

2022 ACCELERATE

State of DevOps Report



Sponsored by



Table of contents

| | | | | | |
|-----------|--|----|-----------|---------------------------------------|----|
| 01 | Executive summary | 03 | 06 | Demographics and Firmographics | 59 |
| 02 | How do you compare? | 08 | 07 | Final thoughts | 67 |
| 03 | How do you improve? | | 08 | Acknowledgements | 68 |
| | Introduction | 19 | 09 | Authors | 69 |
| | Cloud | 21 | 10 | Methodology | 73 |
| | SRE and DevOps | 26 | 11 | Further reading | 76 |
| | Technical DevOps Capabilities | 29 | 12 | Appendix | 78 |
| | Culture | 37 | | | |
| 04 | Why supply chain security matters | 42 | | | |
| 05 | Surprises | 55 | | | |

01

Executive summary



Derek DeBellis



Claire Peters

For the last eight years, we have produced the Accelerate State of DevOps report, hearing from 33,000 professionals along the way. Our research focuses on examining how capabilities and practices predict the outcomes that we consider central to DevOps:

- Software delivery performance – The Four Key Metrics of software delivery performance: deployment frequency, lead time for changes, change failure rate, and time to restore service.
- Operational performance – The Fifth Key Metric, reliability.
- Organizational performance – How well your organization meets performance and profitability goals.

We also focus on the factors that underlie other outcomes like burnout and the likelihood that employees will recommend their teams.



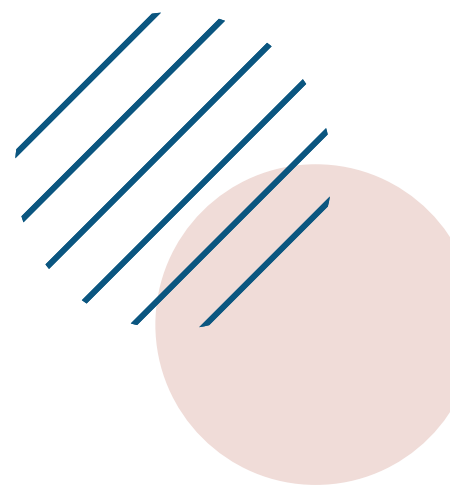
Securing the software supply chain

In 2021, we found that securing the software supply chain is essential to reaching many important outcomes.

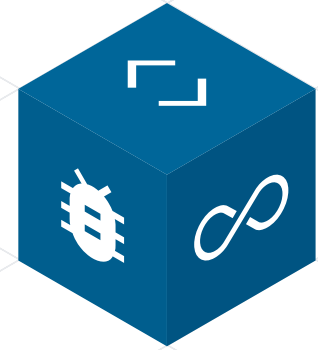
This year we dug deeper on software supply chain security, making it a primary theme of our survey and report. We leveraged the [Supply Chain Levels for Secure Artifacts \(SLSA\)](#) framework to explore technical practices that support the development of software supply chain security. We also used the National Institute for Standards and Technology's [Secure Software Development Framework \(NIST SSDF\)](#) to explore attitudes, processes, and non-technical practices related to securing the software supply chain.

We found that the biggest predictor of an organization's application-development security practices was cultural, not technical: high-trust, low-blame cultures focused on performance were 1.6x more likely to have above average adoption of emerging security practices than low trust, high-blame cultures focused on power or rules. We also found early evidence suggesting that pre-deployment security scanning is effective at finding vulnerable dependencies, resulting in fewer vulnerabilities in production code.

Adoption of good application development security practices was correlated with additional benefits. We found that teams that focus on establishing these security practices have reduced developer burnout; teams with low levels of security practices have 1.4x greater odds of having high levels of burnout than teams with high levels of security.¹ The teams that focus on establishing security practices are significantly more likely to recommend their team to someone else. Further, SLSA-related security practices positively predict both organizational performance and software delivery performance, but this effect needs strong continuous integration capabilities in place to fully emerge.



¹ We conceptualize high in this stat as ≥ 1 standard deviation on the score (e.g. security) and low as ≤ -1 standard deviation on the score.



Drivers of organizational performance

Besides the security practices mentioned above, the key variables that impact organizational performance tend to fall in the following categories:

Organizational and team culture

High-trust and low-blame cultures, as defined by [Westrum](#), tend to have higher organizational performance. Similarly, organizations with teams that feel supported through funding and leadership sponsorships tend to have higher organizational performance. Team stability and positive perceptions about one's team (likelihood to recommend team) also tend to lead to higher levels of organizational performance. Lastly, companies that offer flexible work arrangements tend to see high levels of organizational performance.

Reliability

Both the practices we associate with reliability engineering (e.g., clear reliability goals, salient reliability metrics, etc.) and the extent to which people report meeting their reliability expectations are powerful predictors of high levels of organizational performance.

