChatGPT [because of] having so many layers to check it” with their existing code-quality assurance processes.

We hypothesize that developers do not necessarily expect absolute trust in the accuracy of AI-generated code, nor does absolute trust appear to be required for developers to find AI-generated code useful. Rather, it seems that mostly-correct AI-generated code that can be perfected with some tweaks is acceptable, sufficiently valuable to motivate widespread adoption and use, and compatible with existing quality assurance processes.

Expectations for AI's future

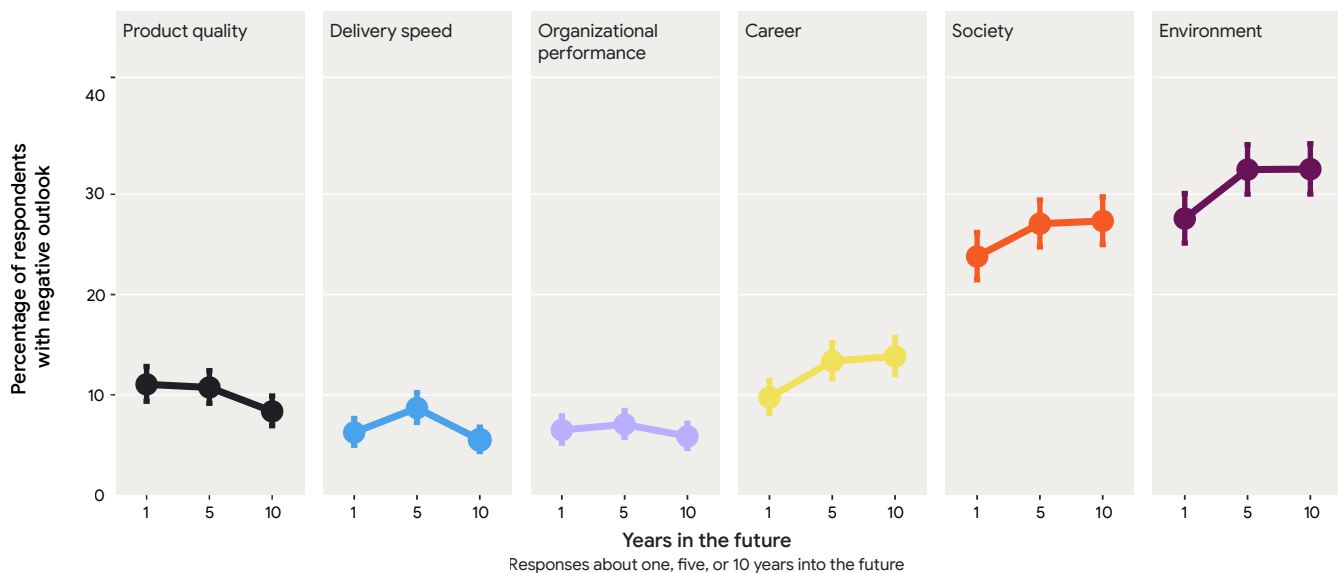
Overall, our findings indicate AI has already had a massive impact on development professionals' work, a trend we expect to continue to grow. While it would be impossible to predict exactly how AI will impact development — and our world — in the future, we asked respondents to speculate and share their expectations about the impacts of AI in the next one, five, and 10 years.

Respondents reported quite positive impacts of AI on their development work in reflecting on their recent experiences, but their predictions for AI's future impacts were not as hopeful.

Optimistically, and consistent with our findings that AI has positively impacted development professionals' performance, respondents reported that they expect the quality of their products to continue to improve as a result of AI over the next one, five, and 10 years.

However, respondents also reported expectations that AI will have net-negative impacts on their careers, the environment, and society, as a whole, and that these negative impacts will be fully realized in about five years time. This data is visualized in Figure 6.

Expected negative impacts of AI



Error bar represents 89% credibility interval

Figure 6: Respondents' expectations about AI's future negative impacts in the next one, five, and 10 years.

Interview participants held similarly mixed feelings about the future impacts of AI as our survey respondents. Some wondered about future legal actions in a yet-to-be-decided regulatory landscape, worrying they might “be on the wrong side of it, if things get decided” (P3).

Others echoed long-held anxieties and asked, “Is it going to replace people? Who knows? Maybe.” (P2), while their peers dismissed their fears by drawing parallels to the past, when “people used to say ‘Oh, Y2K! Everything will be doomed!’ Blah, blah... because it was a new thing, at that time.

[But,] nothing got replaced. In fact, there were more jobs created. I believe the same thing will happen with AI” (P1).

The future effects AI will have on our world remain unclear. But, this year, our survey strongly indicates that AI has produced an unignorable paradigm shift in the field of software development. So far, the changes have been well-received by development professionals.



1. <https://www.sciencedaily.com/releases/2024/03/240306144729.htm>

2. <https://tech.co/news/list-ai-failures-mistakes-errors>

3. <https://klyker.com/absurd-yoga-poses-generated-by-ai/>

4. <https://dora.dev/dora-report-2023>

5. Rogers, Everett M., Arvind Singhal, and Margaret M. Quinlan. “Diffusion of innovations.” An integrated approach to communication theory and research. Routledge, 2014. 432-44, Tornatzky, L. G., & Fleischer, M. (1990). The processes of technological innovation. Lexington, MA: Lexington Books

6. (P[N]), for example (P1), indicates pseudonym of interview participants.

Exploring the downstream impact of AI



Takeaways

This chapter investigates the impact of AI adoption across the spectrum, from individual developers to entire organizations. The findings reveal a complex picture with both clear benefits and unexpected drawbacks. While AI adoption boosts individual productivity, flow, and job satisfaction, it may also decrease time spent on valuable work.

Similarly, AI positively impacts code quality, documentation, and review processes, but surprisingly, these gains do not translate to improved software delivery performance. In fact, AI adoption appears detrimental in this area, while its effect on product performance remains negligible.

Despite these challenges, AI adoption is linked to improved team and organizational performance. This chapter concludes with a call to critically evaluate AI's role in software development and proactively adapt its application to maximize benefits and mitigate unforeseen consequences.

The AI moment & DORA

Estimates suggest that leading tech giants will invest approximately \$1 trillion on the development of AI in the next five years.¹ This aligns well with a statistic presented in the "[Artificial intelligence: Adoption and attitudes](#)" chapter that 81% of respondents say their company has shifted resources into developing AI.

The environmental impacts of AI further compound the costs. Some estimates suggest that by 2030, AI will drive an increase in data center power demand by 160%.² The training of an AI model can add up to roughly “the yearly electricity consumption of over 1,000 U.S. households”.³ It is no surprise that more than 30% of respondents think AI is going to be detrimental to the environment.

Beyond the development and environmental costs, we have the potential for adoption costs.

This could come in many forms, from productivity decreases to the hiring of specialists. These adoption costs could also come at a societal level. Over a third of respondents believe AI will harm society in the coming decade. Given these costs, it seems natural for people to have a deep curiosity about the returns.

This curiosity has manifested itself in a wealth of media, articles, and research whose sentiment and data are both mixed, at least to some extent.



Some believe that AI has dramatically enhanced the ability of humanity,⁴ others suggest that AI is little more than a benign tool for helping with homework,⁵ and some fear that AI will be the downfall of humanity.⁶

Evidence for proximal outcomes, such as the ability to successfully complete a particular task, is largely positive.⁷ When the outcome becomes more distant, such as a team’s codebase, the results start becoming a little less clear and a little less positive. For example, some research has suggested that code churn may double from the pre-2021 baseline.⁸

The challenge of understanding these downstream effects is unsurprising. The further away the effect is from the cause, the less pronounced and clear the connection.

Evaluating the downstream effects of AI mimics quantifying the effect of a rock thrown into a lake. You can most easily attribute the ripples closest to the impact point of the rock in the water, but the farther from the entry point you go, the less pronounced the effect of the rock is and the harder it is to ascribe waves to its impact.

AI is essentially a rock thrown into a stormy sea of other processes and dynamics. Understanding the extent of the waves caused by AI (or any technology or practice) is a challenge. This may be part of the reason the industry has struggled to adopt a principled set of measurement and analytic frameworks for understanding the impact of AI.⁹

Our approach is specifically designed to be useful for these types of challenges. DORA is designed to understand the utility or disutility of a practice. We've explored the downstream impacts of myriad practices over the last 10 years, including security practices, transformational leadership, generative cultures, documentation practices, continuous integration, continuous delivery, and user-centricity.¹⁰

We believe that DORA's approach¹¹ can help us learn about AI's impact, especially as we explore the effects of AI across many outcomes.



Measuring AI adoption

The first challenge of capturing the impact of adopting AI is measuring the adoption of AI. We determined measuring usage frequency is likely not as meaningful as measuring reliance for understanding AI's centrality to development workflows. You might only do code reviews or write documentation a few times a month or every couple of months, but you see these tasks as critically important to your work.

Conversely, just because you use AI frequently does not mean that you are using AI for work that you consider important or central to your role.

Given this, we asked respondents about their reliance on AI in general and for particular tasks. The [previous chapter](#) details the survey results and their interpretation.

Using factor analysis, we found our “general” AI reliance survey item had high overlap with reported AI reliance on the following tasks:

- Code Writing
- Summarizing information
- Code explanation
- Code optimization
- Documentation
- Test writing

The strong commonality and covariance among these seven items suggests an underlying factor that we call AI adoption.

AI's impact on individuals is a story of clear benefits (and some potential tradeoffs)

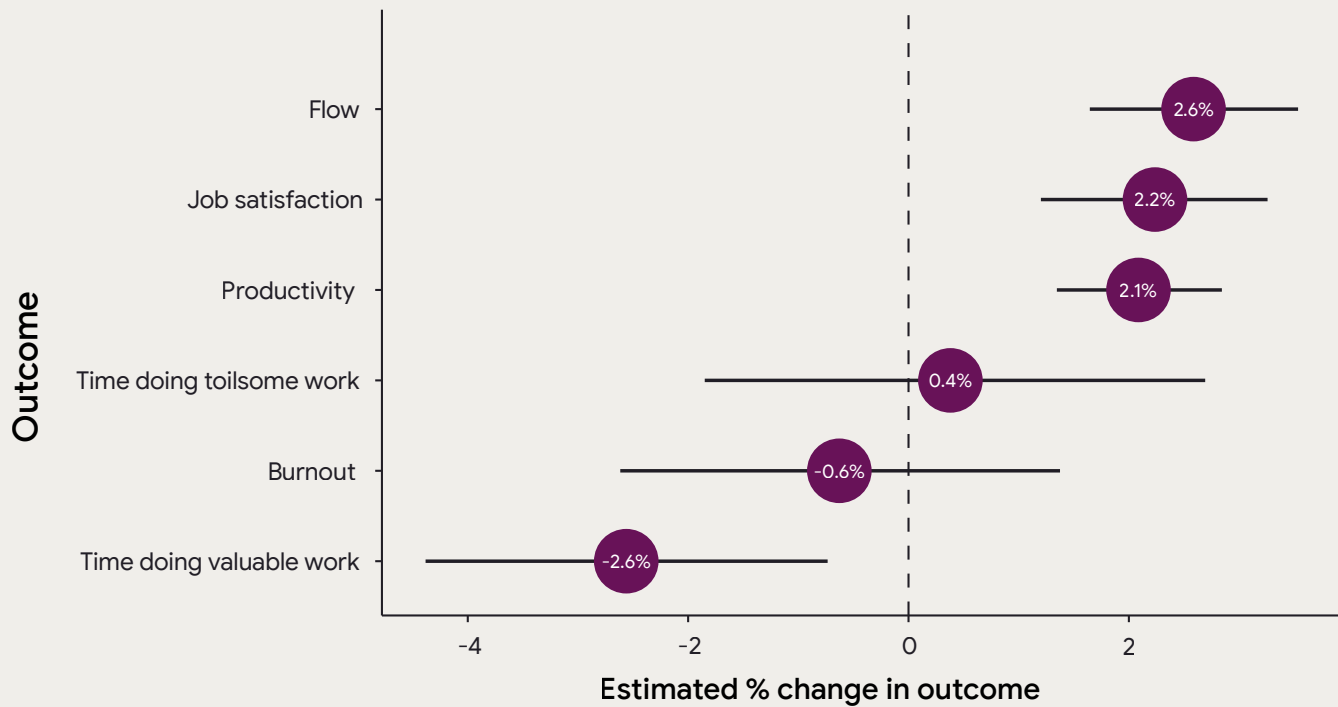
As we do every year, we measured a variety of constructs related to an individual's success and well-being:

Job satisfaction	A single item designed to capture someone's overall feeling about their job.
Burnout	A factor that encapsulates the multifaceted nature of burnout, encompassing its physical, emotional, and psychological dimensions, as well as its impact on personal life.
Flow	A single item designed to capture how much focus a person tends to achieve during development tasks.
Productivity	A factor score designed to measure the extent an individual feels effective and efficient in their work, creating value and achieving tasks.
Time doing toilsome work	A single item measuring the percentage of an individual's time spent on repetitive, manual tasks that offer little long-term value.
Time doing valuable work	A single item measuring the percentage of an individual's time spent on tasks that they consider valuable.

We wanted to figure out if the way respondents answered these questions changes as a function of adopting AI. The results suggest that is often the case.

Figure 7 is a visualization that shows our best estimates about the impact of adopting AI on an individual's success and well-being.

If an individual increases AI adoption by 25%...



Point = estimated value

Error bar = 89% uncertainty interval

Figure 7: Impacts of AI adoption on individual success and well-being

The clear benefits

The story about the benefit of adopting AI for individuals is largely favorable, but like any good story, has some wrinkles. **What seems clear is that AI has a substantial and beneficial impact on flow, productivity, and job satisfaction (see Figure 7).**

Productivity, for example, is likely to increase by approximately 2.1% when an individual's AI adoption is increased by 25% (see Figure 7). This might seem small, but this is at the individual-level. Imagine this pattern extended across tens of developers, or even tens of thousands of developers.

This pattern is what we expected. We believe it emerged in part thanks to AI's ability to synthesize disparate sources of information and give a highly personalized response in a single location. Doing this on your own takes time, lots of context switching, and is less likely to foster flow.

Given the strong connection that productivity and flow have with job satisfaction, it shouldn't be surprising that we see AI adoption leads to higher job satisfaction.

The potential tradeoffs

Here is where the story gets a little complicated. One value proposition for adopting AI is that it will help people spend more time doing valuable work. That is, by automating the manual, repetitive, toilsome tasks, we expect respondents will be free to use their time on “something better.” However, our data suggest that increased AI adoption may have the opposite effect—reducing reported time spent doing valuable work—while time spent on toilsome work appears to be unaffected.

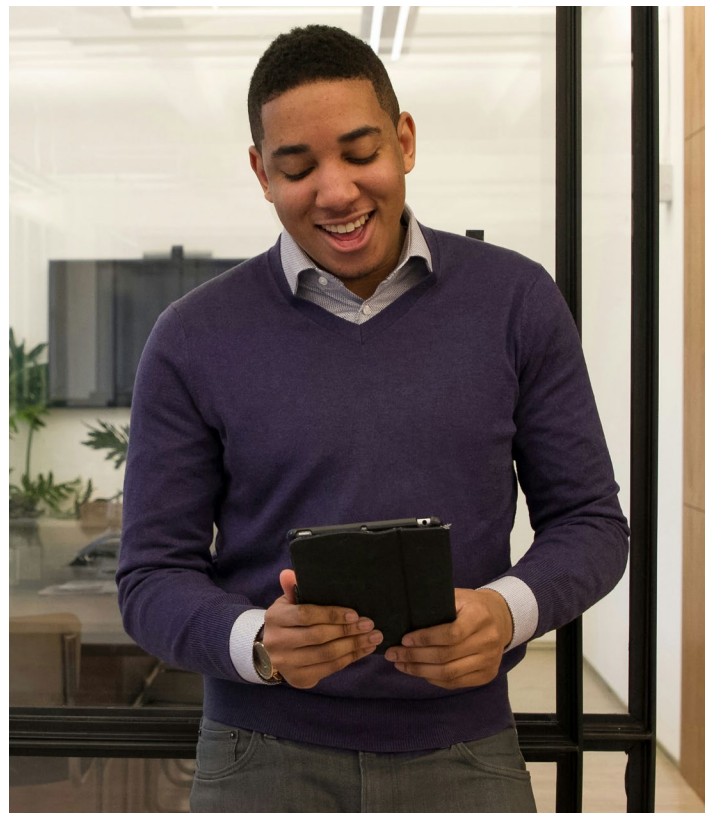
Markers of respondents’ well-being, like flow, job satisfaction, and productivity have historically been associated with time spent doing valuable work. So, observed increases in these measures independently of decreases in time spent on valuable work are surprising.

A good explanation of these patterns will need to wrestle with this seeming incongruity. A good explanation of a movie cannot ignore a scene that contradicts the explanation. A good explanation of a book cannot ignore a chapter that doesn’t fit neatly into the explanation. Similarly, a good explanation of these patterns cannot just focus on a subset of the patterns that allows us to tell a simple story.

There are innumerable hypotheses that could fit the data, but we came up with a hypothesis that seems parsimonious with flow, productivity, and job satisfaction benefitting from AI while time spent doing valuable work decreases and toil remains unchanged.

We call our hypothesis the vacuum hypothesis. By increasing productivity and flow, AI is helping people work more efficiently. This efficiency is helping people finish up work they consider valuable faster.

This is where the vacuum is created; there is extra time. AI does not steal value from respondents’ work, it expedites its realization.



Wait, what is valuable work?

To make sense of these counterintuitive findings we explored more deeply what types of work respondents judge to be valuable or toilsome.

Traditional wisdom, our past reports, and qualitative data from our interviews suggest that respondents find development-related tasks, like coding, to be valuable work, while less-valuable, even toilsome, work typically includes tasks associated with organizational coordination, like attending meetings. Within this categorization scheme, AI is better poised to assist with “valuable” work than “toilsome” work, as defined by respondents.

We turned to qualitative data from our interviews and found that, when responding to the moderator’s question of whether or not they would consider their work “meaningful,” participants frequently measured the value of their work in relation to the impact of their work on others.

This is solidified by two years of past DORA evidence of the extremely beneficial impact of user-centricity on job satisfaction.

For example, when describing a recent role shift, P10¹² indicated making the decision because “It helps me impact more people. It helps me impact more things.” Similarly, P11 noted “if you build something from scratch and see it’s delivered to a consumer or customer, you can feel that achievement, you can say to yourself, ‘Yeah! I delivered this and people use that!’”

Understanding that the “meaningfulness” of development work is derived from the impact of the solution created—not directly from the writing of the code—helps explain why we observed respondents spending less time on valuable work, while also feeling more satisfied with their jobs.

While AI is making the tasks people consider valuable easier and faster, it isn't really helping with the tasks people don't enjoy. That this is happening while toil and burnout remain unchanged, obstinate in the face of AI adoption, highlights that AI hasn't cracked the code of helping us avoid the drudgery of meetings, bureaucracy, and many other toilsome tasks (Figure 8).

The good news is that AI hasn't made it worse, nor has it negatively affected respondents' well-being.

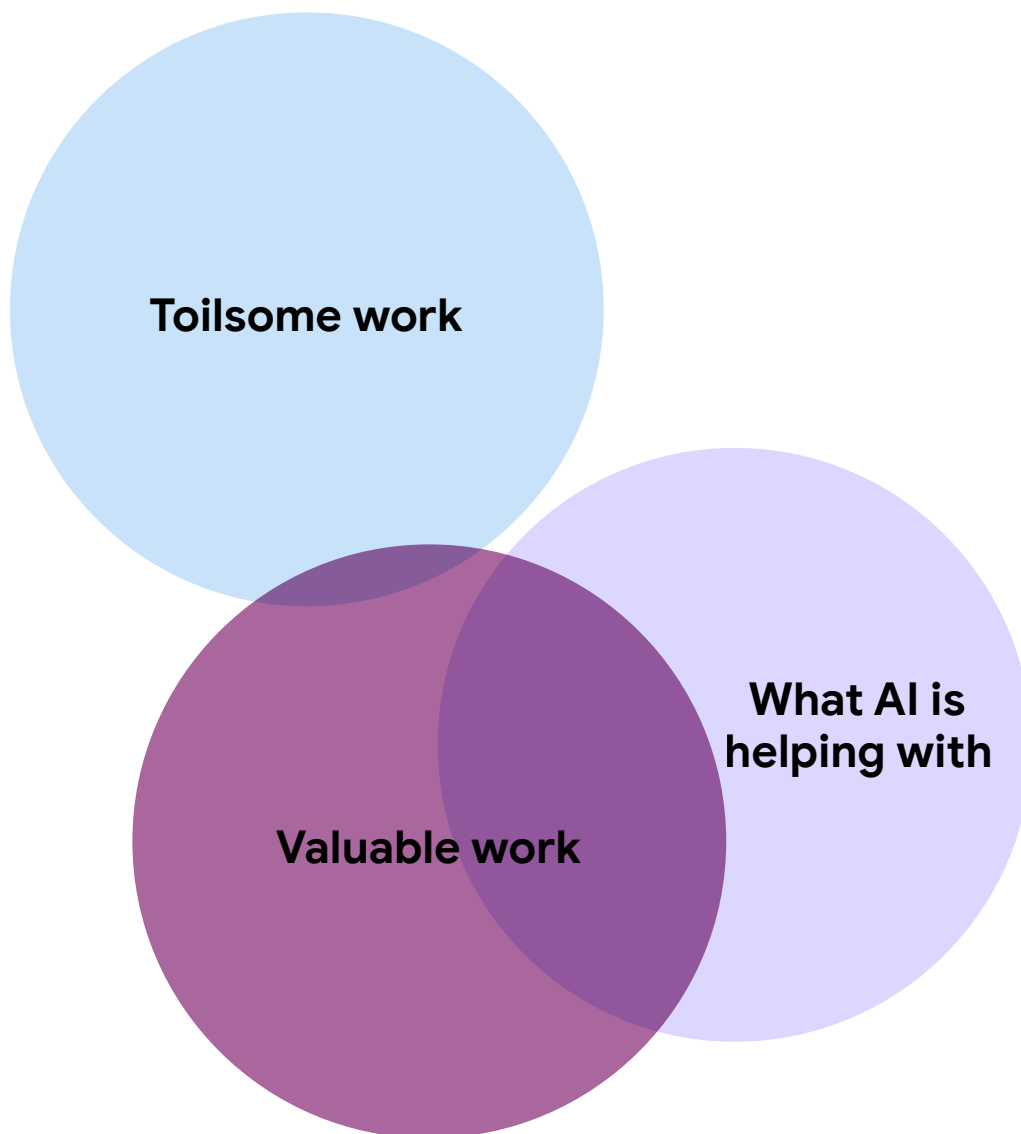


Figure 8: Not data, but a visualization of our hypothesis: AI is helping with our valuable work, but not helping us with our toil.

The promising impact of AI on development workflows

The last section explored outcomes focused on the individual. The next set of outcomes shift focus to explore processes, codebases, and team coordination. Here is a list of the outcomes we measured:

Code complexity	The degree to which code's intricacy and sophistication hinders productivity.
Technical debt	The extent to which existing technical debt within the primary application or service has hindered productivity over the past six months.
Code review speed	The average time required to complete a code review for the primary application or service.
Approval speed	The typical duration from proposing a code change to receiving approval for production use in the primary application or service.
Cross-functional team (XFN) coordination	The level of agreement with the statement: "Over the last three months, I have been able to effectively collaborate with cross-functional team members."
Code quality	The level of satisfaction or dissatisfaction with the quality of code underlying the primary service or application in the last six months.
Documentation quality	The perception of internal documentation (manuals, readmes, code comments) in terms of its reliability, findability, updatedness, and ability to provide support.

As before, our goal here is to understand if these aspects seem to vary as a function of adopting AI. Figure 9 is a visualization that shows our best estimates of the change in these outcomes in relation to a 25% increase in AI adoption.

Overall, the patterns here suggest a very compelling story for AI. Here are the substantial results from this section.

A 25% increase in AI adoption is associated with a...

7.5% increase in documentation quality

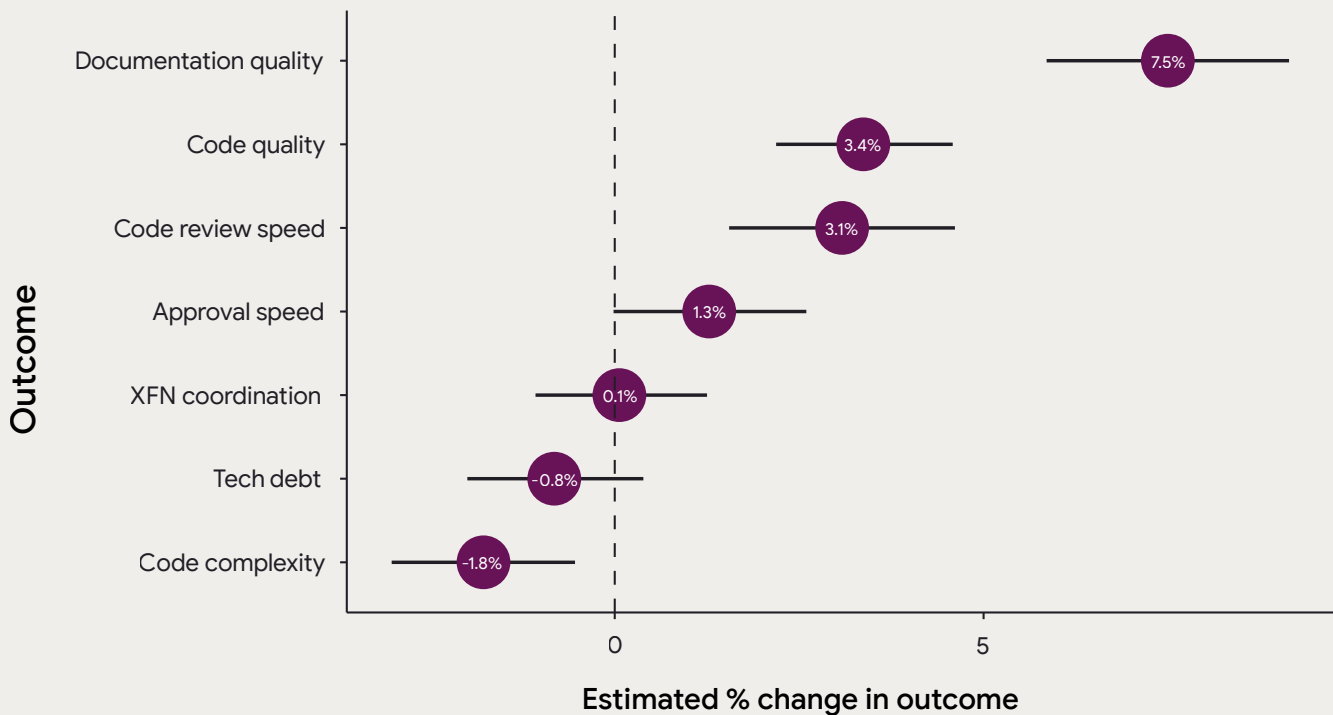
3.4% increase in code quality

3.1% increase in code review speed

1.3% increase in approval speed

1.8% decrease in code complexity

If AI adoption increases by 25%...



Point = estimated value

Error bar = 89% uncertainty interval

Figure 9: Impacts of AI adoption on organizations.

The data presented in the "[Artificial intelligence: Adoption and attitudes](#)" chapter show the most common use of AI is for writing code. 67% of respondents report that AI is helping them improve their code. Here, we see further confirmation of that sentiment. AI seems to improve code quality and reduce code complexity (Figure 9). When combined with some potential refactoring of old code, the high-quality, AI-generated code could lead to an overall better codebase. This codebase might be additionally improved by having better access to quality documentation, which people are using AI to generate (see [Artificial intelligence: Adoption and attitudes](#)).

Better code is easier to review and approve. Combined with AI-assisted code reviews, we can get faster reviews and approvals, a pattern that has clearly emerged in the data (Figure 9).

Of course, faster code reviews and approvals do not equate to better and more thorough code review processes and approval processes. It is possible that we're gaining speed through an over-reliance on AI for assisting in the process or trusting code generated by AI a bit too much. This finding is not at odds with the patterns in Figure 9, but it also not the obvious conclusion.

Further, it isn't obvious whether the quality of the code and the quality of the documentation are improving because AI is generating it or if AI has enhanced

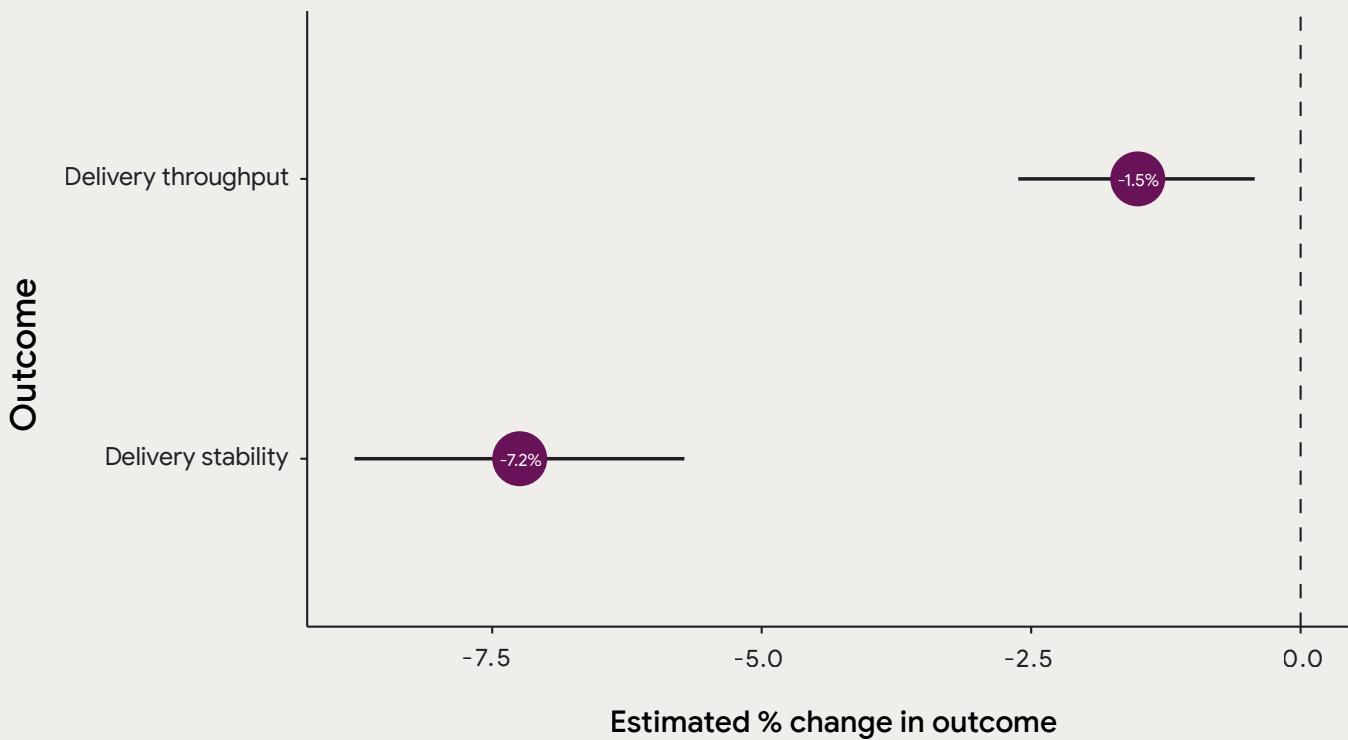
our ability to get value from what would have otherwise been considered low-quality code and documentation. What if the threshold for what we consider quality code and documentation simply moves down a little bit when we're using AI because AI is powerful enough to help us make sense of it? These two ways of understanding these patterns are not mutually exclusive interpretations; both could be contributing to these patterns.

What seems clear in these patterns is that AI helps people get more from the documents they depend on and the codebases they work on. AI also helps reduce costly bottlenecks in the code review and approval process. What isn't obvious is how exactly AI is doing this and if these benefits lead to further downstream benefits, such as software delivery improvements.

AI is hurting delivery performance

For the past few years, we have seen that software delivery throughput and software delivery stability indicators were starting to show some independence from one another. While the traditional association between throughput and stability has persisted, emerging evidence suggests these factors operate with sufficient independence to warrant separate consideration.

If AI adoption increases by 25%...



Point = estimated value

Error bar = 89% uncertainty interval

Figure 10: Impacts of AI adoption on delivery throughput and stability.

Contrary to our expectations, our findings indicate that AI adoption is negatively impacting software delivery performance. We see that the effect on delivery throughput is small, but likely negative (an estimated 1.5% reduction for every 25% increase in AI adoption). The negative impact on delivery stability is larger (an estimated 7.2% reduction for every 25% increase in AI adoption). This data is visualized in Figure 10.

Historically, our research has found that improvements to the software development process, including improved documentation quality, code quality, code review speed, approval speed, and reduced code complexity lead to improvements in software delivery. So, we were surprised to see AI improve these process measures, while seemingly hurting our performance measures of delivery throughput and stability.

Drawing from our prior years' findings, we hypothesize that the fundamental paradigm shift that AI has produced in terms of respondent productivity and code generation speed may have caused the field to forget one of DORA's most basic principles—the importance of small batch sizes. That is, since AI allows respondents to produce a much greater amount of code in the same amount of time, it is possible, even likely, that changelists are growing in size. DORA has consistently shown that larger changes are slower and more prone to creating instability.

Considered together, our data suggest that improving the development process does not automatically improve software delivery—at least not without proper adherence to the basics of successful software delivery, like small batch sizes and robust testing mechanisms.

The beneficial impact that AI has on many important individual and organizational factors that foster the conditions for high software delivery performance is reason for optimism. But, AI does not appear to be a panacea.



High-performing teams and organizations use AI, but products don't seem to benefit.

Here we look at AI's relationship with our most downstream outcomes:

Organizational performance

This is a factor score that accounts for an organization's overall performance, profitability, market share, total customers, operating efficiency, customer satisfaction, quality of products/service, and ability to achieve goals.

Team performance

This is a factor score that accounts for a team's ability to collaborate, innovate, work efficiently, rely on each other, and adapt.

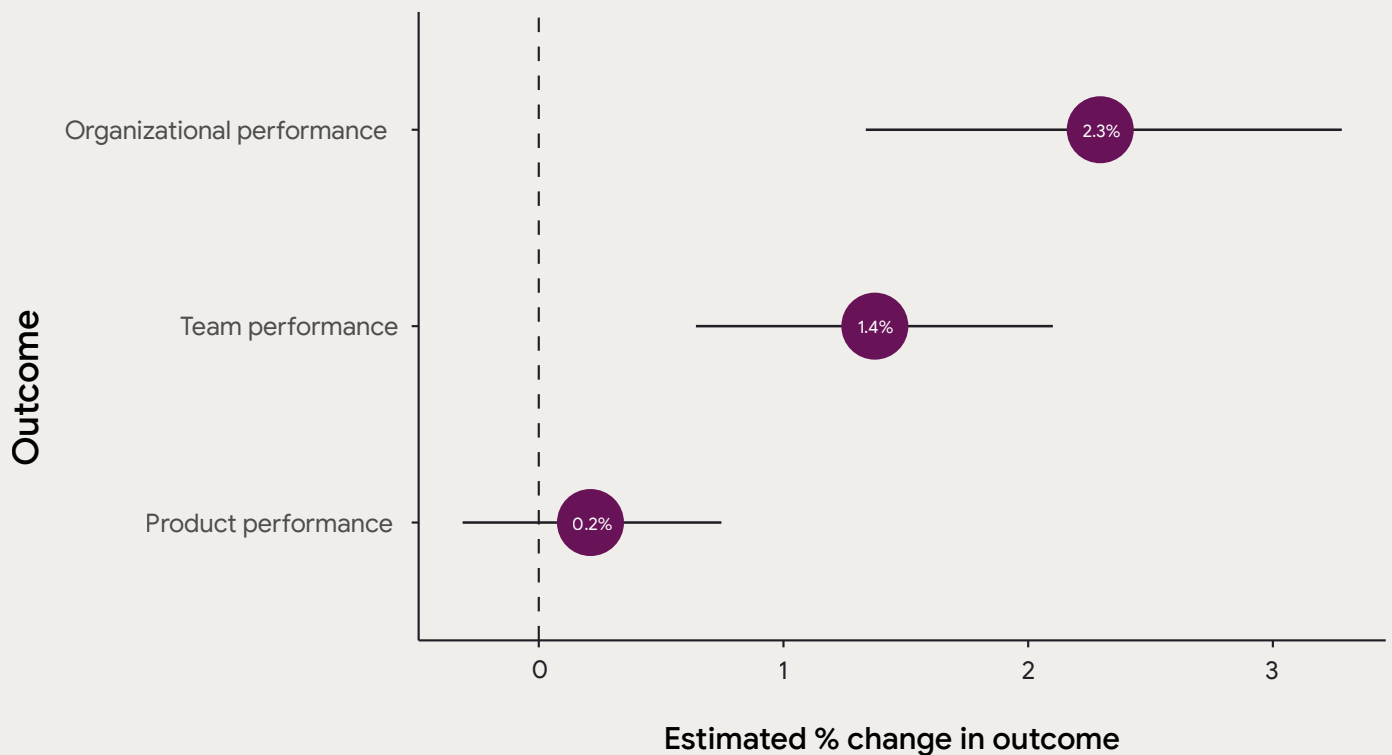
Product performance

This is a factor score that accounts for the usability, functionality, value, availability, performance (for example, latency), and security of a product.

Drawing a connection from these outcomes to an individual adopting AI is difficult and noisy. Sometimes it feels like we're trying to analyze the impact of what you had for lunch today on how well your organization performs this year.

There is a logic to making jumps between the micro-level (for example, an individual) to the macro-level (for example, an organization). We discuss that inferential leap in the [Methodology chapter](#). For now, let's just check out the associations:

If AI adoption increases by 25%...



Point = estimated value

Error bar = 89% uncertainty interval

Figure 11. Impacts of AI adoption on organizational, team, and product performance.

Organization-level performance (an estimated 2.3% increase for every 25% increase in AI adoption) and team-level performance (an estimated 1.4% increase for every 25% increase in AI adoption) seem to benefit from AI adoption (Figure 11). Product performance, however, does not seem to have an obvious association with AI adoption. Now, we can shift to trying to understand what is underlying these effects.

We hypothesize that the factors contributing to strong team and organizational performance differ from those influencing product performance.

Teams and organizations rely heavily on communication, knowledge sharing, decision making, and healthy culture. AI could be alleviating some bottlenecks in those areas, beneficially impacting teams and the organizations.

Product success, however, might involve additional factors. Although good products surely have similar underlying causes as good high performing teams and organizations, there is likely a closer and more direct connection to the development workflow and the software delivery, both of which may be still stabilizing after the introduction of AI.

The unique importance of technical aspects underlying a good product might explain part of it, but there is also an art and empathy underlying a great product. This might be difficult to believe for people who think everything

is a problem to be resolved through computation, but certain elements of product development, such as creativity or user experience design, may still (or forever) heavily rely on human intuition and expertise.

The fact remains that organization, team, and product performance are undeniably interconnected. When looking at bivariate correlations (Pearson), we find product performance has a medium positive correlation with both team performance ($r = 0.56$, 95% confidence interval = 0.51 to 0.60) and organizational performance ($r = 0.47$, 95% confidence interval = 0.41 to 0.53).

These outcomes influence each other reciprocally, creating clear interdependencies. High-performing teams tend to develop better products, but inheriting a subpar product can hinder their success. Similarly, high-performing organizations foster high-performing teams through resources and processes, but organizational struggles can stifle team performance. Therefore, if AI adoption significantly benefits teams and organizations, it's reasonable to expect benefits for products to emerge as well.

The adoption of AI is just starting. Some benefits and detriments may take time to materialize, either due to the inherent nature of AI's impact or the learning curve associated with its effective utilization.

Perhaps the story is simply that we're figuring out how AI can help organizations and teams before we've fully realized its potential for product innovation and development. Figure 12 tries to visualize how this might be unfolding.