Cloud

We found that cloud usage is predictive of organizational performance. Companies with software initially built on and for the cloud tend to have higher organizational performance. Using private clouds, public clouds, hybrid clouds, or a mixture of clouds, corresponds with higher organizational performance than the use of on-premises servers alone. Those who use multiple public clouds are 1.4x more likely to have above average organizational performance than those who don't.

Cloud usage also seems to impact organizational performance through other factors in our dataset. One example is supply chain security, where we found that organizations using public clouds were also more likely to implement SLSA practices—perhaps because cloud providers encourage and provide building blocks for many SLSA practices, such as automating builds and deployments.^{2,3} The broader point is that using cloud platforms opens a team up to inherit many capabilities and practices that eventually flow into higher organizational performance.”

²Jung, Sun Jae. “Introduction to Mediation Analysis and Examples of Its Application to Real-world Data.” *Journal of preventive medicine and public health = Yebang Uihakhoe chi* vol. 54,3 (2021): 166-172. doi:10.3961/jpmph.21.069


³ Carrión, Gabriel Cepeda, Christian Nitzl, and José L. Roldán. “Mediation analyses in partial least squares structural equation modeling: Guidelines and empirical examples.” *Partial least squares path modeling*. Springer, Cham, 2017. 173-195.

Context matters

For a long time, DORA has taken into consideration that effects depend on broader team context. We believe it's important to understand a team's characteristics (processes, strengths, constraints, and goals), and the environment in which the work takes place. For example, a technical capability that is advantageous in one context could be deleterious in another. This year we focused on explicitly modeling these hypothesized conditions in the form of interactions; many of these hypotheses are supported by this year's data:

- High software delivery performance is only beneficial to organizational performance when operational performance is also high. Delivering quickly might not matter if your service is unable to meet users' reliability expectations.
- Implementing software supply chain security controls, like those recommended by the SLSA framework, has a positive effect on software delivery performance when continuous integration is firmly established. Without continuous integration capabilities in place, software delivery performance and security controls might be in conflict.
- The impact of Site Reliability Engineering (SRE) practices on a team's ability to reach reliability targets is non-linear. Practicing SRE doesn't positively affect reliability until a team achieves a certain level of SRE maturity. Before a team reaches this level of SRE maturity, we don't detect a relationship between SRE and reaching reliability targets. As a team's SRE adoption grows, however, it reaches an inflection point where the use of SRE starts to strongly predict reliability. The improved reliability then impacts organizational performance.
- Technical capabilities build upon one another. Continuous delivery and version control amplify each other's ability to promote high levels of software delivery performance. Combining continuous delivery, loosely-coupled architecture, version control, and continuous integration fosters software delivery performance that is greater than the sum of its parts.



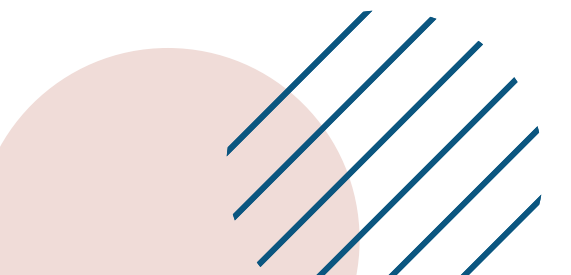


The conditions upon which delivery depends, and the need to understand a team's broader context, leads us to a conclusion that is similar to this insight from 2021:

“To make meaningful improvements, teams must adopt a philosophy of continuous improvement. Use the benchmarks to measure your current state, identify constraints based on the capabilities investigated by the research, and experiment with improvements to relieve those constraints. Experimentation will involve a mix of victories and failures, but in both scenarios teams can take meaningful actions as a result of lessons learned.”

Indeed, we found an effect this year that is deeply aligned with this overarching philosophy: teams that recognize the need to continuously improve tend to have higher organizational performance than those that don't.

In short, teams need to continuously adapt, and to experiment with software development practices.



We know this because, overall, teams that do this have higher organizational performance. Not always (what works for one organization does not necessarily work for another), but most of the time. As you engage in your own experiments with DevOps practices, be prepared for the occasional failure as you hone in on what works for your team.

This year we also uncovered a number of [surprises](#) in the data but you'll have to read on to find out what those are.

Teams that recognize the need to continuously improve tend to have higher organizational performance than those that don't.

02

How do you compare?



Derek DeBellis

Are you curious about how your team compares to others in the industry? This section includes the latest benchmark assessment of DevOps performance. We examine how teams develop, deliver, and operate software systems, and then segment respondents into clusters that capture the most common combinations of DevOps performance.

This year we include two different clustering approaches. The first is based on historical precedent. This clustering approach focuses on creating clusters based on four metrics that capture software delivery performance: lead time, deployment frequency, time to restore service, and change failure rate. We summarize each of these below. The goal of this approach is to help you quantify your team's current performance so that you can compare your performance to other teams.