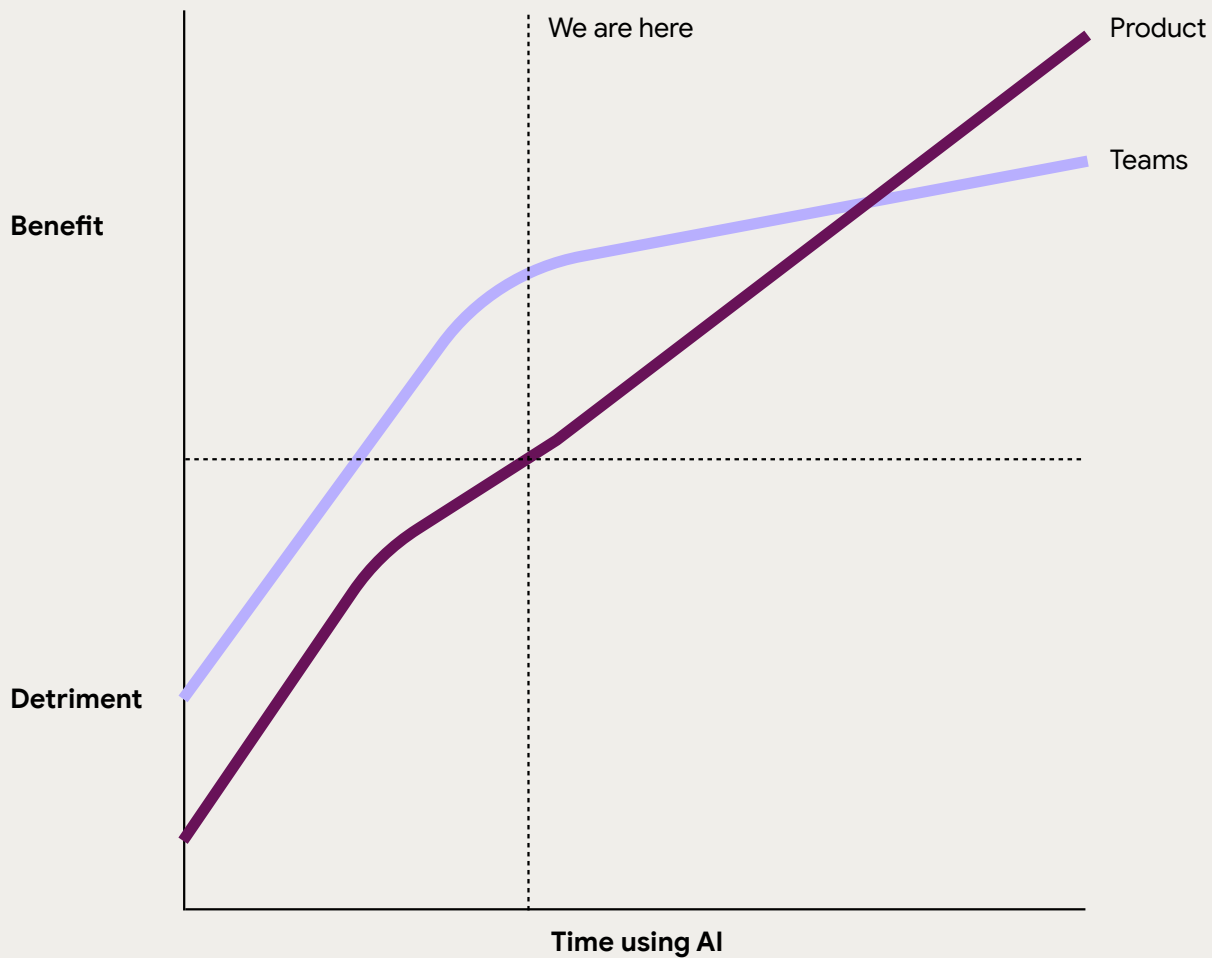


Figure 12: Representations of different learning curves. This is an abstraction for demonstrative purposes. This is not derived from real data.

So now what?

We wanted to understand the potential of AI as it currently stands to help individuals, teams, and organizations. The patterns that are emerging underscore that it isn't all hot air; there really is something happening.

There is clear evidence in favor of adopting AI. That said, it is also abundantly clear that there are plenty of potential roadblocks, growing pains, and ways AI might have deleterious effects.

Here are some thoughts about how to orient your AI adoption strategy:

Adopting AI at scale might not be as easy as pressing play. A measured, transparent, and adaptable strategy has the potential to lead to substantial benefits. This strategy is going to need to be co-developed by leaders, teams, organizations, researchers, and those developing AI.

Leaders and organizations need to find ways to prioritize adoption in the areas that will best support their employees.

Define a clear AI mission and policies to empower your organization and team.

Provide employees with transparent information about your AI mission, goals, and AI adoption plan. By articulating both the overarching vision and specific policies — addressing procedural concerns such as permitted code placement and available tools — you can alleviate apprehension and position AI as a means to help everyone focus on more valuable, fulfilling, and creative work.

Create a culture of continuous learning and experimentation with AI.

Foster an environment that encourages continuous exploration of AI tools by dedicating time for individuals and teams to discover beneficial use cases and granting them autonomy to chart their own course. Build trust with AI technologies through hands-on experience in sandbox or low-risk environments. Consider further mitigating risks by focusing on developing robust test automation. Implement a measurement framework that evaluates AI not by sheer adoption but by meaningful downstream impacts — how it helps employees thrive, benefits those who rely on your products, and unlocks team potential.

Recognize and leverage AI's trade-offs for competitive advantage.

By acknowledging potential drawbacks — such as reduced time spent on valuable work, over-reliance on AI, the potential for benefits gained in one area leading to challenges in another, and impacts on software delivery stability and throughput — you can identify opportunities to avoid pitfalls and positively shape AI's trajectory at your organization, on your team. Developing an understanding not only of how AI can be beneficial, but of how it can be detrimental allows you to expedite learning curves, support exploration, and translate your learnings into action and a real competitive advantage.

It is obvious that there is a lot to be excited about and even more to learn. DORA will stay tuned in and do our best to offer honest, accurate, and useful perspectives, just as it has over the past decade.

1. <https://www.goldmansachs.com/insights/top-of-mind/gen-ai-too-much-spend-too-little-benefit>

2. <https://www.goldmansachs.com/insights/articles/AI-poised-to-drive-160-increase-in-power-demand>

3. <https://www.washington.edu/news/2023/07/27/how-much-energy-does-chatgpt-use/>

4. <https://www.gatesnotes.com/The-Age-of-AI-Has-Begun>

5. <https://www.businessinsider.com/ai-chatgpt-homework-cheating-machine-sam-altman-openai-2024-8>

6. <https://www.safe.ai/work/statement-on-ai-risk>

7. <https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>

8. https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality

9. <https://www.nytimes.com/2024/04/15/technology/ai-models-measurement.html>

10. <https://dora.dev/capabilities>

11. we should be clear that this isn't a unique approach, but it is a somewhat unique approach for this space

12. (P[N]), for example (P1), indicates pseudonym of interview participants.

Platform engineering



Introduction

Platform engineering is an emerging engineering discipline that has been gaining interest and momentum across the industry. Industry leaders such as Spotify and Netflix, and books such as *Team Topologies*¹ have helped excite audiences.

Platform engineering is a sociotechnical discipline where engineers focus on the intersection of social interactions between different teams and the technical aspects of automation, self-service, and repeatability of processes. The concepts behind platform engineering have been studied for many years, including by DORA.

Generally, our research is focused on how we deliver a software to external users, whereas the output of platform teams is typically an inwardly-focused set of APIs, tools, and services designed to support the software development and operations lifecycle.

In platform engineering, a lot of energy and focus is spent on improving the developer experience by building golden paths, which are highly-automated, self-service workflows that users of the platform use when interacting with resources required to deliver and operate applications. Their purpose is to abstract away the complexities of building and delivering software such that the developer only needs to worry about their code.

Some examples of the tasks automated through golden paths include new application provisioning, database provisioning, schema management, test execution, build and deployment infrastructure provisioning, and DNS management.

Concepts in platform engineering such as moving a capability down (sometimes called “shifting down”)² into a shared system can seem counter to approaches like 'you build it, you run it.' However, we think of platform engineering as a method to scale the adoption of these practices across an organization because once a capability is in the platform, teams essentially get it for free through adoption of the platform.

For example, if the platform has the capability to execute unit tests and report back results directly to development teams, but without that team needing to build and manage the testing execution environment, then the continuous integration platform feature enables teams to focus on writing high-quality tests. In this example, the continuous integration feature can scale across the larger organization and make it easier for multiple teams to improve their capabilities with continuous testing³ and test automation.⁴

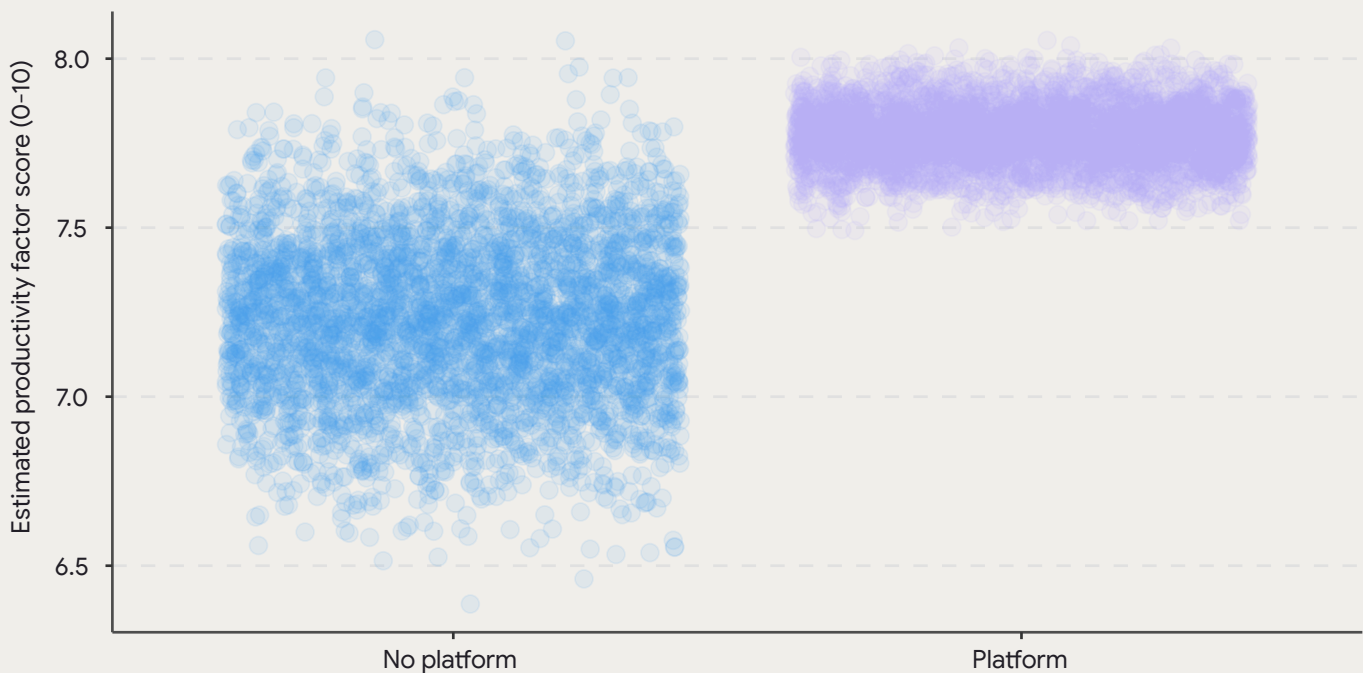


A key factor in the success is to approach platform engineering with user-centeredness (users in the context of an internal developer platform are developers), developer independence, and a product mindset. This isn't too surprising given that user centricity was identified as a key factor in improved organizational performance this year and in previous years.⁵ Without a user-centered approach, the platform will be more a hindrance rather than an aid.

In this year's report, we sought to test the relationship between platforms and software delivery and operational performance. We found some positive results. Internal developer platform users had 8% higher levels of individual

productivity and 10% higher levels of team performance. Additionally, an organization's software delivery and operations performance increases 6% when using a platform. However, these gains do not come without some drawbacks. Throughput and change stability saw decreases of 8% and 14%, respectively, which was a surprising result.

In the next sections we'll dig deeper into the numbers, nuances, and some surprising data that this survey revealed. Whether your platform engineering initiative is just starting or has been underway for many years, application of the key findings can help your platform be more successful.



Each dot is one of 8000 estimates of the most plausible mean productivity score
Figure 13: Productivity factor for individuals when using or not using an internal developer platform.

The promise of platform engineering

Internal developer platforms are garnering interest from large sections of the software developer and IT industry given the potential efficiency and productivity gains that could be achieved through the practice. For this year's survey, we left the definition of an internal developer platform quite broad⁶ and found that 89% of respondents are using an internal developer platform. The interaction models are very diverse across that population.

These data points align with the broad level of industry interest in platform engineering and the emerging nature of the field.

Overall, the impact of a platform is positive, individuals were 8% more productive and teams performed 10% better when using an internal developer platform.

Beyond productivity, we also see gains when a platform is used in an organization's overall performance, with an increase of 6%. On the whole, the organization is able to quickly deliver software, meet user needs, and drive business value due to the platform.

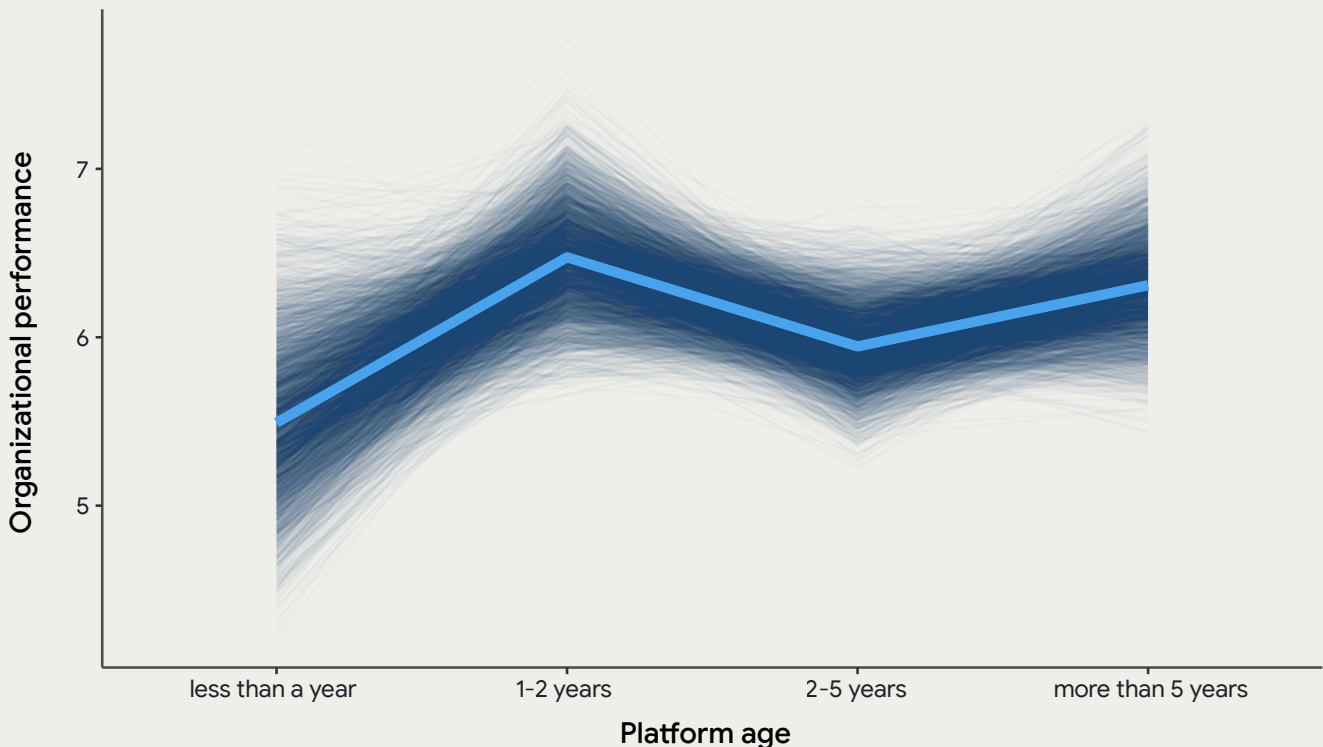


Figure 14: Organization performance change when using an internal developer platform vs the age of the platform.

When taking into account the age of the platform with productivity, we see initial performance gains at the onset of a platform engineering initiative, followed by decrease and recovery as the platform ages and matures. This pattern is typical of transformation initiatives that experience early gains but encounter challenges once those have been realized.

In the long run, productivity gains are maintained showing the overall potential of an internal developer platform's role in the software delivery and operational processes.



Key finding - impact of developer independence

Developer independence had a significant impact on the level of productivity at both the individual and team levels when delivering software using an internal developer platform. Developer independence is defined as “developers' ability to perform their tasks for the entire application lifecycle, without relying on an enabling team.”

At both the team and individual level we see a 5% improvement in productivity when users of the platform are able to complete their tasks without involving an enabling team. This finding points back to one of the key principles of platform engineering, focusing on enabling self-service workflows.

For platform teams, this is key because it points to an important part of the platform engineering process, collecting feedback from users. Survey responses did not indicate which forms of feedback are most effective, but common methods are informal conversations and issue trackers, followed by ongoing co-development, surveys, telemetry, and interviews.

All of these methods can be effective at understanding whether or not users are able to complete their tasks independently. The survey data also showed that not collecting feedback on the platform has a negative impact.

Secondary finding - impact of a dedicated platform team

Interestingly, the impact on productivity of having a dedicated platform team was negligible for individuals. However, it resulted in a 6% gain in productivity at the team level. This finding is surprising because of its uneven impact, suggesting that having a dedicated platform team is useful to individuals, but the dedicated platform team is more impactful for teams overall.

Since teams have multiple developers with different responsibilities and skills, they naturally have a more diverse set of tasks when compared to an individual engineer. It is possible that having a dedicated platform engineering team allows the platform to be more supportive of the diversity in tasks represented by a team.

Overall, the impact of having an internal developer platform has a positive impact on productivity.

The key factors are:

A user-centered approach that enables developer independence through self-service and workflows that can be completed autonomously. Recall that in the context of the platform, users are internal engineering and development teams.

As with other transformations, the “j-curve” also applies to platform engineering, so productivity gains will stabilize through continuous improvement.

The unexpected downside

While platform engineering presents some definite upsides, in terms of teams and individuals feeling more productive and improvements in organizational performance, platform engineering had an unexpected downside: We also found that throughput and change stability decreased.

Unexpectedly, we discovered a very interesting linkage between change instability and burnout.

Throughput

In the case of throughput, we saw approximately an 8% decrease when compared to those who don't use a platform. We have hypotheses about what might be the underlying cause.

First, the added machinery that changes need to pass through before getting deployed to production decreases the overall throughput of changes. In general, when an internal developer platform is being used to build and deliver software, there is usually an increase in the number of "handoffs" between systems and implicitly teams.

For example, when code is committed to source control, it is automatically picked up by different systems for testing, security checks, deployment, and monitoring.

Each of these handoffs is an opportunity for time to be introduced into the overall process resulting in a decrease in throughput, but a net increase in ability to get work done.

Second, for respondents who reported, they are required to "exclusively use the platform to perform tasks for the entire app lifecycle," there was a 6% decrease in throughput. While not a definitive connection, it could also be related to the first hypothesis.

If the systems and tools involved in developing and releasing software increases with the presence of a platform, being required to use the platform when it might not be fit for purpose or naturally-increasing latency in the process could account for the relationship between exclusivity and decrease in productivity.

To counter this it is important to be user-centered and work toward user independence in your platform engineering initiatives.

Change instability and burnout

When considering the stability of the changes to applications being developed and operated when using an internal developer platform, we observed a surprising 14% decrease in change stability. This indicates that the change failure rate and rate of rework are significantly increased when a platform is being used.

Even more interesting, in the results we discovered that instability in combination with a platform is linked to higher levels of burnout. That isn't to say that platforms lead to burnout, but the combination of instability and platforms are particularly troublesome when it comes to burnout. Similar to the decrease in throughput, we aren't entirely sure why the change in burnout occurs, but we have some hypotheses.

First, the platform enables developers and teams to push changes with a higher degree of confidence that if the change is bad, it can be quickly remediated. In this instance the higher level of instability isn't necessarily a bad thing since the platform is empowering teams to experiment and deliver changes, which results in an increased level of change failure and rework.

A second idea is that the platform isn't effective at ensuring the quality of changes and/or deployments to production.

It could also be that the platform provides an automated testing capability that exercises whatever tests are included in the application. Yet application teams aren't fully using that capability by prioritizing throughput over quality and not improving their tests. In either scenario, bad changes are actually making it through the process, resulting in rework.

A third possibility is that teams with a high level of change instability and burnout tend to create platforms in an effort to improve stability and reduce burnout. This makes sense because platform engineering is often viewed as a practice which reduces burnout and increases the ability to consistently ship smaller changes. With this hypothesis, platform engineering is symptomatic of an organization with burnout and change instability.

In the first two scenarios, the rework allowed by the platform could be seen as burdensome which could also be increasing burnout. In particular, the second scenario where the platform is enabling bad changes would contribute more to burnout, but in both scenarios the team or individual could still feel productive because of their ability to push changes and features. In the third scenario, change instability and burnout are predictive of a platform engineering initiative and the platform is seen as a solution to those challenges.

Balancing the Trade-offs

While platform engineering is no panacea, it has the potential to be a powerful discipline when it comes to the overall software development and operations process. As with any discipline, platform engineering has benefits and drawbacks.

Based on our research, there are a couple actions you can take to balance the trade-offs when embarking on a platform engineering initiative. Doing so will help your organization achieve the benefits of platform engineering while being able to monitor and manage any potential downsides.

First, prioritize platform functionality that enables developer independence and self-service capabilities. When doing this, pay attention to the balance between exclusively requiring the platform to be used for all aspects of the application lifecycle, which could hinder developer independence.

As good practice, a platform should provide methods for users of a platform to break out of the tools and automations provided in the platform, which contributes to independence, however, it comes at the cost of complexity. This trade-off can be mitigated with a dedicated platform team that actively collaborates with and collects feedback from users of the platform.

Collaboration and feedback improve the user-centeredness of the platform initiative and will contribute to the long-term success of the platform. As we saw in the data, there are many different methods used to collect feedback, so employ more than one approach to maximize feedback collection.

Second, carefully monitor the instability of your application changes and try to understand whether the instability being experienced is intentional or not. Platforms have the potential to unlock experimentation in the terms of instability, increase productivity, and improve performance at scale.

However, that same instability can also have the potential to do this at the cost of instability and burnout, so it needs to be carefully monitored and accounted for throughout the platform engineering journey. When doing so it is important to understand your appetite for instability. Using service level objectives (SLOs) and error budgets from site reliability engineering (SRE) can help you gauge your risk tolerance and effectiveness of the platform in safely enabling experimentation.

Internal developer platforms put a lot of emphasis on the developer experience, however, there are many other teams (including database administrators, security, and operations) who are required to effectively deliver and operate software.

In your platform engineering initiatives, foster a culture of user-centeredness and continuous improvement across all teams and aligned with the organization's goals.

Doing so will align the platform's features, services, and APIs to best serve individual and team needs as they work to deliver software and business value.



1. Skelton, Matthew and Pais, Manuel. 2019. Team Topologies: Organizing Business and Technology Teams for Fast Flow. IT Revolution Press. <https://teamtopologies.com/>
2. <https://cloud.google.com/blog/products/application-development/richard-seroter-on-shifting-down-vs-shifting-left>
3. <https://dora.dev/capabilities/continuous-integration/>
4. <https://dora.dev/capabilities/test-automation/>
5. <https://dora.dev/research/2023/>, <https://dora.dev/research/2016/>
6. <https://dora.dev/research/2024/questions/#platform-engineering>

Developer experience



Takeaways

Software doesn't build itself. Even when assisted by AI, people build software, and their experiences at work are a foundational component of successful organizations.

In this year's report, we again found that alignment between what developers build and what users need allows employees and organizations to thrive. Developers are more productive, less prone to experiencing burnout, and more likely to build high quality products when they build software with a user-centered mindset.

Ultimately, software is built for people, so it's the organization's responsibility to foster environments that help developers focus on building software that will improve the user experience. We also find that stable environments, where priorities are not constantly shifting, lead to small but meaningful increases in productivity and important, meaningful decreases in employee burnout.

Environmental factors have substantial consequences in the quality of the products developed, and the overall experience of developers whose job is to build those products.

Put the user first, and (almost) everything else falls into place

We think that the job of a developer is pretty cool. Developers are at the forefront of technological advancements and help shape how we live, work, and interact with the world.

Their jobs are fundamentally tied to people—the users of the software and applications they create. Yet developers often work in environments that prioritize features and innovation. There's less emphasis on figuring out whether these features provide value to the people who use the products they make.

Here we provide compelling evidence showing that an approach to software development that prioritizes the end user positively impacts employees and organizations alike.

This year, we asked questions focused on understanding whether developers:

1. Incorporate user feedback to revisit and reprioritize features
2. Know what users want to accomplish with a specific application/service
3. Believe focusing on the user is key to the success of the business
4. Believe the user experience is a top business priority





Our findings and what they mean

Our data strongly suggests that organizations that see users' needs and challenges as a guiding light make better products.

We find that focusing on the user increases productivity and job satisfaction, while reducing the risk of burnout.

Importantly, these benefits extend beyond the individual employee to the organization. In previous years, we've highlighted that high performing organizations deliver software quickly and reliably. The implication is that software-delivery performance is a requirement for success.

However, our data indicates there's another path that leads to success:

Developers and their employers, and organizations in general, can create a user-centered approach to software development.

We find that when organizations know and understand users' needs, stability and throughput of software delivery are not a requirement for product quality. Product quality will be high as long as the user experience is at the forefront.

When organizations don't focus on the user, don't incorporate user feedback into their development process, doubling down on stable and fast delivery is the only path to product quality (see Figure 15).

We understand the inclination that some organizations might have to focus on creating features and innovating on technologies. At face value, this approach makes sense. After all, developers most certainly know the ins and outs of the technology much better than their average user.

However, developing software based on assumptions about the user experience increases the likelihood of developers building features that are perhaps shiny but hardly used.¹

When organizations and employees understand how their users experience the world, they increase the likelihood of building features that address the real needs of their users. Addressing real user needs increases the chances of those features being actually used.

Focus on building for your user and you will create delightful products.

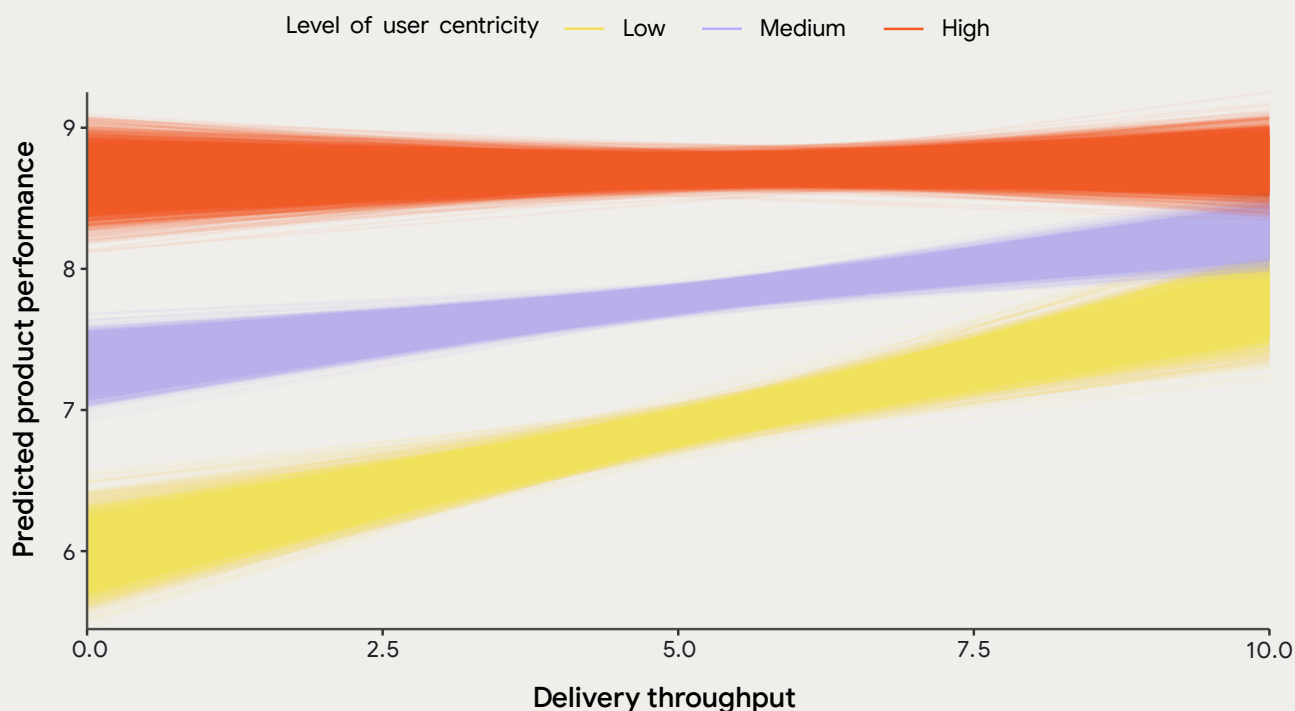


Figure 15: Product performance and delivery throughput across 3 levels of user centrality

Why is a user-centered approach to software development such a powerful philosophy and practice?

Academic research shows that deriving a sense of purpose from work benefits employees and organizations.^{2,3}

For example, a recent survey showed that 93% of workers reported that it's important to have a job where they feel the work they do is meaningful.⁴ In a similar vein, another survey found that on average, respondents were willing to relinquish 23% of their entire future earnings if it meant they could have a job that was always meaningful.⁵

That's an eye-popping trade-off employees are willing to make. It tells us something about what motivates people, and that people want to spend their time doing something that matters.

“It would be grand if everybody could work at a company that affects individuals outside of the company, or [in] your local community in a positive way. That's not always the case. That's not always possible. A lot of the grand vision of autonomous driving is that it is going to enable people that can drive [to] sleep while they're on a motorway. That's not why I'm here. I want to help people that can't drive to be able to get about, wherever they want, have the freedom to do whatever they want to do.” (P2)⁶

Provides a clear sense of direction:

A user-centered approach to software development can fundamentally alter how developers view their work. Instead of shipping arbitrary features and guessing whether users might use them, developers can rely on user feedback to help them prioritize what to build.

This approach gives developers confidence that the features they are working on have a reason for being. Suddenly, their work has meaning: to ensure people have a superb experience when using their products and services. There's no longer a disconnect between the software that's developed and the world in which it lives.

Developers can see the direct impact of their work through the software they create.

“We are, as a company, under pressure to deliver. So, all of these, like, nice shiny things, or discussion points about how you want to improve, it's kind of, like, with the recent change in how we're structured, we're focusing on delivery, not quality, and for me, personally, that's kind of a big bugbear.” (P9)

Increases cross-functional collaborations:

Even the most talented developer doesn't build software on their own. Building high-quality products takes the collaboration of many people often with different yet complementary talents.

A user-centered approach to development allows developers to engage in cross-functional collaborations across the organization. In doing so, their responsibilities extend beyond simply shipping software. They are now part of a team driven to create incredible experiences for the people who use them.

This approach to software development can help developers break out of silos, seek alignment, foster teamwork, and create opportunities to learn more from others. Problem solving takes a different shape. It's not just about how to solve technical problems, but how to do so in ways that serve the user best.

This approach can help increase employee engagement and create an even more intellectually-stimulating environment that can stave off the feelings of stagnation that are associated with burnout.

What can organizations do?

Based on our findings, we recommend organizations invest time and resources in getting to know their users. Focus on understanding who you are building for, and the challenges they experience. We strongly believe this is a worthy investment.

Resist the temptation to make assumptions about your users. Observe them in their environments, ask them questions, and be humble enough to pivot based on what they tell you. In doing so, developers will be more productive and be less prone to burnout while delivering higher quality products.

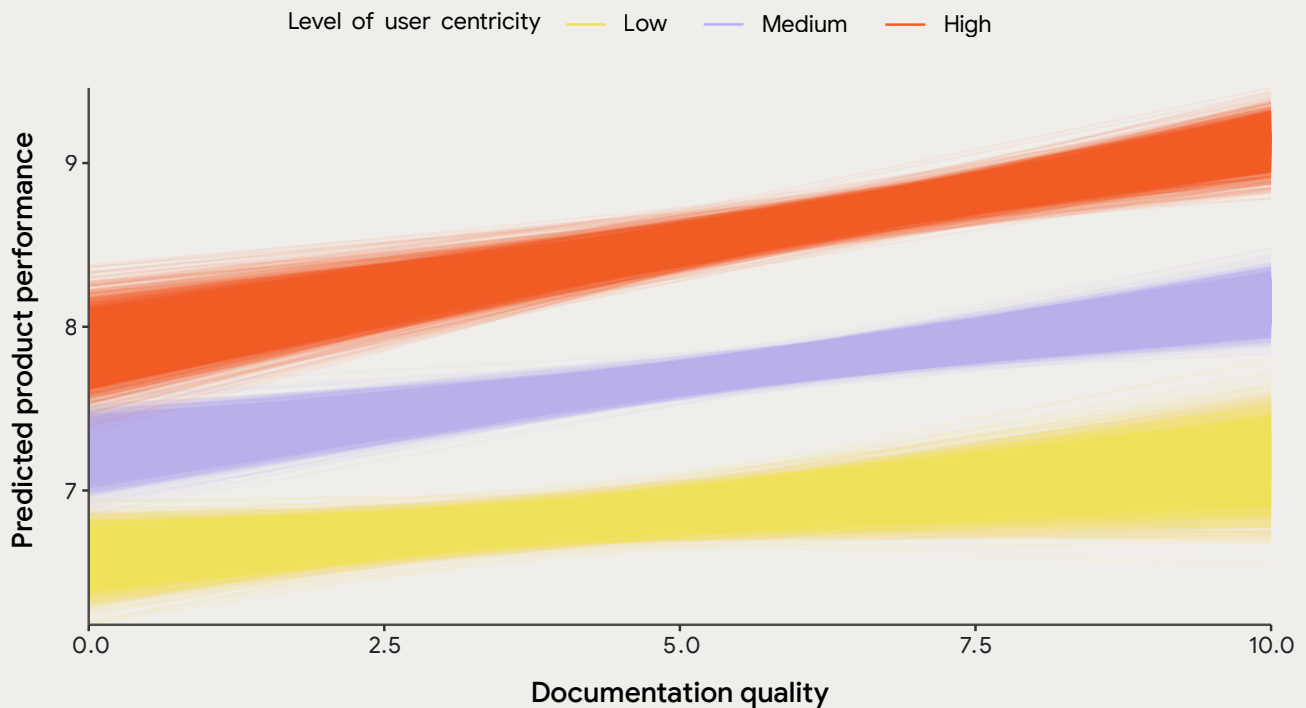
The combination of good docs and a user-centered approach to software development is a powerful one.

Teams that focus on the user see an increase in product performance. When this focus on the user is combined with an environment of quality internal documentation, this increase in product performance is amplified (see Figure 16). This finding is similar to the behavior that we see where documentation amplifies a technical capability's impact on organizational performance.⁷

Documentation helps propagate user signals and feedback across the team and into the product itself.

We see that internal documentation doesn't meaningfully affect predicted product performance without user signals. However, if a team has a high quality internal documentation then user signals included in it will have a higher impact on product performance.

We started to look at documentation in 2021, and every year we continue to find extensive impact of quality documentation. This year's findings adds internal documentation's impact on predicted product performance to the list.



The graph is a composite of 12000 lines from simulations trying to estimate the most plausible pattern

Figure 16: Product performance and documentation quality across 3 levels of user centrality

Culture of documentation

The Agile manifesto advocates for “working software over comprehensive documentation”.⁷ We continue to find, however, that quality documentation is a key component of working software.

“Comprehensive documentation” may be a phrase standing in for unhealthy practices, which might include documentation. Problematic documentation includes documentation that is created only for bureaucratic purposes, or to paper over mistrust between management and employees. An unhealthy documentation culture can also include writing documentation, but not maintaining or consolidating the documentation.

In these cases, our measure of quality documentation would likely score low. This type of content is written for the wrong audience so doesn’t perform as well when you try to use it while doing your work. And too much documentation can be as problematic as not enough.

Our measure of quality documentation includes attributes like findability and reliability of the documentation. Remember, for internal documentation, the primary audience is your colleagues or even your future self trying to accomplish specific tasks.⁸ Teams with a healthy documentation culture have a focus on serving these readers. This is another way that focusing on your users matters.

You can create a healthy culture of documentation on your own teams by following the practices we’ve identified to create quality documentation, such as:

Documenting critical use cases.

Taking training in technical writing.

Defining ownership and processes to update the documentation.

Distributing documentation work within the team.

Maintaining documentation as part of the software development lifecycle.

Deleting out-of-date or redundant documentation.

Recognizing documentation work in performance reviews and promotions.

The perils of ever-shifting priorities

We all know the feeling. You've spent the last few months working on a new feature. You know it's the right thing to build for your users, you are focused and motivated. Suddenly, or seemingly so, the leadership team decides to change the organization's priorities. Now it's unclear whether your project will be paused, scrapped, Frankensteined, or mutated.

This common experience can have profound implications for employees and organizations. Here we examine what happens when organizations constantly shift their priorities.

Our findings and what they mean

Overall, our findings show small but meaningful decreases in productivity and substantial increases in burnout when organizations have unstable priorities.

Our data indicates it is challenging to mitigate this increase in burnout. We examined whether having strong leaders, good internal documents, and a user-centered approach to software development can help counteract the effect of shifting priorities on burnout.

The answer is: They can't. An organization can have all these positive traits and, if priorities are unstable, employees will still be at risk of experiencing burnout.

Why are unstable organizational priorities bad for employees' well-being?

We hypothesize that unstable organizational priorities increase employee burnout by creating unclear expectations, decreasing employees' sense of control, and increasing the size of their workloads.

To be clear, we believe that the problem is not with changing priorities themselves. Business goals and product direction shift all the time. It can be good for organizational priorities to be malleable.

We believe it is the frequency with which priorities change that has a negative impact on employees' well-being. The uncertainty that accompanies unstable priorities implies something chronic about the frequency with which priorities change.

Decades of academic research have shown the detrimental effects of chronic stress on health and well-being.⁹ We see parallels between research on chronic stress and our findings. Chronic instability increases uncertainty and decreases perceived control. This combination is an excellent recipe for burnout.

What happens when priorities stabilize?

Our findings here are a little puzzling. We find that when priorities are stabilized, software delivery performance declines. It becomes slow and less stable in its delivery.

We hypothesize that this might be because organizations with stable priorities might have products and services that are generally in good shape so changes are made less frequently. It is also possible that stability of priorities leads to shipping less and in larger batches than recommended.

Nevertheless, we find this to be an unexpected finding. Why do you think stabilizing organizational priorities decreases the speed and stability of software delivery?

Building AI for end users creates stability in priorities, but not stability in delivery.

Incorporating AI-powered experiences for end users stabilizes organizational priorities. This sounds like a flashy endorsement for AI. However, we do not interpret this finding as telling us something meaningful about AI itself.

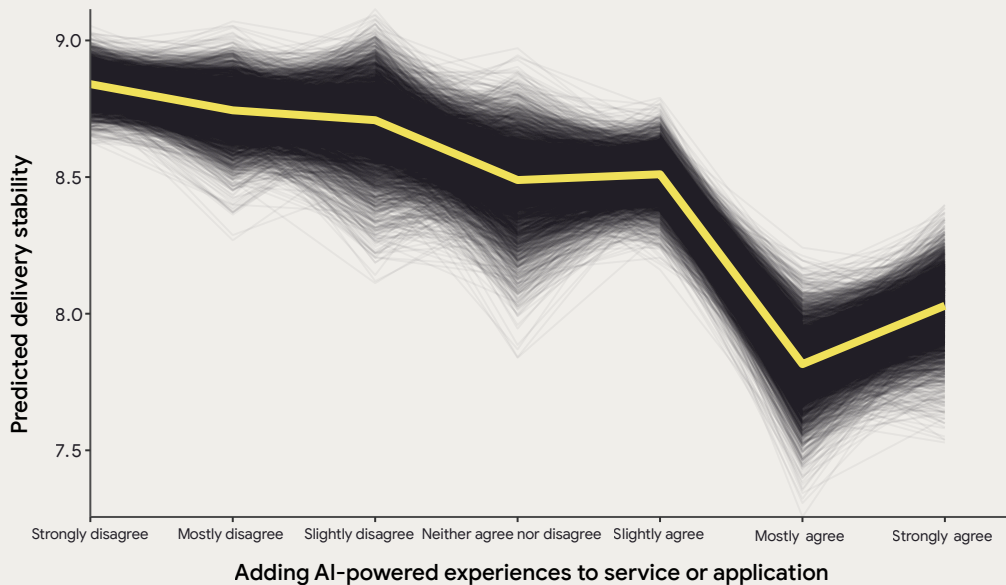
Instead, we believe that shifting efforts towards building AI provides clarity and a northstar for organizations to follow. This clarity, and not AI, is what leads to a stabilization of organizational priorities.