The second clustering approach includes a fifth metric, reliability, which we use to understand operational performance. Why add a new metric to the cluster analysis? Because we have consistently seen the importance of this metric. Indeed, we have evidence that suggests that delivery performance can be detrimental to organizational performance if not paired with strong operational performance. Unlike our traditional clustering approach, this is a descriptive exercise that attempts to paint a picture of common ways teams perform across delivery and operational performance. Hence, it isn't always obvious which cluster is better.

First, let's look at a brief overview of the five measures we're using to understand software delivery and operational performance.

Software delivery and operational performance

To meet the demands of their ever-changing industries, organizations must deliver and operate software quickly and reliably. The faster your teams can make changes to your software, the sooner you can deliver value to your customers, run experiments, and receive valuable feedback. With eight years of data collection and research, we have developed and validated four metrics that measure software delivery performance. Since 2018, we've included a fifth metric

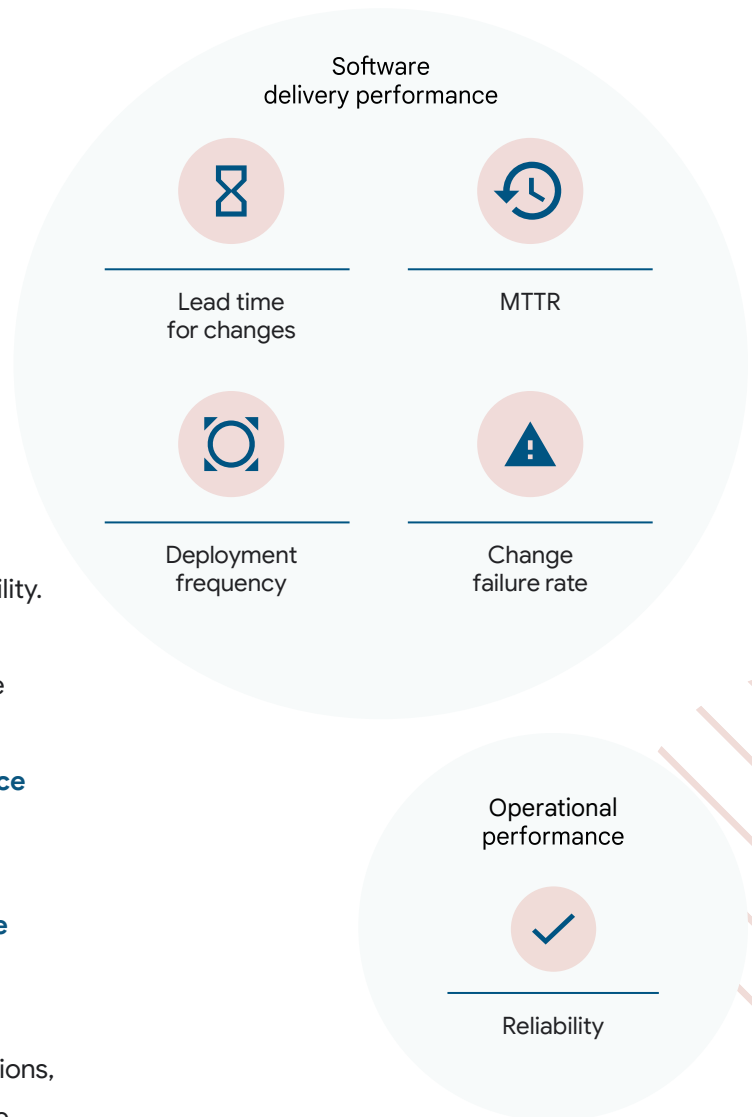
to capture operational capabilities. Teams that excel in all five measures exhibit exceptional organizational performance. We call these five measures **software delivery and operational (SDO) performance**. Note that these metrics focus on system-level outcomes, which helps avoid the common pitfalls of tracking software metrics, that may result in pitting functions against each other and making local optimizations at the cost of overall outcomes.

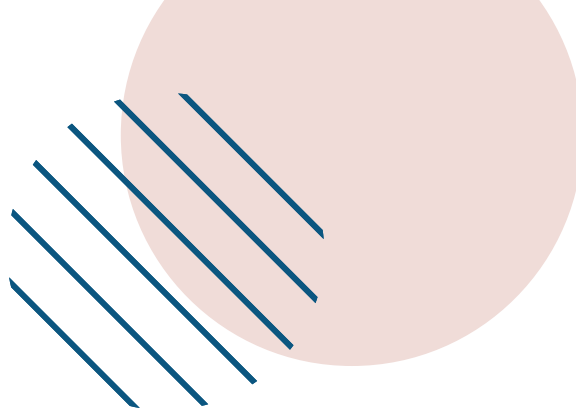


Five metrics of delivery and operational performance

The four metrics of software delivery performance can be considered in terms of throughput and stability. We measure throughput using **lead time for code changes** (that is, time from code commit to release in production), and **deployment frequency**. We measure stability using **time to restore a service** after an incident and **change failure rate**.