This is worth highlighting because it tells us something about what happens to organizations when new technologies emerge. New technologies bring change and organizations need time to adapt.

This period likely leads to a destabilization of priorities as leaders try to figure the best move for the organization. As the dust settles, and organizations clarify their next steps, priorities begin to stabilize.

Priorities stabilizing, however, doesn't immediately translate into the software delivery process stabilizing. Our analyses show that a shift to adding AI-powered experiences into your service or application comes with challenges and growing pains.

We find that teams that have shifted have a significant 10% decrease in software delivery stability relative to teams who have not. Here is a visualization depicting the challenge.



*Each line is one of 4000 simulations trying to estimate the most plausible pattern

Figure 17: Software delivery stability as a function of adding AI-powered experiences to service or application

What can organizations do?

The answer, while easy, might not be so simple. Based on our findings, we recommend organizations focus on stabilizing their priorities. This is one sure way to counteract the negative effects of unstable priorities on employee burnout.

Our findings show the negative effects of unstable priorities are resistant to having good leaders, good documentation, and a user-centered approach to software development. This leads us to believe that, aside from creating stability, there's not much organizations can do to avoid burnout aside from finding ways to (1) stabilize priorities and (2) shield employees from having their day-to-day be impacted by the constant shift in priorities.

-
1. <https://www.nngroup.com/articles/bridging-the-designer-user-gap/>
 2. <https://executiveeducation.wharton.upenn.edu/thought-leadership/wharton-at-work/2024/03/creating-meaning-at-work/>
 3. <https://www.apa.org/pubs/reports/work-in-america/2023-workplace-health-well-being>
 4. <https://bigthink.com/the-present/harvard-business-review-americans-meaningful-work/>
 5. <https://hbr.org/2018/11/9-out-of-10-people-are-willing-to-earn-less-money-to-do-more-meaningful-work>
 6. (P[N]), for example (P1), indicates pseudonym of interview participants.
 7. <https://cloud.google.com/blog/products/devops-sre/deep-dive-into-2022-state-of-devops-report-on-documentation> and [Accelerate State of DevOps Report 2023 - https://dora.dev/research/2023/dora-report](https://dora.dev/research/2023/dora-report)
 8. <https://agilemanifesto.org/>
 9. Other audiences exist, such as management, regulators, or auditors.
 10. Cohen S, Janicki-Deverts D, Miller GE. Psychological Stress and Disease. *JAMA*. 2007;298(14):1685–1687.doi:10.1001/jama.298.14.1685

Leading transformations

A lot needs to be in place for transformation to work. This year, we've found high-performing teams are ones that prioritize stability, focus on their users, have good leaders, and craft quality documentation. Our research points to some useful paths in helping you plot a course towards successful transformation.

We have found the key to success is to approach transformation from a mindset of continuous improvement. High performers in our study understand the variables holding them back, and methodically and continuously improve using the DORA metrics as a baseline. While long-term success requires excellence in all pillars, a decade of DORA research has pointed us to four specific, impactful ways to get started on driving transformation in your own organization.



Transformational leadership

Transformational leadership is a model in which leaders inspire and motivate employees to achieve higher performance by appealing to their values and sense of purpose, facilitating wide-scale organizational change.

These leaders encourage their teams to work towards a common goal through the following dimensions:¹

Vision	They have a clear vision of where their team and the organization are going.
Inspirational communication	They say positive things about the team; make employees proud to be a part of their organization; encourage people to see changing conditions as situations full of opportunities.
Intellectual stimulation	They challenge team members to think about old problems in new ways and to rethink some of their basic assumptions about their work.
Supportive leadership	They consider others' personal feelings before acting; behave in a manner which is thoughtful of others' personal needs.
Personal recognition	They commend team members when they do a better-than-average job; acknowledge improvement in quality of team members' work.

This year, we saw that transformational leadership leads to a boost in employee productivity. We see that increasing transformational leadership by 25% leads to a 9% increase in employee productivity.

Transformational leadership can help improve more than just productivity. Having good leaders can also lead to:

- A decrease in employee burnout
- An increase in job satisfaction
- An increase in team performance
- An improved product performance
- An improved organizational performance

Our research found a statistically significant relationship between the above qualities of leadership and IT performance in 2017. High-performing teams had leaders with strong scores across all five characteristics and low-performing teams had the lowest scores. Additionally, we saw that there's a strong correlation between transformative leadership and Employee Net Promoter Score (eNPS), the likelihood to recommend working at a company.

That said, transformative leadership by itself does not lead to high performance, but should be seen as an enabler.

Transformative leadership plays a key role in enabling the adoption of technical and product-management capabilities and practices. This is enabled by (1) delegating authority and autonomy to teams; (2) providing them the metrics and business intelligence needed to solve problems; and (3) creating incentive structures around value delivery as opposed to feature delivery.

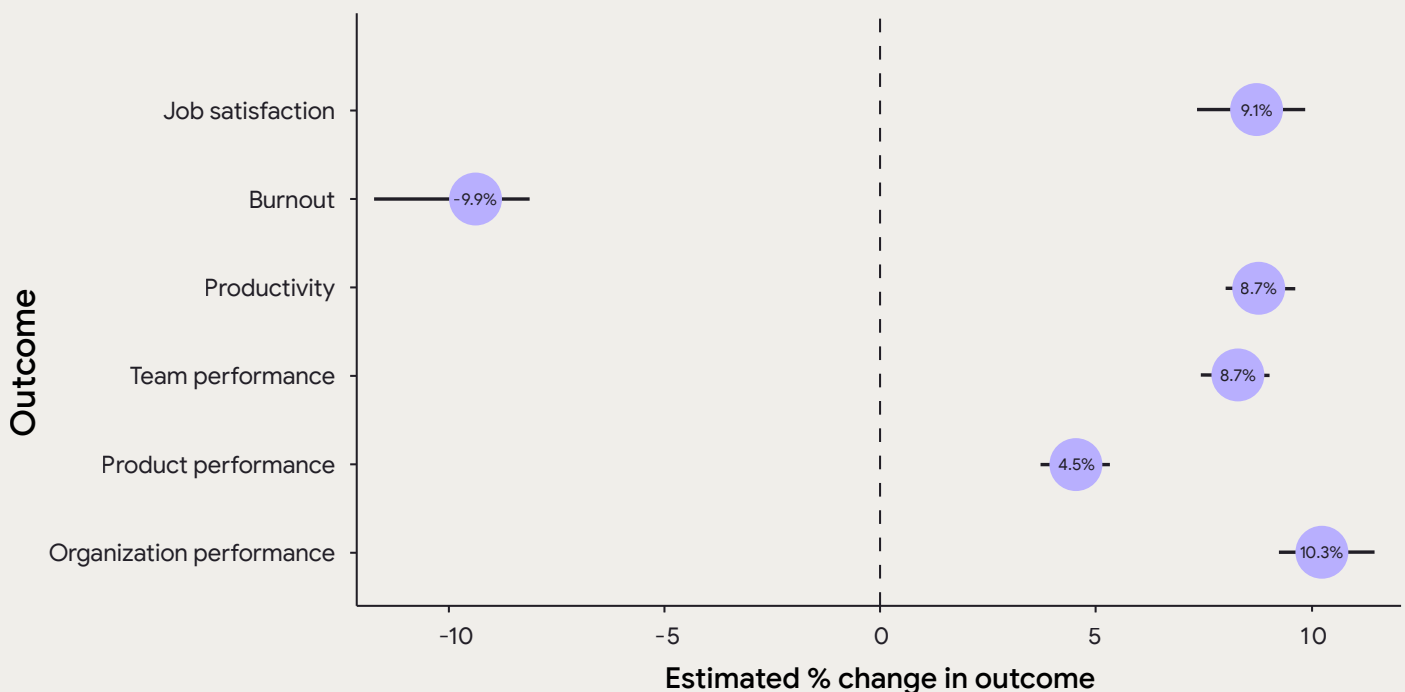
Transformation takes time and requires tools. Resources must be allocated by leadership specifically for the task of improvement. Good leaders play a key role in providing teams with the time and funding necessary to improve. Engineers should not be expected to learn new things and automate on their off time, this should be baked into their schedule.

Our research has helped to flip the narrative of IT being a cost-center to IT being an investment that drives business success. In 2020, we wrote the ROI of DevOps whitepaper,² which contains calculations you can use to help articulate potential value created by investing in IT improvement.

Monetary return is only one of the returns you can expect from this investment. Our research in 2015 showed that, “organizational investment in DevOps is strongly correlated with organizational culture; the ability of development, operations, and infosec

teams to achieve win-win outcomes; lower levels of burnout; more effective leadership; and effective implementation of both continuous delivery and lean management practices.”³ We recommend dedicating a certain amount of capacity specifically for improvement.

If transformational leadership increases by 25%...



Point = estimated value

Error bar = 89% uncertainty interval

Figure 18: Impacts of transformational leadership on various outcomes.

Be relentlessly user-centric

This year's research shows that organizations with strong leaders and a focus on building software that addresses user needs leads to the development of better products: It's a powerful combination. When the user is at the center of software development, leaders have a clear vision to articulate.

The ultimate goal is for users to love the products we create. As we discuss in the [Developer experience](#) chapter, focusing on the user gives product capabilities a reason to exist. Developers can confidently build these features knowing they'll help improve the user experience.

We see that teams that have a deep desire to understand and align to their users' needs and the mechanisms to collect, track, and respond to user feedback have the highest levels of organizational performance. In fact, organizations can be successful even without high levels of software velocity and stability, as long as they are user-focused. In 2023 we saw user-centered teams have a 40% higher level of organizational performance compared to those that did not,⁴ and in 2016 we also saw that user-centered teams had better organizational performance.

This year's research echoes previous findings. Teams that focus on the user make better products.

Not only do products improve, but employees are more satisfied with their jobs and less likely to experience burnout.

Fast, stable software delivery enables organizations more frequent opportunities to experiment and learn. Ideally, these experiments and iterations are based on user feedback. Fast and stable software delivery allows you to experiment, better understand user needs, and quickly respond if those needs are not being met.

Having speed and stability baked into your delivery also allows you to more easily adjust to market changes or competition.

It is important to remember that your internal developers are also users. Internal Developer Platforms (IDPs) are a way your organization can deliver value to developers that in turn deliver value to external users or other internal users.

Our research shows that successful IDPs are developed as a product and focus on user centricity to deliver an experience that allows developers to work independently. An IDP deployed in this way leads to higher individual productivity, higher team productivity, and higher organizational performance.

Become a data-informed organization

The ability to visualize your progress toward success is critical. Over the last 10 years we have made the case for becoming a data-informed organization. DORA's four key metrics⁵ have become a global standard for measuring software delivery performance, but this is only part of the story. We have identified more than 30 capabilities and processes⁶ that can be used to drive organizational improvement.

The value in the metrics lies in their ability to tell you if you are improving. The four key metrics should be used at the application and service levels, and not at the organization or line-of-business level. The metrics should be used to visualize your efforts in continuous improvement and not to compare teams — and certainly not to compare individuals.

The metrics should also not be used as a maturity model for your application or service teams. Being a low, medium, high, or elite performer is interesting, but we urge caution as these monikers have little value in the context of your transformation journey.

As our research progresses and evolves, we encourage you to think beyond the four keys. It has become clear that user feedback metrics are as important as the four key metrics. We believe this is because most teams have devised workable solutions for improving

speed and stability. As a result, the benefits gained by speed and stability are diminished as higher performance becomes ubiquitous.

Thinking about transformation holistically, we recommend creating dashboards and visualizations that combine both technical metrics (such as our four keys and reliability metrics) and business metrics. This helps bridge the gap between the top-down and bottom-up transformation efforts. This also helps connect your northstar, OKRs, and employee goals with the investments made in IT. They can help quantify the ROI.

We believe metrics are a requirement for excellence. Metrics facilitate decision making. The more metrics you collect, quantitative and qualitative, the better and more informed decisions you can make. People will always have opinions on the value of the data or the meaning of the data, but using data as the basis by which to make a decision is often preferable to relying on opinion or intuition.



Be all-in on cloud or stay in the data center

We have been investigating the relationship between the NIST defined-5 characteristics of cloud computing⁷ (on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service also known as flexible infrastructure) and organizational performance since 2018. We see that successful teams are more likely to take advantage of flexible infrastructure than less successful teams.

Last year, our research led us to the most striking bit of information on this topic to date: Using the cloud without taking advantage of the five characteristics can be detrimental and predicts decreased organizational performance.

Organizations may be better off staying in the data center if they are not willing to radically transform their application or service. Of course, to accomplish this, it is not simply adopting tools or technologies, but often an entire new paradigm in designing, building, deploying, and running applications. Making large-scale changes is easier when starting with a small number of services, we recommend an iterative approach that helps teams and organizations to learn and improve as they move forward.

Summary

What we've seen consistently over the last 10 years is that transformation is a requirement for success. What many organizations misunderstand is that transformation isn't a destination, but a journey of continuous improvement.⁸ Our research is clear: Companies that are not continuously improving are actually falling behind. Conversely, companies that adopt a mindset of continuous improvement see the highest levels of success.

On this journey, be aware that you will likely hit a little bit of pain and discomfort along the way. Our research has shown an initial drop in performance followed by big gains (also known as the “j-curve”) with DevOps,⁹ SRE,¹⁰ and this year with Platform Engineering. This is normal, and if you are continuously improving, things will get better and you will come out the other end in much better shape than when you started.

The idea of a never-ending journey can seem daunting. It's easy to get stuck in planning or designing the perfect transformation. The key to success is rolling up your sleeves and just getting to work. The goal of the organization and your teams should be to simply be a little better than you were yesterday. The goal of our last 10 years of research and into the future is to help you get better at getting better.

-
1. Dimensions of transformational leadership: Conceptual and empirical extensions - Rafferty, A. E., & Griffin, M. A.
 2. The ROI of DevOps Transformation - <https://dora.dev/research/2020/>
 3. 2015 State of DevOps Report <https://dora.dev/research/2015/2015-state-of-devops-report.pdf#page=25>
 4. 2023 Accelerate State of DevOps Report - <https://dora.dev/research/2023/dora-report/2023-dora-accelerate-state-of-devops-report.pdf#page=17>
 5. DORA's Four Key Metrics <https://dora.dev/guides/dora-metrics-four-keys/>
 6. DORA's capabilities and processes <https://dora.dev/capabilities/>
 7. NIST defined-5 characteristics of cloud computing <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>
 8. Journey of continuous improvement <https://cloud.google.com/transform/moving-shields-into-position-organizing-security-for-digital-transformation>
 9. 2018 Accelerate State of DevOps Report <https://dora.dev/research/2018/dora-report/>
 10. 2022 State of DevOps Report <https://dora.dev/research/2022/dora-report/>

A decade with DORA



History

The DevOps movement was born from two topically-related but otherwise uncoordinated events in 2009. John Allspaw and Paul Hammond gave a talk that June at the Velocity conference titled, "10 deploys per day: Dev & ops cooperation at Flickr".¹ Patrick Debois followed a few months later when he led a team of volunteer organizers to host the first DevOpsDays event in Ghent, Belgium.²

It didn't take long for the DevOps community to want to learn more about how it was evolving. Alana Brown, who was working at Puppet Labs, ran a survey in 2011 to learn more about DevOps. This survey helped confirm that, "working in a 'DevOps' way is emerging as a new way to do business in IT."

As the movement continued to expand to new industries and organizations, Alana built on this success and partnered with IT Revolution Press to field another survey in 2012, publishing their findings in the 2013 State of DevOps Report.³

Dr. Nicole Forsgren joined the research team the following year, bringing more scientific rigor to the program. The 2014 State of DevOps Report⁴ made the connection between software delivery performance and organizational performance, finding that, "publicly traded companies that had high-performing IT teams had 50 percent

higher market capitalization growth over three years than those with low-performing IT organizations."

The trend of annual reports was well-established by 2016, and Forsgren, Jez Humble, and Gene Kim founded DevOps Research and Assessment (DORA). That year, the State of DevOps Report included calculations to help measure the investments made by teams adopting DevOps practices. This work was extended in the ROI of DevOps Transformation⁵ whitepaper, published in 2020.

Accelerate: The science behind devops: Building and scaling high performing technology organizations,⁶ written by Forsgren, Humble, and Kim was published by IT Revolution Press in 2017. This book summarized the early years of the research program and included a focus on the capabilities that drive improvement.

DORA, the company, published an independent report in 2018, the Accelerate State of DevOps: Strategies for a New Economy.⁷ The team at Puppet continued their own series of reports,⁸ separate from DORA, beginning that same year.

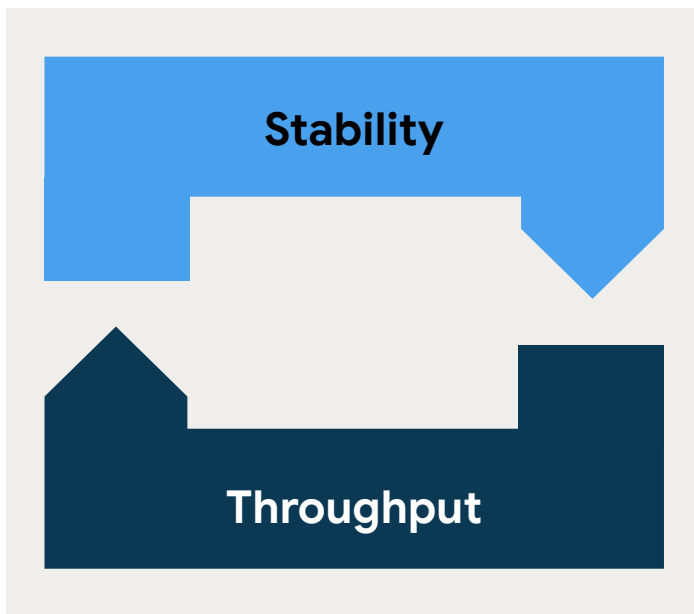
In late 2018, DORA was acquired by Google Cloud⁹ where the platform-agnostic, scientific research continues. This year marks the tenth DORA Report,¹⁰ we are happy to share our findings with you, thank you for reading!

Key insights from DORA

Teams do not need to sacrifice speed for stability

Technology-driven teams need ways to measure performance so that they can assess how they're doing today, prioritize improvements, and validate their progress. DORA identified and has validated four software-delivery metrics—the four keys—that provide an effective way of measuring the outcomes of the software delivery process. These measures of software delivery performance have become an industry standard.

The research has demonstrated that the throughput and stability of changes tend to move together, we have seen teams achieving high levels of both in every industry vertical.



There are many ways that teams measure the four keys including:

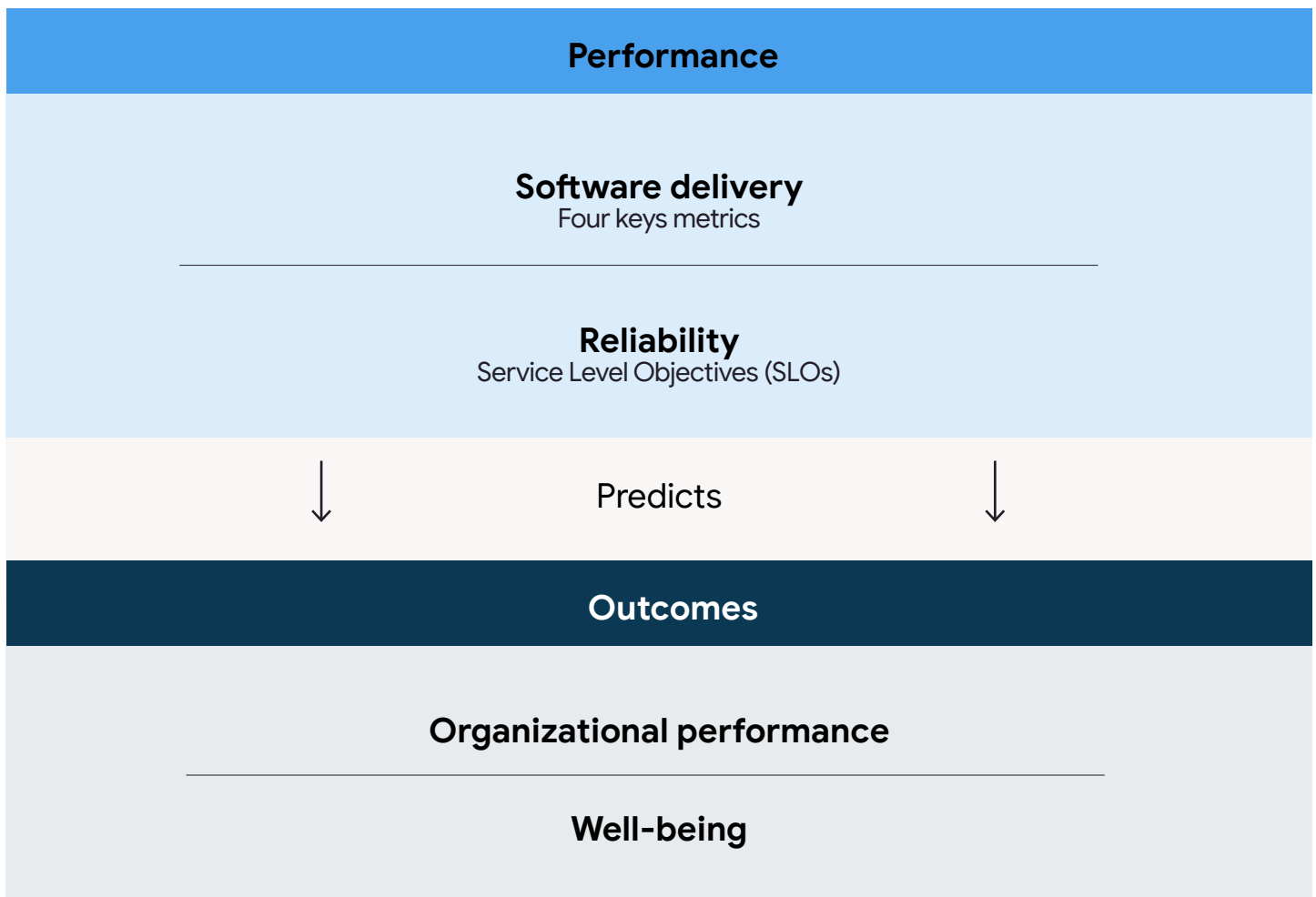
- Through conversations and reflection during team meetings
- The DORA Quick Check (<https://dora.dev/quickcheck>)
- Commercial and source-available¹¹ tools in the Software Engineering Intelligence (SEI) category
- Bespoke integrations built for the specific tools in use by a team

Software delivery and operational performance drive organizational performance

DORA uses the four keys to measure software delivery performance. Operational performance was first studied by DORA in 2018. It measures the ability to make and keep promises and assertions about the software product or service.

The best results are seen when both software delivery and operational performance come together to drive organizational performance and employee well-being.

Practitioners working in technology-driven teams recognize the importance of reducing friction in the delivery process while meeting the reliability expectations of an application's users.





Culture is paramount to success

One of the clearest predictors of performance is the culture of the organization. We've continually seen the power of a high-trust culture that encourages a climate for learning and collaboration. For example, culture was shown to be biggest predictor of an organization's application-development security practices in our 2022 research.¹²

Culture impacts every aspect of our research, and it's multifaceted and always in flux. We've used many different measures over the years with inspiration from research such as Westrum's Typology of Organizational Culture.¹³ Our measures of well-being have included burnout, productivity, and job satisfaction.

Get better at getting better

We encourage teams to set a goal to get better at getting better. Driving improvement requires a mindset and a practice of continuous improvement. This requires a way to assess how you're doing today, prioritize improvement work, and feedback mechanisms that help you measure progress.

An experimental approach to improving will involve a mix of victories and failures, but in both scenarios teams can take meaningful actions as a result of lessons learned.

The decade ahead

Collectively, we've learned a lot from each other over the past decade. Thank you for engaging in our annual surveys, participating in the [DORA Community of Practice](#),¹⁴ and putting DORA to work in your own context.

As the technology landscape continues to evolve, DORA will continue to research the capabilities and practices that help technology-driven teams and organizations succeed. We will continue to prioritize the human aspects of technology and are committed to publishing platform-agnostic research that you can use to guide your own journey.

Many of our past insights are durable enough to inform your approach to emerging technologies and practices and we're excited to find new insights along with you!

We are committed to the fundamentals principles that have always been a part of the DevOps movement: culture, collaboration, automation, learning, and using technology to achieve business goals. Our community and research benefit from the perspectives of diverse roles, including people who might not associate with the "DevOps" label. You should expect to see the term "DevOps" moving out of the spotlight.

This year's report has a heavy focus on the use and impacts of artificial intelligence (AI). As you've read, adoption is growing and there is a lot of room for experimentation in this space. We will continue to investigate this and other emerging technologies and practices into the future. Use our past research, together with our new findings, to drive adoption and help improve the experience of everyone on your team.

-
1. Slides - <https://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr>, recording - <https://www.youtube.com/watch?v=LdOe18KhtT4>
 2. <https://legacy.devopsdays.org/events/2009-ghent/>
 3. <https://www.puppet.com/resources/history-of-devops-reports#2013>
 4. 2014 State of DevOps Report - <https://dora.dev/research/2014/>
 5. The ROI of DevOps Transformation - <https://dora.dev/research/2020/>
 6. Forsgren, Nicole, Jez Humble, and Gene Kim. 2018. Accelerate: The Science Behind DevOps : Building and Scaling High Performing Technology Organizations. IT Revolution Press.
 7. Accelerate State of DevOps: Strategies for a New Economy - <https://dora.dev/research/2018/dora-report/>
 8. <https://www.puppet.com/resources/history-of-devops-reports#2018>
 9. <https://dora.dev/news/dora-joins-google-cloud>
 10. We consider 2014, the year that Dr. Forsgren joined the program, to be the first DORA report, even though DORA was founded a few years later. There was no report in 2020, making 2024 the tenth report.
 11. <https://dora.dev/resources/#source-available-tools>
 12. 2022 Accelerate State of DevOps Report - <https://dora.dev/research/2022/dora-report/>
 13. Ron Westrum, "A typology of organisation culture", BMJ Quality & Safety 13, no. 2(2004), doi:10.1136/qshc.2003.009522
 14. <https://dora.community>

Final thoughts

DORA has established itself as a trusted source of research, insights, and information over the past decade. As the industry continues to adopt new practices and technologies like platform engineering and artificial intelligence, DORA will be here with you, investigating the ways of working that help teams improve. Thank you for having DORA along for the journey.

Replicate our research

The area of research and the findings in this year's report are complex and sometimes unclear or even contradictory. We encourage you to replicate our research. Focusing on a single team or organization opens many opportunities for deeper understanding.

Run experiments within your organization

DORA's findings can serve as hypotheses for your next experiments. Learn more about how your team operates and identify areas for improvement which may be inspired by findings from the DORA research program.

Run surveys within your organization

Take inspiration from this report and the questions used in this year's survey¹ to design your own internal survey. Your survey can incorporate more nuanced questions that are relevant to your audience.² Read the [Methodology](#) chapter for more details into how our research is conducted. Be sure to focus on putting your findings into practice.

Share what you learn

As you learn from your experiments, spread that knowledge throughout your organization. Methods for sharing can range from formal reports for large audiences, through informal communities of practice, to casual chats among peers. Try a variety of approaches and learn what works best in your context and culture. This, too, is an experimental process.



How are you leveraging this research?

Share your experiences, learn from others, and get inspiration from other travelers on the continuous improvement journey by joining the DORA community at <https://dora.community>.



¹ 2024 Survey <https://dora.dev/research/2024/questions/>
² Experiences from Doing DORA Surveys Internally in Software Companies - <https://www.infoq.com/news/2024/08/dora-surveys-software-company/>

Acknowledgements

This year marks a special milestone: the 10th DORA report. We are thankful for all of the dedicated work of researchers, experts, practitioners, leaders, and transformation agents who have joined in shaping this body of work and evolved alongside us.

We've come a long way since the first State of DevOps Report published by Puppet Labs and IT Revolution Press. A heartfelt thank you to our DORA founders for paving the way. It's remarkable to reflect on how much has changed since then and how much we've learned throughout the years.

We're deeply grateful to everyone involved in this year's publication. It's a tremendous responsibility to guide and influence industry practices, and your contributions are invaluable.

To everyone who has been part of this journey, from the early days to this exciting era of AI, thank you. Your support and insights have been instrumental. Here's to the next decade of discovery and collaboration!

DORA Report Team

James Brookbank

Kim Castillo

Derek DeBellis

Benjamin Good

Nathen Harvey

Michelle Irvine

Amanda Lewis

Eric Maxwell

Steve McGhee

Allison Park

Dave Stanke

Kevin Storer

Daniella Villalba

Editor

Seth Rosenblatt

Localization volunteers

Andrew Anolasco

Mauricio Meléndez

Marie-Blanche Panthou

Miguel Reyes

Yoshi Yamaguchi

Jinhong Yu

DORA guides

Lisa Crispin

Steve Fenton

Denali Lumma

Betsalel (Saul) Williamson

Advisors/experts in the field

John Allspaw

Birgitta Böckeler

Sander Bogdan

Michele Chubirka

Thomas De Meo

Jessica DeVita

Rob Edwards

Dr. Nicole Forsgren

Gene Kim and IT Revolution

Laura Maguire, PhD

James Pashutinski

Ryan J. Salva

Majed Samad

Harini Sampath

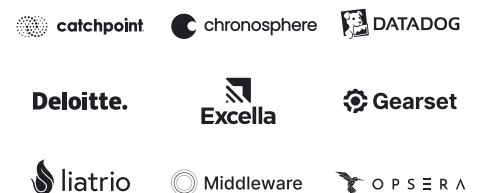
Robin Savinar

Sean Sedlock

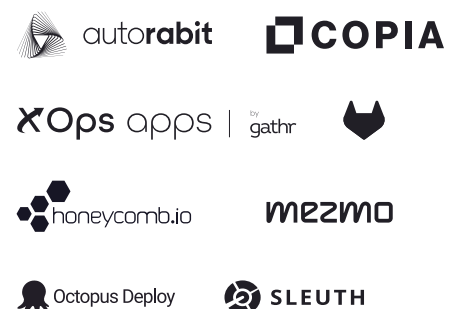
Dustin Smith

Finn Toner

Gold sponsors



Silver sponsors



Authors



Derek DeBellis

Derek is a quantitative user experience researcher at Google and the lead investigator for DORA. Derek focuses on survey research, logs analysis, and figuring out ways to measure concepts that demonstrate a product or feature is delivering capital-v value to people. Derek has published on human-AI interaction, the impact of COVID-19's onset on smoking cessation, designing for NLP errors, the role of UX in privacy discussions, team culture, and AI's relationship to employee well-being and productivity. His current extracurricular research is exploring ways to simulate the propagation of beliefs and power.



Kevin M. Storer

Dr. Kevin M. Storer is a developer experience researcher at Google, where he serves as qualitative research lead for the DORA team. Leveraging professional experience in software engineering and postgraduate transdisciplinary training in the social sciences and humanities, Kevin has been leading human-centered studies of software developers since 2015, spanning a diverse set of problem contexts, participant profiles, and research methods. Kevin's research has been published in top scientific venues on the topics of artificial intelligence, information retrieval, embedded systems, programming languages, ubiquitous computing, and interaction design.



Amanda Lewis

Amanda Lewis is the [DORA.community](#) development lead and a developer relations engineer at Google Cloud. She has spent her career building connections across developers, operators, product managers, project managers, and leadership. She has worked on teams that developed e-commerce platforms, content management systems, observability tools, and supported developers. These connections and conversations lead to happy customers and better outcomes for the business. She brings her experience and empathy to the work that she does helping teams understand and implement software delivery and artificial intelligence practices.



Benjamin Good

Ben Good is cloud solutions architect at Google. He is passionate about improving software delivery practices through cloud technologies and automation. As a solutions architect he gets to help Google Cloud customers solve their problems by providing architectural guidance, publication of technical guides and open source contributions. Prior to joining Google, Ben ran cloud operations for a few different companies in the Denver/Boulder area, implementing DevOps practices along the way.



Daniella Villalba

Daniella Villalba is a user experience researcher at Google. She uses survey research to understand the factors that make developers happy and productive. Before Google, Daniella studied the benefits of meditation training, and the psycho-social factors that affect the experiences of college students. She received her PhD in Experimental Psychology from Florida International University.



Eric Maxwell

Eric Maxwell leads Google's DevOps Transformation practice, where he advises the world's best companies on how to improve by delivering value faster. Eric spent the first half of his career as an engineer in the trenches, automating all the things and building empathy for other practitioners. Eric co-created Google's Cloud Application Modernization Program (CAMP), and is a member of the DORA team. Before Google, Eric spent time whipping up awesome with other punny folks at Chef Software.



Kim Castillo

Kim Castillo is a user experience program manager at Google. Kim leads the cross-functional effort behind DORA, overseeing its research operations and the publication of this report since 2022. Kim also works on UX research for Gemini in Google Cloud. Prior to Google, Kim enjoyed a career in tech working in technical program management and agile coaching. Kim's roots are in psycho-social research focused on topics of extrajudicial killings, urban poor development, and community resilience in her country of origin, the Philippines.



Michelle Irvine

Michelle Irvine is a technical writer at Google, and her research focuses on documentation and other technical communication. Before Google, she worked in educational publishing and as a technical writer for physics simulation software. Michelle has a BSc in Physics, as well as an MA in Rhetoric and Communication Design from the University of Waterloo.



Nathen Harvey

Nathen Harvey leads the DORA team at Google Cloud. Nathen has learned and shared lessons from some incredible organizations, teams, and open source communities. He is a co-author of multiple DORA reports and was a contributor and editor for *97 Things Every Cloud Engineer Should Know*, published by O'Reilly in 2020.

Demographics and firmographics

Who took the survey

The DORA research program has been researching the capabilities, practices, and measures of high-performing, technology-driven organizations for over a decade. We've heard from roughly 39,000 professionals working in organizations of every size and across many different industries. Thank you for sharing your insights! This year, nearly 3,000 working professionals from a variety of industries around the world shared their experiences to help grow our understanding of the factors that drive high-performing, technology-driven organizations.

This year's demographic and firmographic questions leveraged research done by Stack Overflow.

Over 90,000 respondents participated in the 2023 Stack Overflow Developer Survey.¹ That survey didn't reach every technical practitioner, but is about as close as you can get to a census of the developer world.

With a sense of the population provided from that survey, we can locate response bias in our data and understand how far we might want to generalize our findings. Further, the demographic and firmographic questions asked in this Stack Overflow Developer Survey are well-crafted and worth borrowing.

In short, there are no major discrepancies between our sample and Stack Overflow's. This means we have every reason to believe that our sample is reflective of the population.

Industry

We asked survey respondents to identify the industry sector in which their organization primarily operates, across 12 categories. The most common sectors in which respondents worked were Technology (35.69%), Financial Services (15.66%) and Retail/Consumer/E-commerce (9.49%).

Industry	Percentage of respondents
Technology	35.69%
Financial Services	15.66%
Retail/Consumer/E-commerce	9.49%
Other	5.94%
Industrials & Manufacturing	5.49%
Healthcare & Pharmaceuticals	4.60%
Media/Entertainment	4.26%
Government	3.89%
Education	3.66%
Energy	3.03%
Insurance	2.39%
Non-Profit	1%

Number of employees

We asked survey respondents to identify the number of employees at their organization, using nine buckets. The organizations in which respondents worked most commonly had 10,000 or more employees (24.10%), 100 to 499 employees (18.50%) and 1,000 to 9,999 employees (15.60%).

Organization Size	Percentage
solo	2.0%
2 to 9	3.2%
10 to 19	4.3%
20 to 99	14.5%
100 to 499	18.5%
500 to 999	11.2%
1,000 to 4,999	15.6%
5,000 to 9,999	6.7%
10,000 or more	24.1%

Disability

We identified disability along six dimensions that follow guidance from the Washington Group Short Set.²

This is the fifth year we have asked about disability. The percentage of respondents reporting disabilities has decreased from 11% in 2022 to 6% in 2023, and 4% in 2024.

Disability	% of respondents
None of the disabilities applied	92%
At least one of the disabilities applied	4%
Preferred not to say	4%

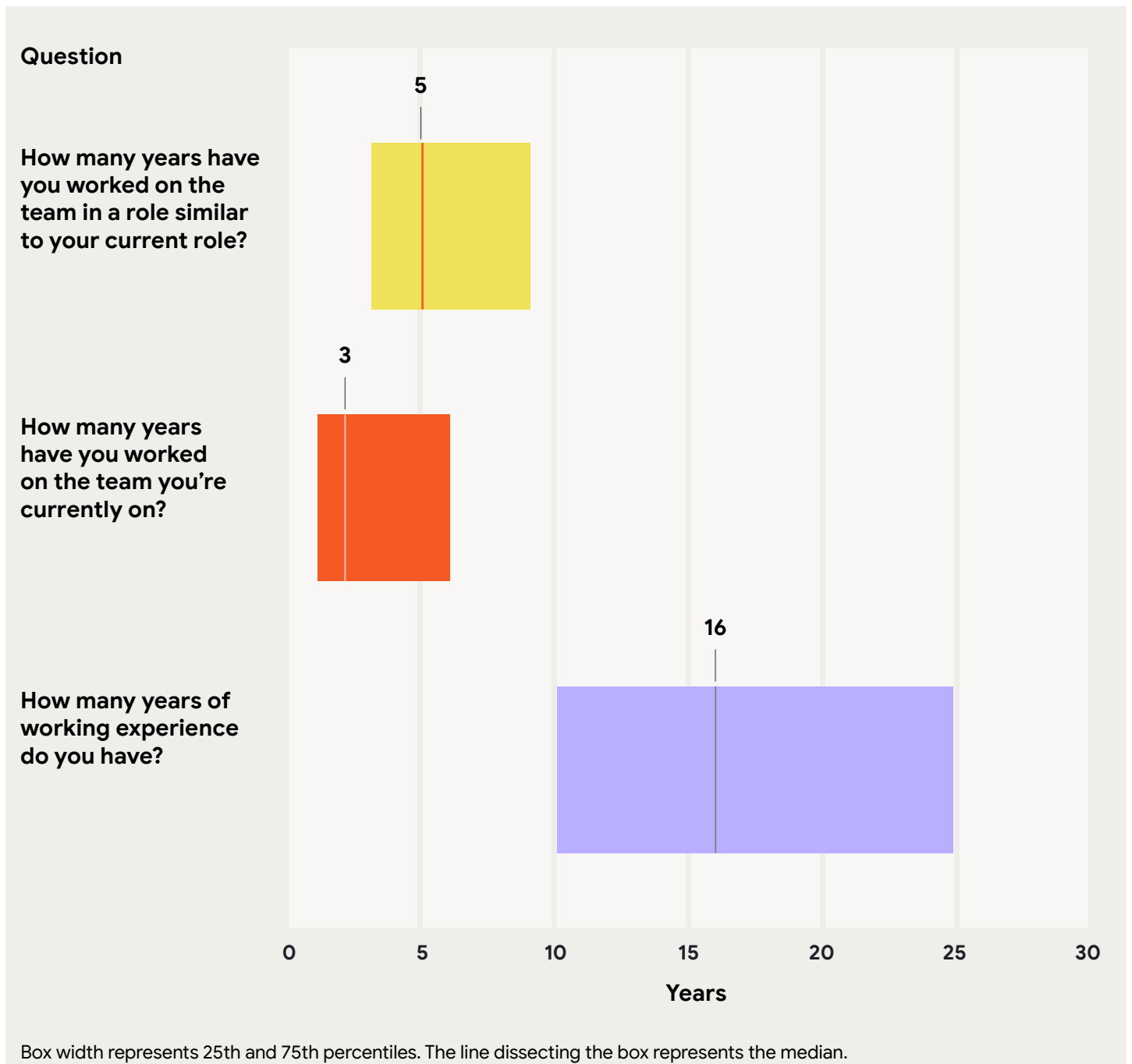
Gender

We asked survey respondents to report their gender. 83% of respondents identified as men, 12% as women, 1% chose to self-describe, and 4% declined to answer.