A fifth metric represents **operational performance** and is a measure of modern operational practices. We base **operational performance** on **reliability**, which is how well your services meet user expectations, such as availability and performance. Historically we measured availability rather than reliability, but because availability is a specific focus of reliability engineering, we expanded to measure reliability in 2021 so that availability, latency, performance, and scalability would be more broadly represented. Specifically, we asked respondents to rate their ability to meet or exceed their reliability targets. We found that teams with varying degrees of delivery performance see better outcomes (e.g., less burnout) when they also prioritize operational performance.





Historical clustering approach: clustering delivery performance

This year, when evaluating the four-cluster solution we've used since 2018, we noticed that the data showed a clear low performance cluster and a clear high performance cluster. However, the two clusters that we would traditionally use to demarcate medium performance and high performance were not differentiated enough to warrant a split. Furthermore, the various indices that we use to pick the right cluster

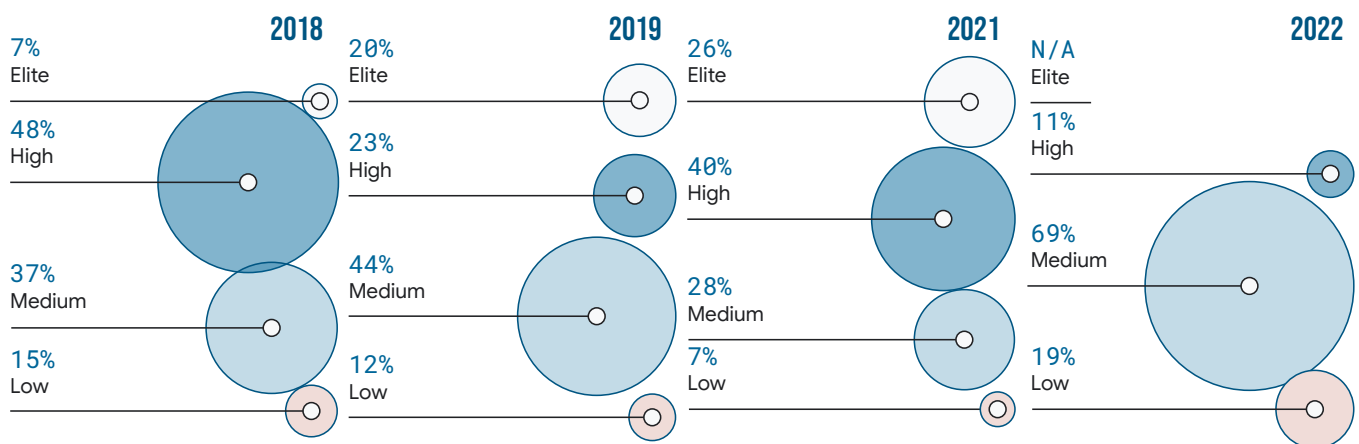
solution invariably suggested that three clusters best captured the data, regardless of the clustering techniques applied. The table below describes the delivery performance characteristics for each cluster.

The striking difference from last year is that we don't consider any cluster to be elite this year. This year's high cluster is a blend of last year's high and elite clusters. We decided to omit an elite cluster because the highest performing cluster simply isn't indicating enough of the characteristics of last year's elite cluster.

| Software delivery performance metric | Low | Medium | High |
|--|--|--|--------------------------------------|
| Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users? | Between once per month and once every 6 months | Between once per week and once per month | On-demand (multiple deploys per day) |
| Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)? | Between one month and six months | Between one week and one month | Between one day and one week |
| Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)? | Between one week and one month | Between one day and one week | Less than one day |
| Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)? | 46%-60% | 16%-30% | 0%-15% |

It suggests that this sample doesn't represent teams or organizations with employees who feel they have forged ahead. One possible hypothesis, which we currently lack the data to support, is that software development has seen reduced innovation in terms of practices, tooling, and information sharing. This could be the result of the ongoing pandemic hampering the ability to share knowledge and practices across teams and organizations. There might have been fewer opportunities to gather and learn from one another, which, in turn, might have slowed innovation. We are hopeful to do a deeper dive to explore what underpins this finding.

All that said, if you compare this year's low, medium and high clusters with last year's, you'll see that there is a shift toward slightly higher delivery performance. It appears that this year's clusters are between two of last year's. 2022's high cluster is between 2021's high and elite. 2022's low cluster seems to be between 2021's low and medium. The shift upwards for the low performance cluster suggests that while the ceiling for delivery performance has been lowered, the floor has been raised.