Gender	Percentage
Man	83%
Woman	12%
Used their own words	1%
Preferred not to answer	4%

Experience

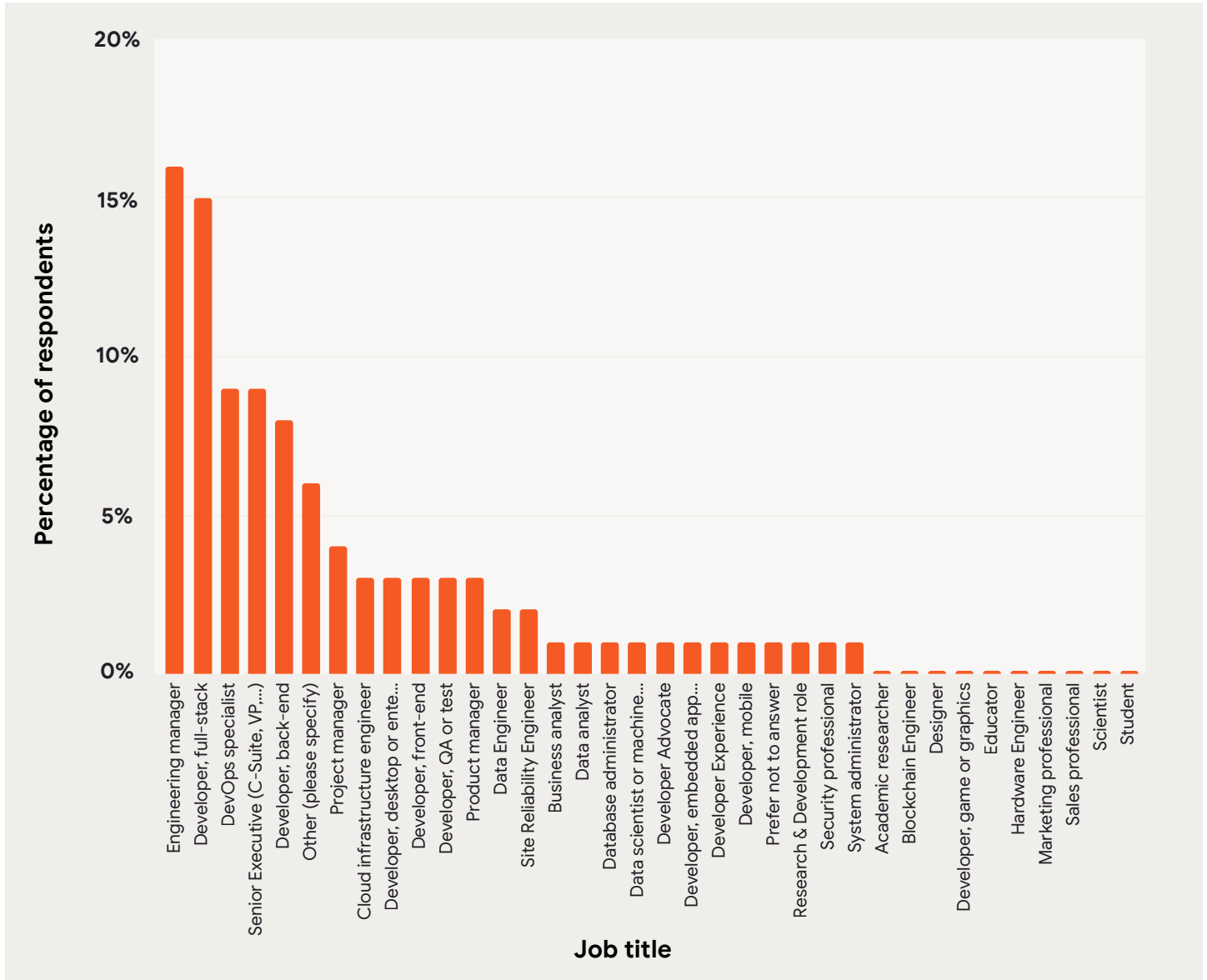
We asked survey respondents to report their years of experience in their role and team. Respondents had a median of 16 years of working experience, five years of experience in their current role, and three years of experience on their current team.



Role

In analyses, some individual roles were grouped together, to help us meaningfully include roles which represented a small proportion of respondents in our analyses. Other categories were highly represented in our data, including:

- Developers, representing 29% of the respondents.
- Managers, representing 23% of the respondents.
- Senior executives, representing 9% of the respondents (+33% from 2023).
- Analytic roles, representing about 5% of the respondents



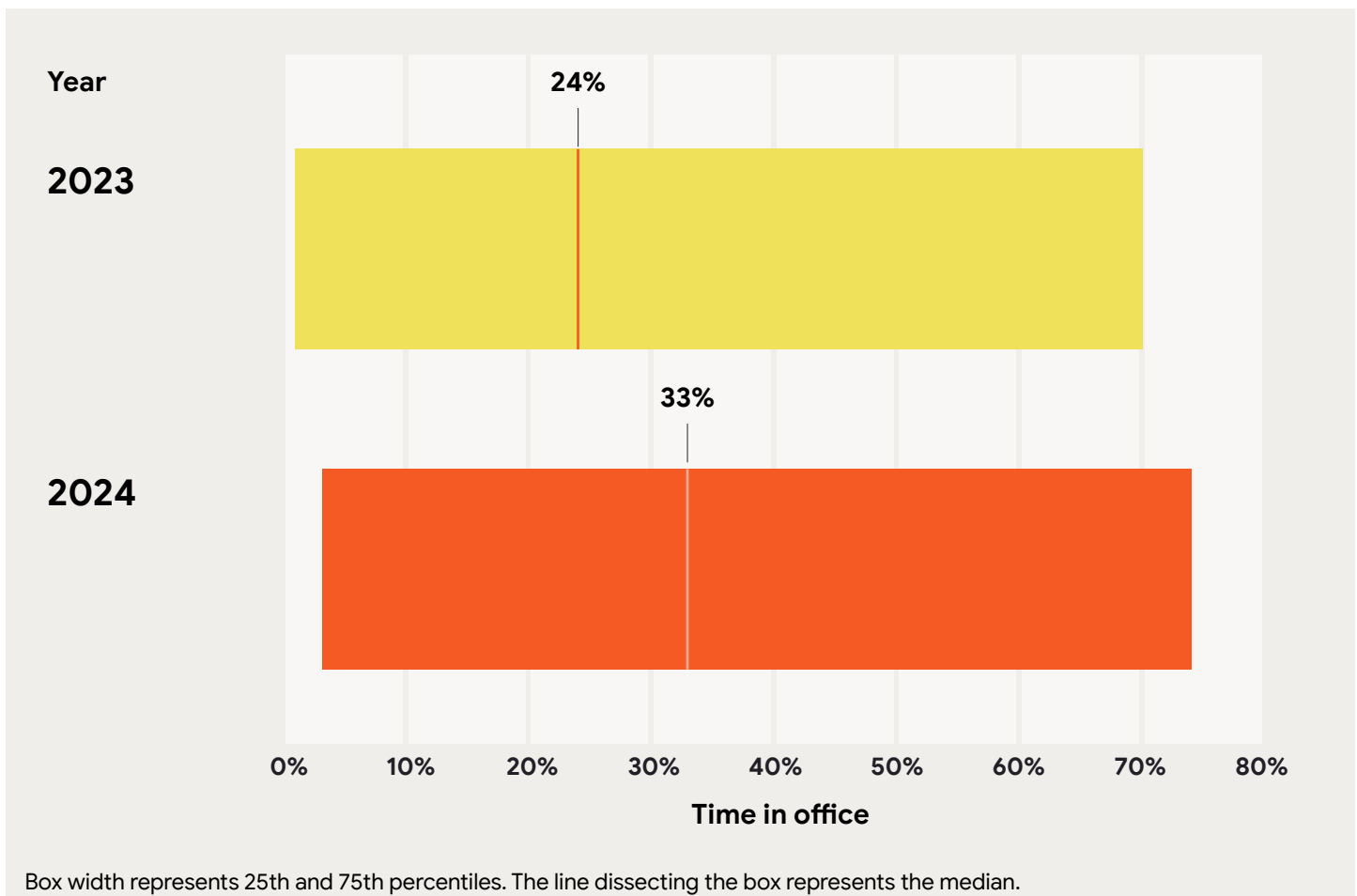
Employment status

We asked survey respondents to report their current employment status. The vast majority (90%) of respondents were full-time employees of an organization.

Employment Type	Percentage
Full-time contractor	6%
Full-time employee	90%
Part-time contractor	1%
Part-time employee	2%

Work location

Despite another year of return-to-office (RTO) pushes, the pattern from last year has largely been retained, especially toward the tails of the distribution. The 37.5% increase in the median values does suggest that hybrid work or at least, some regular visits, are becoming more common.



Country

We had respondents from 104 different countries. We are always thrilled to see people from all over the world participate in the survey. Thank you all!



Country					
USA	Italy	Singapore	Iceland	Luxembourg	Guatemala
UK	Switzerland	Albania	Iran	Nicaragua	Hong Kong (S.A.R.)
Canada	Argentina	Georgia	Jordan	Pakistan	Malta
Germany	Mexico	Greece	Kenya	Peru	Mauritius
Japan	Portugal	Philippines	Saudi Arabia	South Korea	Morocco
India	Austria	Hungary	Slovakia	Sri Lanka	Nepal
France	Romania	Serbia	Slovenia	Tunisia	Paraguay
Brazil	Finland	Afghanistan	Thailand	Andorra	Swaziland
Spain	Turkey	Algeria	Uzbekistan	Barbados	Syrian Arab Republic
Australia	Bulgaria	Egypt	Angola	Belize	Taiwan
Netherlands	Ireland	Indonesia	Armenia	Benin	The former Yugoslav Republic of Macedonia
China	Israel	Russian Federation	Bosnia and Herzegovina	Bolivia	Trinidad and Tobago
Sweden	Belgium	Ukraine	Dominican Republic	Burkina Faso	Uruguay
Norway	Chile	Viet Nam	Ecuador	Comoros	Venezuela, Bolivarian Republic of..
New Zealand	Colombia	Bangladesh	Estonia	Côte d'Ivoire	
Poland	Czech Republic	Belarus	Kazakhstan	El Salvador	
South Africa	Malaysia	Costa Rica	Latvia	Ethiopia	
Denmark	Nigeria	Croatia	Lithuania	Gambia	

Race and ethnicity

We asked survey respondents to report their race and ethnicity. Our largest group of respondents were White (32.4%), and/or European (22.7%).

Race or ethnicity	Percentage
White	32.4
European	22.7
Asian	9.9
North American	4.6
Indian	4.1
Prefer not to say	4.1
Hispanic or Latino/a	3.5
South American	3.2
East Asian	2.5
African	1.8
South Asian	1.7
Multiracial	1.5
Or, in your own words:	1.5
Southeast Asian	1.4
Black	1.3

Race or ethnicity	Percentage
Middle Eastern	1.3
Biracial	0.4
Central American	0.4
I don't know	0.4
North African	0.4
Caribbean	0.2
Central Asian	0.2
South Asian	1.7
Ethnoreligious group	0.2
Pacific Islander	0.2
Indigenous (such as Native American or Indigenous Australian)	0.1

¹ <https://survey.stackoverflow.co/2023/>

² <https://www.washingtongroup-disability.com/question-sets/wg-short-set-on-functioning-wg-ss/>

Methodology

A methodology is supposed to be like a recipe that will help you replicate our work and determine if the way our data was generated and analyzed is likely to return valuable information. Although we don't have the space to go into the exacts, hopefully this is a great starting point for those considerations.

Survey development

Question selection

We think about the following aspects when considering whether to include a question into a survey:

Is this question...

- **Established so we can connect our work to previous efforts?**
- **Capturing an outcome the industry wants to accomplish (for example, high team performance)?**
- **Capturing a capability the industry is considering investing resources into (for example, AI)?**
- **Capturing a capability we believe will help people accomplish their goals (for example, quality documentation)?**
- **Something that helps us evaluate the representativeness of our sample (for example, role or gender)?**
- **Something that helps us block biasing pathways (for example, coding language or role)?**
- **Something that is possible to answer with at least a decent degree of accuracy for the vast majority of respondents?**

We address the literature, engage with the [DORA community](#), conduct cognitive interviews, run parallel qualitative research, work with subject matter experts, and hold team workshops to inform our decision as to whether to include a question into our survey.

Survey experience

We take great care to improve the usability of the survey. We conduct cognitive interviews and usability tests to make sure that the survey hits certain specification points:

- **Time needed to complete survey should, on average, be low**
- **Comprehension of the questionnaire should be high**
- **Effortfulness should be reasonably low, which is a huge challenge given the technical nature of the concepts**

Data collection

Localizations

People around the world have responded to our survey every year. This year we worked to make the survey more accessible to a larger audience by localizing the survey into English, Español, Français, Português, 日本語, and 简体中文.



Collect survey responses

We use multiple channels to recruit. These channels fall into two categories: organic and panel.

The organic approach is to use all the social means at our disposal to let people know that there is a survey that we want them to take. We create blog posts. We use email campaigns. We post on social media, and we ask people in the community to do the same (that is, snowball sampling).

We use the panel approach to supplement the organic channel. Here we try to recruit people who are traditionally underrepresented in the broader technical community and try to get adequate responses from certain industries and organization types.

In short, this is where we get some control over our recruitment—control we don't have with the organic approach. The panel approach also allows us to simply make sure that we get enough respondents, because we never know if the organic approach is going to yield the responses necessary to do the types of analyses we do. This year we had sufficient organic responses to run our analysis and the panel helped round out our group of participants.

Survey flow

This year we had a lot of questions we wanted to ask, but not enough time to ask them. Our options were...

- **Make an extremely long survey**
- **Choose a subset of areas to focus on**
- **Randomly assign people to different topics**

We didn't want to give up on any of our interests, so we chose to randomly assign participants to one of three separate flows. There was a lot of overlap among the three different flows, but each flow dove deeply in a different space.

Here are the three different pathways:

- **AI**
- **Workplace**
- **Platform Engineering**

Survey analysis

Measurement validation

There is a wide variety of concepts that we try to capture in the survey. There are a lot of different language games we could partake in, but one view is that this measure of a concept is called a variable. These variables are the ingredients of the models, which are the elements included in our research. There are two broad ways to analyze the validity of these measures: internally and externally.

To understand the internal validity of the measure, we look at what we think indicates the presence of a concept. For example, quality documentation might be indicated by people using their documentation to solve problems.

A majority of our variables consist of multiple indicators because the constructs we're interested in appear to be multifaceted.

To understand the multifaceted nature of a variable, we test how well the items we use to represent that concept gel. If they gel well (that is, they share a high level of communal variance), we assume that something underlies them—such as the concept of interest.

Think of happiness, for example, happiness is multifaceted. We expect someone to feel a certain way, act a certain way, and think a certain way

when they're happy. We assume that happiness is underlying a certain pattern of feelings, thoughts, and action.

Therefore, we expect certain types of feelings, thoughts, and actions to emerge together when happiness is present. We would then ask questions about these feelings, thoughts, and actions. We would use confirmatory factor analysis to test whether they actually do show up together.

This year we used the lavaan¹ R package to do this analysis. Lavaan returns a variety of fit statistics that help us understand whether constructs actually represent the way people answer the questions.

If the indicators of a concept don't gel, the concepts might need to be revised or dropped because it's clear that we haven't found a reliable way to measure the concept.

The external validity of a construct is all about looking at how the construct fits into the world. We might expect a construct to have certain relationships to other constructs. Sometimes we might expect two constructs to have a negative relationship, like happiness and sadness.

If our happiness measure comes back positively correlated with sadness, we might question our measure or our theory.

Similarly, we might expect two constructs to have positive relationships, but not strong ones. Productivity and job satisfaction are likely to be positively correlated, but we don't think they're identical. If the correlation gets too high, we might say it looks like we're measuring the same thing. This then means that our measures are not calibrated enough to pick up on the differences between the two concepts, or the difference we hypothesized isn't actually there.

Model evaluation

Using a set of hypotheses as our guiding principle, we build hypothetical models, little toys that try to capture some aspect about how the world works. We examine how well those models fit the data we collected. For evaluating a model, we go for parsimony. This amounts to starting with a very simplistic model² and adding complexity until the complexity is no longer justified.

For example, we predict that organizational performance is the product of the interaction between software delivery performance and operational performance. Our simplistic model doesn't include the interaction:

Organizational performance ~ Software delivery performance + Operational performance

Our second model adds the interaction:

Organizational performance ~ Software delivery performance + Operational performance + Software delivery performance × Operational performance

Based on the recommendations in “Regression and other stories”³ and “Statistical Rethinking,”⁴ we use leave-one-out cross-validation (LOOCV)⁵ and Watanabe–Akaike information criterion⁶ to determine whether the additional complexity is necessary.

Directed Acyclic Graphs for Causal Inference

A validated model tells us what we need to know to start thinking causally. We talk about the challenges of thinking causally below.

Here are some reasons why we're *trying* to talk causally:

We think your question is fundamentally a causal one. You want to know if doing something is going to create something. You are not going to invest in doing something if you just think there is a non-causal correlation.

The results of our analyses depend on our causal understanding of the world. The actual numbers we get from the regression change based on what we include in the regression. What we include in the regression should depend on how we think the data is generated, which is a causal claim. Hence, we should be clear.

Causal thinking is where our curiosity will take us and where we all spend a lot of time. We are often wondering about how the various aspects of the world are connected and why. We don't need to run experiments on every facet of our lives to think causally about them.

Causal thinking is central to action, which is what we're hoping this report helps you with, making decisions to act.

We are able to use the validated model to tell us what we need to account for to understand an effect. In short, it lets us try to get our data in the form of an A/B experiment, where one tries to create two identical worlds with only one difference between them. The logic suggests that in doing so any differences that emerge between those two worlds is attributable to that initial difference.

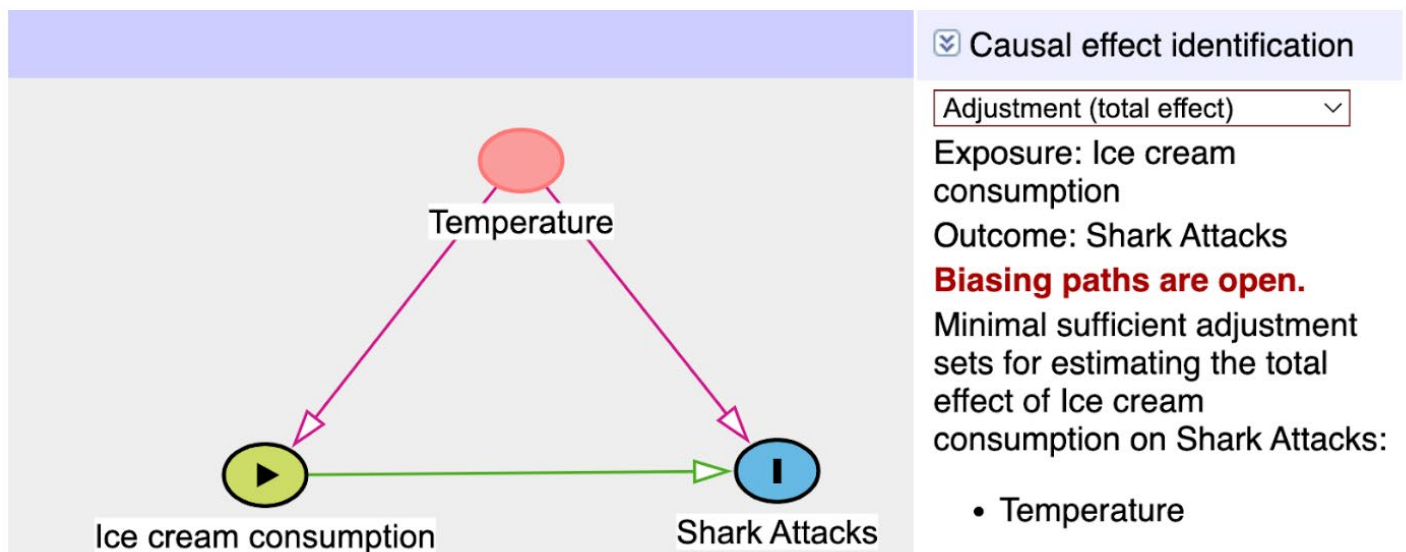
In observational data and survey data, things are not as clearly divided — many things are different between participants, which introduces confounds. Our method of causal inference tries to account for these differences in an attempt to mimic an experiment — that is, holding everything constant except for one thing (for example, AI adoption).