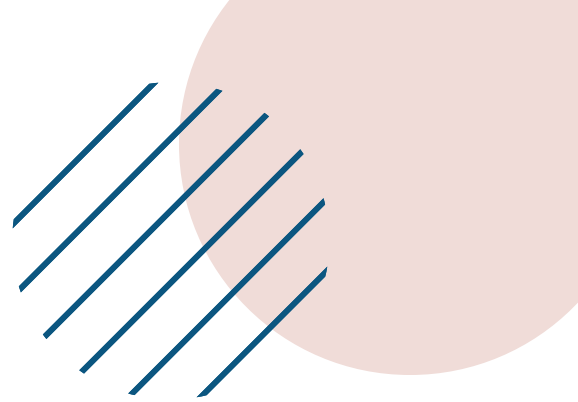


The percentage breakdowns in the table above, show that the percentage of high performers is at a 4-year low, while the percentage of low performers rose dramatically, from 7% in 2021 to 19% this year. Over two-thirds of this year's respondents fall into the medium cluster. The clear drop in high and elite performers might suggest that many of this year's respondents are in organizations or on teams that either haven't established, or are in the process of establishing, a DevOps culture that we're seeing emerge across many modern teams.

We might be focusing too much on the differences between 2021 and 2022, rather than highlighting the similarities. The clusters from 2021 and 2022 share many characteristics, including a huge separation between high performers from low performers. For example, high performers are estimated¹ to have 417x more deployments than low performers.

¹ See "Methodology" section to understand what is behind this estimate





Clustering delivery performance and operational performance

We decided to do a cluster analysis on the three categories the five metrics are designed to represent: **throughput** (a composite of lead time of code changes and deployment frequency), **stability** (a composite of time to restore a service and change failure rate) and **operational performance** (reliability). The reason for this was the pivotal role operational performance plays

in our models. For organizations that do not show solid operational performance, throughput and stability have less of an impact on organizational performance. We feel that describing the landscape of DevOps performance without accounting for operational performance leaves out a crucial part of the picture.

Exploring the data led us to a four cluster solution. Here is a breakdown of the four clusters and their names:

| Cluster | Stability | | Operational Performance | Throughput | | % respondents |
|----------|----------------------------------|---------------------|-----------------------------|----------------------------------|--|---------------|
| | Time to restore service | Change failure rate | Reliability | Lead time | Deployment frequency | |
| Starting | Between one day and one week | 31%-45% | Sometimes meet expectations | Between one week and one month | Between once per week and once per month | 28% |
| Flowing | Less than one hour | 0%-15% | Usually meet expectations | Less than one day | On demand (multiple deploys per day) | 17% |
| Slowing | Less than one day | 0%-15% | Usually meet expectations | Between one week and one month | Between once per week and once per month | 34% |
| Retiring | Between one month and six months | 46%-60% | Usually meet expectations | Between one month and six months | Between once per month and once every 6 months | 21% |

If you visited two teams in the same cluster, however, they'd likely come across as very different, and our story might not capture what you see. The story we provide for each cluster above is an attempt to use our experience working with many teams to make these patterns in the data intelligible. Further, if you visited the same team at two different points in time, it is possible that they would not have stayed in the same cluster. One possible reason for this could be that the team has either improved or degraded; another possibility is that the team has moved into a deployment pattern more appropriate for the current state of its application or service. For example, at an earlier point in an application or service's development, a team might have been focused on exploring (Starting cluster), but as they start to find their niche, they may shift their focus to reliability (Flowing cluster or Slowing cluster).

Each cluster has unique characteristics and accounts for a substantial proportion of the responses.

The **Starting** cluster performs neither well nor poorly across any of our dimensions. This cluster might be in the early stages of their product, feature, or service's development. They might be less focused on reliability because they're focusing on getting feedback, understanding their product-market fit, and more generally, exploring.

The **Flowing** cluster performs well across all characteristics: high reliability, high stability, high throughput. Only 17% of respondents are in this flow state.

Respondents in the **Slowing** cluster do not deploy too often, but when they do, they are likely to succeed. Over a third of responses fall into this cluster, making it the largest and most representative of our sample. This pattern is likely typical (though far from exclusive) to a team that is incrementally improving, but they and their customers are mostly happy with the current state of their application or product.

And finally, the **Retiring** cluster looks like a team that is working on a service or application that is still valuable to them and their customers, but no longer under active development.



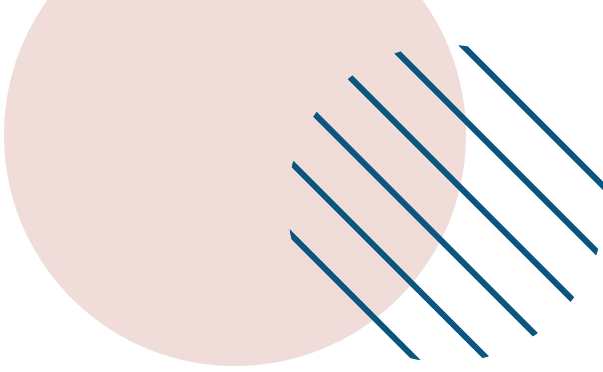
These clusters differ notably in their practices and technical capabilities. Given the high levels of software delivery and operational performance demonstrated by the **Flowing** cluster, we decided to look into how they differ from the other clusters in their practices and technical capabilities. We found that relative to other clusters, the **Flowing** cluster focuses more on:

- Loosely-coupled architectures: the extent teams can make large-scale changes to the design of their system without depending on other teams to make changes in their systems
- Providing flexibility: how flexible a company is with regard to employee work arrangements
- Version control: how changes to application code, system configuration, application configuration, etc. are managed
- Continuous integration (CI): how frequently branches are integrated into the trunk
- Continuous delivery (CD): capabilities focused on getting changes into production safely, sustainably and efficiently

Curiously, the **Flowing** cluster tends to focus less on documentation. Last year, we found that documentation practices are central to both delivery performance and operational performance (SDO). How does the **Flowing** cluster have strong SDO performance without a heavy focus on documentation? For one, there are many routes to strong SDO performance outside of documentation. Further, perhaps the **Flowing** cluster is continuously refactoring its code to create a more self-documenting process and, thereby, have less of a need for documents as we describe them in the survey.

The **Slowing** cluster, which makes up the highest proportion of our respondents, tends to be made up of respondents from larger organizations that tend to be less cloud-native than other clusters. Picture a very mature company with some calcified processes that, at the end of the day, still provides end-users with stable and reliable (and valuable) experiences. This cluster exhibits a performance oriented, generative culture.² One of the Slowing cluster's most interesting characteristics is that it has low throughput and high positive-work-culture (a "generative" work culture, according to Westrum); this is an uncommon combination. It's more common to see proportional throughput and work culture (high/high or low/low). We hope to conduct future research to better understand the relationship between throughput and culture.

² (Westrum)



We also looked at how these different clusters compare on three outcomes: burnout, organizational performance, and unplanned work. What we found broke our expectations. The **Retiring** cluster outperformed the other clusters in organizational performance. Looking at the characteristics of this cluster (poor stability and poor throughput) this is seemingly at odds with most of DORA's previous findings. But instead of blaming randomness for creating an anomaly (highly possible), we want to explore some possible explanations.

When trying to unpack these findings, there is an important complementary finding to keep in mind. The **Retiring** cluster achieves high organizational performance at a great cost: its teams have the highest burnout rates, feel the most susceptible to errors, and are burdened with the most unplanned work. In tandem, these results suggest that reliability might be enough to achieve high organizational performance, but without speed and stability, your team will pay the cost of burnout and unplanned work.

We have other hypotheses to explain why the **Retiring** cluster has higher organizational performance than the other clusters, particularly the **Flowing** cluster. Here is a quick list.

- There are features that underlie these four clusters that may not be in our data. For example, organization size may be a decent proxy for maturity. The **Flowing** cluster tends to be from smaller companies, which may indicate their products are in more formative stages.
- The members of the **Flowing** cluster tend to be from smaller companies, which may be less bound by historical processes and infrastructure and, as a consequence, have more sophisticated DevOps processes in place. That said, the data show that an organization's size is positively correlated with organizational performance for reasons that may be largely unrelated to technology.
- The **Flowing** cluster tended to disregard the principles described in Westrum's generative culture. We have seen that this is often to the detriment of organizational performance.
- Organizations in each cluster may differ on what they mean by reliability expectations and how they monitor them. The same goes for how they define their organizational performance goals.

- The **Retiring** cluster may have high short-term organizational performance, but we wonder how they would perform long-term. Will the burnout result in turnover? Will they be able to scale their processes?
- We are asking the questions at varying levels. The technical capabilities (for example, loosely-coupled architecture) are asked at the level of the team; the organizational performance questions are asked at the level of the organization. Organizations often have many teams, and it is possible for the respondent to recognize that while their organization is functioning well, the team they're working on isn't.

Further, the **Flowing** cluster scores second highest on organizational performance behind the **Retiring** cluster, and has some of the lowest levels of burnout and unplanned work. As demonstrated by both the **Flowing** and **Slowing** clusters, a DevOps philosophy is most effective when reliability is in place.

We're excited to continue exploring new ways to describe variations within the industry. Going forward, we want to continue to include operational performance as a relevant dimension in our understanding of these variations. We also want to avoid highly prescriptive and evaluative clusters (e.g., Elite), and instead concentrate on the simply descriptive exercise of identifying common constellations of SDO performance.