Let's take the classic example of ice cream "causing" shark attacks. There is a problem in that observation, namely that people tend to eat ice cream on hot days and also go to the beach on hot days. The situation where people tend to eat ice cream and go to the beach is not the same as the situation where people tend not to eat ice cream and not go to the beach. The data isn't following the logic of an experiment. We've got a confounding variable, temperature.

Directed Acyclic Graphs (DAGs) help you identify the ways in which the world is different and offer approaches to remedy the situation, to try to mimic an experiment by making everything in the world except one thing constant. Let's see how the DAG directs us in the ice cream and shark attack example, where we want to quantify the impact of ice cream consumption on shark attacks:

I draw my model, tell the tool what effect I want to understand, and the tool tells me what is going to bias my estimate of the effect. In this case, the tool says that I cannot estimate the effect of ice cream consumption on shark attacks without adjusting for temperature, which is a statistical approach of trying to make everything equal besides ice cream consumption and then seeing if shark attacks continue to fluctuate as a function of ice cream consumption.

We outline our models in the, you guessed it, [Models](#) chapter.



The image is from <https://www.dagitty.net/dags.html>.

The directed acyclic graph tells us what to account for in our analyses of particular effects.

For example, what do we need to account for in our analysis of AI adoption's impact on productivity?

Bayesian statistics

This analysis is done using Bayesian statistics. Bayesian statistics offer a lot of benefits:

- **We move away from thinking in terms of significant or insignificant (ask 10 people to explain frequentist p-values and you'll get 10 different answers)**
- **We want to know the probability of hypothesis given the data, not the probability of the data given our hypothesis**
- **We like to incorporate our prior knowledge into our models, or at least be explicit about how much we don't know⁷**
- **We are forced to confront the underlying assumptions of the modeling process**
- **We can explore the posterior distributions to get a sense of the magnitude, uncertainty, and overall, how and how well the model made sense of the data. Ultimately, it gives a great sense of what we do and do not know given our data**
- **A flexible framework that addresses many statistical problems in a very unified manner**

What do you mean by “simulation”?

It isn't that we made up the data. We use Bayesian statistics to calculate a posterior, which tries to capture “the expected frequency that different parameter values will appear.”⁸ The “simulation” part is drawing from this posterior more than 1,000 times to explore the values that are most credible for a parameter (mean, beta weight, sigma, intercept, etc.) given our data.

“Imagine the posterior is a bucket full of parameter values, numbers such as 0.1, 0.7, 0.5, 1, etc. Within the bucket, each value exists in proportion to its posterior probability, such that values near the peak are much more common than those in the tails.”⁹

This all amounts to our using simulations to explore possible interpretations of the data and get a sense of how much

uncertainty there is. You can think of each simulation as a little AI that knows nothing besides our data and a few rules trying to fill in a blank (parameter) with an informed guess. You do this 4,000 times and you get the guesses of 4,000 little AIs for a given parameter.

You can learn a lot from these guesses. You can learn what the average guess is, between which values do 89%¹⁰ of these guesses fall, how many guesses are above a certain level, how much variation is there in these guesses, etc. You can even do fun things like combine guesses (simulations) across many models.

When we show a graph with a bunch of lines or a distribution of potential values, we are trying to show you what is most plausible given our data and how much uncertainty there is.

Synthesize findings with the community

Our findings offer a valuable perspectives for technology-driven teams and organizations, but they are best understood through dialogue and shared learning. Engaging with the DORA community gives us diverse insights, challenges our assumptions, and helps us discover new ways to interpret and apply these findings.

We encourage you to join the DORA community (<https://dora.community>) to share your experiences, learn from others, and discover diverse approaches to implementing these recommendations. Together, we can explore the best ways to leverage these insights and drive meaningful change within your organization.

Interviews

This year, we supplemented our annual survey with in-depth, semi-structured interviews to triangulate, contextualize, and clarify our quantitative findings. The interview guide paralleled the topics included in our survey and was designed for sessions to last approximately 75 minutes each, conducted remotely via Google Meet.

In total, we interviewed 11 participants whose profiles matched the inclusion criteria of our survey. All interviews were video- and audio-recorded. Sessions lasted between 57 minutes and 85 minutes, totaling 14 hours and 15 minutes of data collected across all participants. Participants' data were pseudonymized using identifiers in the form of P(N), where N corresponds to the order in which they were interviewed.

All interviews were transcribed using automated software. Transcriptions were manually coded using our survey topics as a priori codes. Quotations appearing in the final publication of this report were revisited and transcribed manually prior to inclusion. Words added to participant quotations by the authors of this report are indicated by brackets ([]), words removed are indicated by ellipses (..), and edits were made only in cases where required for clarity.

Inferential leaps in results

Our goal is to create a pragmatic representation of the world, something that we can all leverage to help improve the way we work. We know there is complexity we're simplifying. That is kind of the point of the model. Jorge Luis Borges has a very short story, called "On Exactitude in Science", where he talks of an empire that makes maps of the empire on a 1:1 scale.¹¹ The absurdity is that this renders the map absolutely useless (at least that's my interpretation). The simplifications we make are supposed to be helpful.

That said, there are some inferential leaps that we want to be clear about.

Causality

According to John Stuart Mill, you needed to check three boxes to say X causes Y:¹²

- **Correlation:** X needs to covary with Y?
- **Temporal precedence:** X needs to happen before Y?
- Biasing pathways are accounted for (as described in the DAG section above)?

We feel confident that we can understand correlation — that's often a standard statistical procedure. Our survey is capturing a moment in time, so temporal precedence is theoretical, not part of our data.

As for biasing pathways, as we mention above when talking about structural equation models and directed acyclic

graphs, we do the work to account for biasing pathways, but that is a highly theoretical exercise, one that, unlike temporal precedence, has implications that can be explored in the data.

This is all to say that we didn't do longitudinal studies or a proper experiment. Despite this, we think causal thinking is how we understand the world and we try our best to use emerging techniques in causal inference to provide you with good estimates. Correlation does not imply causation, but it does imply how you think about causation.

Micro-level phenomena

-> Macro-level phenomena

Often we take capabilities at an individual level and see how those connect to higher levels. For example, we tied the individual adoption of AI to an application or service and to team performance. This isn't terribly intuitive at first glance. The story of a macro-level phenomenon causing an individual level phenomenon is usually easier to tell. Inflation (macro) impacting whether I buy eggs (micro) seems like a more palatable story than me not buying eggs causing inflation.

The same is true for an organization's performance (macro) impacting an individual's well-being (micro). As a heuristic, it is likely the organization exerts more of an influence on the individual than the individual on the organization.

So, why do we even bother saying an individual action impacts something like team or organizational performance? We make an inferential leap that we think isn't completely illogical. Namely, we assume that at scale, the following statement tends to be true:

$$p(\text{individual does X} \mid \text{organization does X}) > p(\text{individual does X} \mid \text{organization doesn't do X}).$$

That is, we believe that the probability of an individual doing something (X) is higher when they are in an organization or a team that also does X. Hence, individuals who do something represent teams and organizations that also tend to do X. Of course the noise here is pretty loud, but the pattern should emerge and allow this assumption to give us some important abilities.

Let's back up for an example outside of DORA: imagine two different countries where the average height differs. In one country, people have an average height of 5'6". The other's average height is 6'2". The standard deviation is identical. If you picked a person at random from each country, which country do you think the taller person would be more likely to be drawn from? If you do this thousands of times, taller countries would be represented by taller people. The height of the individuals would loosely approximate the heights of the countries.

Not that it is necessary, but we ran a quick simulation to validate this is true:

```
#R code
#set seed for reproducibility
set.seed(10)

#6'2 and 5'6
height_means = c(6 + 1/6, 5.5)

#constant standard deviation at 1/4 of
foot
std_dev = 0.25

#random draws
draws = 1000

#random draws from country A
country_a <- rnorm(draws, mean = height_
means[1], sd = std_dev)

#random draws from country B
country_b <- rnorm(draws, mean = height_
means[2], sd = std_dev)

#how of the draws represent the correct
difference
represented_difference = sum(country_a >
country_b) / 1000

#show results as percentage
represented_difference * 100
```

The results are unsurprising. 97.2% of the 1,000 random draws are in the correct direction. Of course, it would be easy to get fooled with non-random draws, smaller differences between the countries, and small samples. Still, the point stands: differences at the macro-level tend to be represented in the micro-level.

1. Rosseel Y (2012). "lavaan: An R Package for Structural Equation Modeling." *Journal of Statistical Software*, 48(2), 1–36. <https://doi.org/10.18637/jss.v048.i02>
2. This would also involve the examination of potential confounds.
3. Gelman, Andrew, Jennifer Hill, and Aki Vehtari. 2021. *Regression and Other Stories*. N.p.: Cambridge University Press.
4. McElreath, Richard. 2016. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. N.p.: CRC Press/Taylor & Francis Group.
5. Gelman, Andrew, Jennifer Hill, and Aki Vehtari. 2021. *Regression and Other Stories*. N.p.: Cambridge University Press
6. McElreath, Richard. 2016. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. N.p.: CRC Press/Taylor & Francis Group.
7. Our priors tend to be weak (skeptical, neutral, and low information) and we check that the results are not conditioned by our priors.
8. McElreath, Richard. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2018, pg. 50
9. McElreath, Richard. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2018, pg. 52
10. Followed McElreath's reasoning in *Statistical Rethinking*, pg. 56 for choosing 89%. "Why these values? No reason... And these values avoid conventional 95%, since conventional 95% intervals encourage many readers to conduct unconscious hypothesis tests." The interval we're providing is simply trying to show a plausible "range of parameter values compatible with the model and data".
11. Borges, J. L. (1999). *Collected fictions*. Penguin.
12. Duckworth, Angela Lee, Eli Tsukayama, and Henry May. "Establishing causality using longitudinal hierarchical linear modeling: An illustration predicting achievement from self-control." *Social psychological and personality science* 1, no. 4 (2010): 311-317.

Models

Traditionally, we have built one giant model that we validated using various structural equation modeling techniques (partial least squares, covariance-based, bayesian). For the 2023 report, we switched to focusing on many smaller models aimed at helping us understand specific processes.

For example, we made a nuanced model to understand the physics of quality documentation. There are important benefits that come with creating smaller models¹ tailored to understanding specific effects:

- **Ease of identifying areas of poor model fit**
- **Everything you add to a model exerts a force, has a gravity. As your model gets large, it is really difficult to understand all the different ways the variables are exerting force on each other**
- **Prevents you from conditioning on something that creates spurious relationships²**

How do we use the models?

We all have a lot of questions, but many vital questions have the following form:

if we do X, what happens to Y?

X is usually a practice, such as creating quality documentation, adopting AI, or investing in culture.

Y is usually something that we care about achieving or avoiding, which could happen at the individual level (for example, productivity) up to the organizational level (for example, market share).

We construct, evaluate, and use the models³ with the goal of addressing questions of this form. We work to provide an accurate estimate of what happens to important outcomes as a result of doing X.⁴ When we report effects, we convey two vital features:

1. How much certainty we have in the **direction** of the effect, that is, how clear is it that this practice will be beneficial or detrimental?
2. How much certainty we have in the **magnitude** of the effect. We will provide an estimate a relative sense of how impactful certain practices are and the degree of uncertainty surrounding these estimates.

Here are some of this year's capabilities of interest:

- **AI adoption**
- **platform use**
- **platform age**
- **transformational leadership**
- **priority stability**
- **user centricity**

Here are some of this year's outcomes and outcome groups:

- **individual performance and well-being (for example, burnout)**
- **team performance**
- **product performance**
- **development workflow (for example, codebase complexity and document quality)**
- **software delivery performance**
- **organizational performance**

We focus on these outcomes because we believe that they are ends in themselves. Of course, that is more true for some of these outcomes than others. If you found out that organizational performance and team performance had nothing to do with the software delivery performance, you would probably be okay having low software delivery performance.

We hope, however, that even if organizational performance did not depend on individual well-being you would still want to prioritize the well-being of employees.

A repeated model

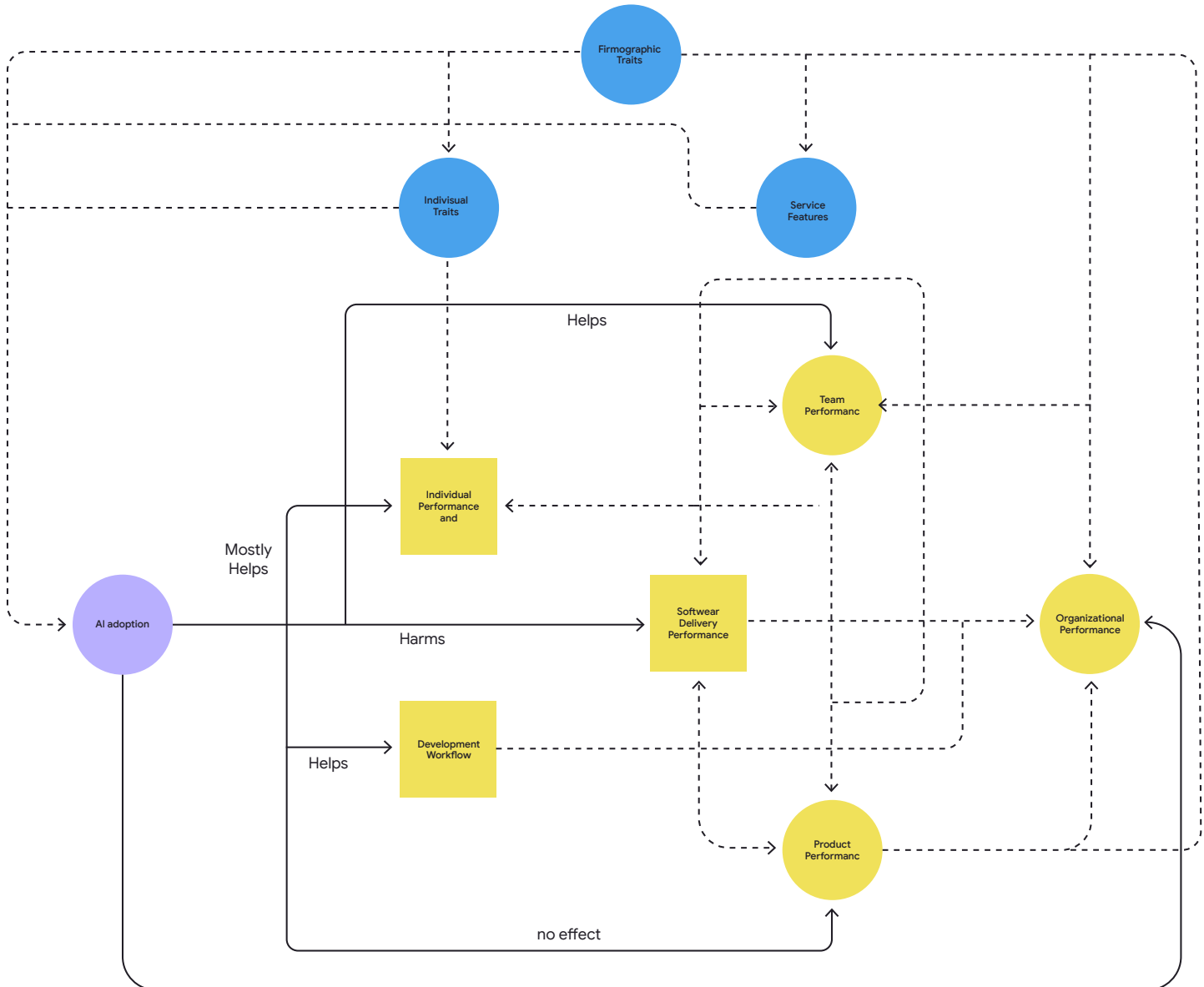
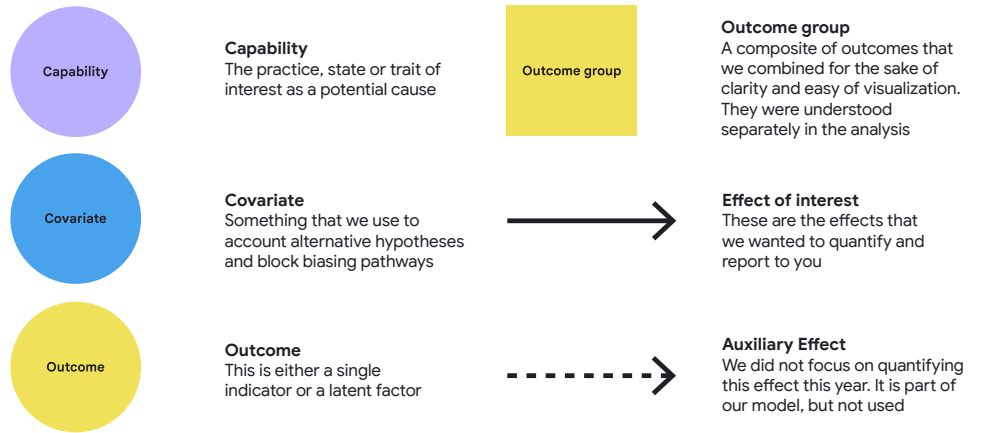
We developed and explored many nuanced hypotheses over the past three years, especially about moderation and mediation.

This year, we spent less time focusing on those types of hypotheses and more time trying to estimate a capability's effect on an outcome. This means that the model for each capability is largely the same.

Hence, the model for AI adoption's effects is very similar in design to the model for User-centricity's effects. We could copy the model and change the name of capability, but that might not be terribly useful for you.

Instead, we are just going to show the AI model, but know it is the schematic or form behind each of our models. Should you be interested in running your analysis, constructing this model in a tool like [DAGitty](#) should allow you to get close to replicating the regressions we used in our analysis. That said, what is presented is slightly simplified for readability. Additionally, while the models are very similar across each capability, the effects are different. For example you'll see below that AI adoption generally harms software delivery performance but the opposite is true for things like internal documentation and user-centricity, see each chapter for additional details.

The key



1. Gelman et. al's "Regression and other stories" offers some important tips on page 495 through 496 that seem illuminating: B.6 Fit many models and B.9 Do causal inference in a targeted way, not as a byproduct of a large regression
2. A great discussion about this can be found in chapter 6 of Statistical Rethinking. I am talking particularly about collider bias.
3. See the conversation about how these models are tied with directed acyclic graphs in the methodology chapter
4. We talk about causality briefly in the methods chapter.

Recommended reading

Join the DORA Community to discuss, learn, and collaborate on improving software delivery and operations performance. <https://dora.community>

Take the DORA Quick Check. <https://dora.dev/quickcheck>

Explore the capabilities that enable a climate for learning, fast flow, and fast feedback. <https://dora.dev/capabilities>

Fostering developers' trust in generative artificial intelligence. <https://dora.dev/research/2024/trust-in-ai/>

Read the book: *Accelerate: The science behind devops: Building and scaling high performing technology organizations*. IT Revolution. <https://itrevolution.com/product/accelerate>

Read the book: *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. IT Revolution Press. <https://teampologies.com/>

Publications from DORA's research program, including prior DORA Reports. <https://dora.dev/publications>

Frequently asked questions about the research and the reports. <http://dora.dev/faq>

Errata - Read and submit changes, corrections, and clarifications to this report. <https://dora.dev/publications/errata>

Check if this is the latest version of the 2024 DORA Report: <https://dora.dev/vc/?v=2024.3>

“Accelerate State of DevOps 2024”
by Google LLC is licensed under [CC BY-NC-SA 4.0](#)