03

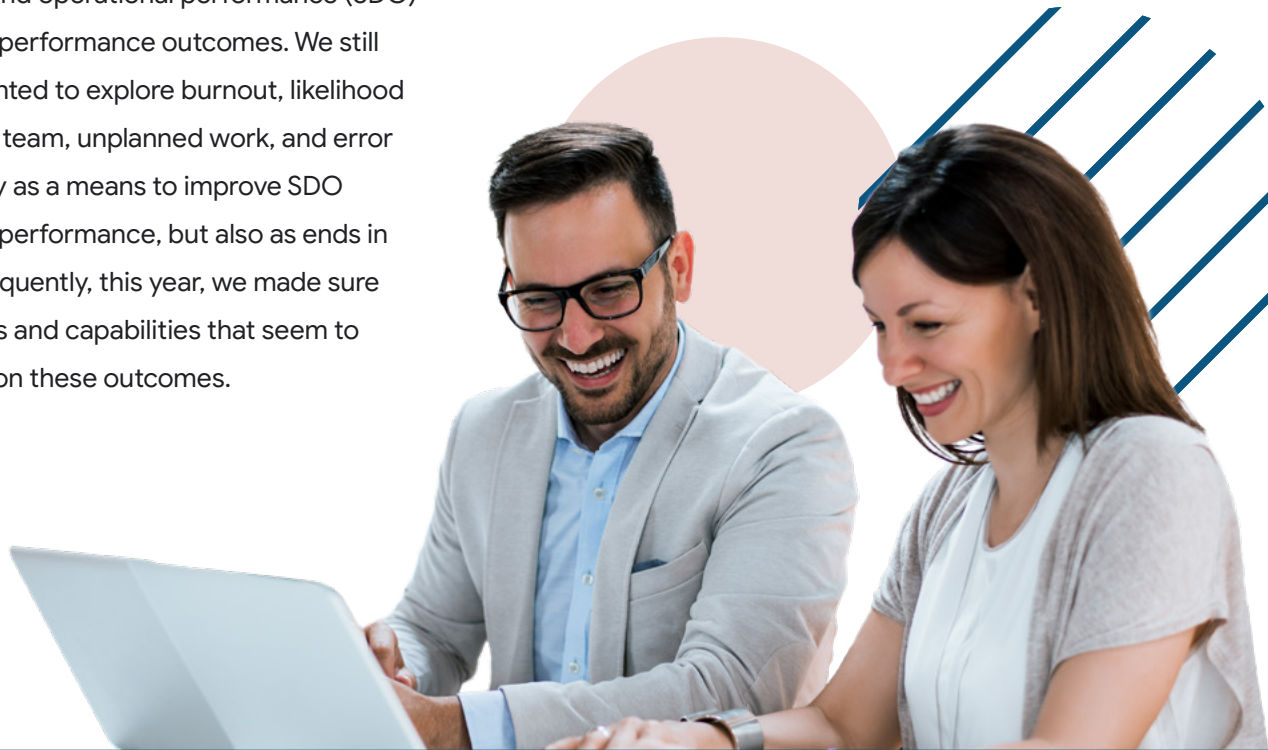
How do you improve?



Derek DeBellis

How do you improve across a multitude of outcomes?

The State of DevOps report is designed to provide evidence-based guidance to help your team focus on the DevOps practices and capabilities that get to the outcomes you care about. This year we expanded our investigation into both security and the set of outcomes teams desire. In the past, we focused on software delivery and operational performance (SDO) and organizational performance outcomes. We still do, but we also wanted to explore burnout, likelihood to recommend the team, unplanned work, and error proneness, not only as a means to improve SDO and organizational performance, but also as ends in themselves. Consequently, this year, we made sure to call out practices and capabilities that seem to exert an influence on these outcomes.





The research model shifted this year to better reflect a theory underlying DORA: there isn't a one-size-fits-all approach to DevOps. In practice, we've found that making recommendations involves understanding a team's broader context. A practice that is beneficial to one team might be detrimental to another team. For example, we have long hypothesized that technical capabilities (such as loosely-coupled architecture, trunk based development, version control, and continuous integration) have a more pronounced positive impact on software delivery performance when continuous delivery is in place. This year we explicitly modeled this and other interactions. The goal is to enhance our understanding from simply "what has an effect on what?" to include "under what conditions do these effects exist, get amplified, or get attenuated?" Understanding all this conditionality has proven to be a complicated and mind-bending endeavor, but we're excited to share with you some of these early findings.

You can find this year's and previous years' research models online on [our website](#).

Beyond the four keys

How do DORA metrics improve the performance of development and operations? A cross-functional team of software engineers at Liberty Mutual Insurance regularly reviews performance using DORA's ["four keys" metrics](#). For example, Jenna Dailey, Sr. Scrum Master at Liberty Mutual, shared that a squad leveraged DORA's research to help the team identify a bottleneck, move towards a test-driven development approach, and realize improvements in their overall performance.

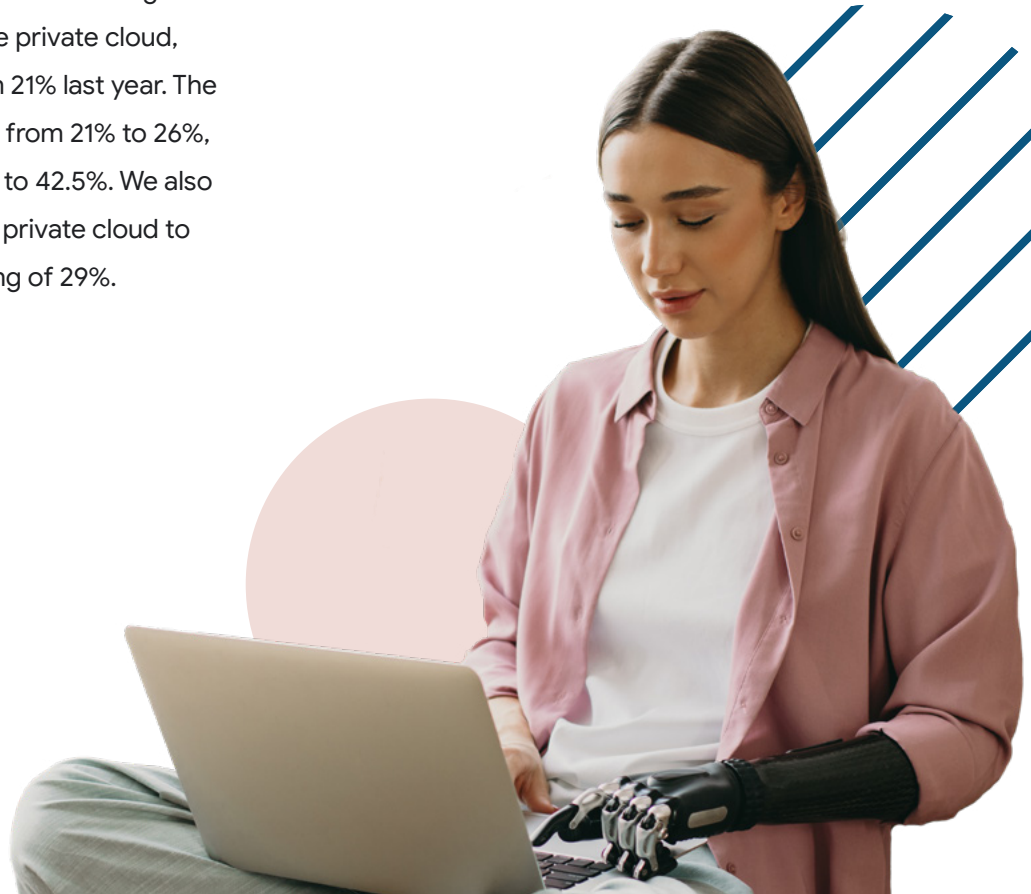
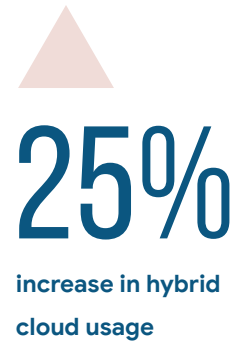
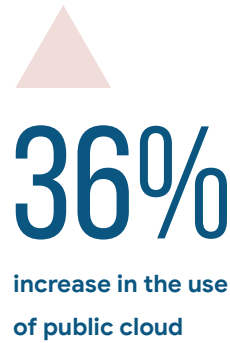
Learn more about Liberty Mutual's approach to leveraging data and DORA metrics to improve the quality and delivery of software in their recent [Tomorrow Talks](#).



Eric Maxwell

Cloud

Building on the momentum we have seen over the past several years, the use of cloud computing continues to accelerate. In fact, the percentage of people reporting the use of public cloud, including multiple clouds, is now 76%, up from 56% in 2021. The number of people reporting no cloud usage at all, including those that do not use private cloud, dropped to just 10.5%, down from 21% last year. The use of multiple public clouds rose from 21% to 26%, and hybrid cloud usage is up 25% to 42.5%. We also saw a small increase in the use of private cloud to 32.5%, up from last year's reporting of 29%.



The use of cloud computing has a positive impact on overall organizational performance. Respondents that used cloud were **14% more likely to exceed** in organizational performance goals than their non-cloud using peers.

As we have shown in previous years, and continue to validate in this report, the use of cloud computing has a positive impact on overall organizational performance. Respondents that used cloud were **14% more likely to exceed** in organizational performance goals than their non-cloud using peers. Our research shows that cloud computing enables teams to excel at things like software supply chain security and reliability and *those things* lead to organizational performance.

Surprisingly, users of all types of cloud – public, private, hybrid, and multi – showed a negative association with change failure rate, meaning an increased change

failure rate. This warrants further investigation. Instead of speculating on the reasons for this, we’ll investigate further in future research. But with few exceptions, the use of cloud-native applications (applications that were originally designed and architected for the cloud) stood out with positive signals on everything we surveyed.

Use of any cloud computing platform, public or private, positively contributes to culture and work environment outcomes (for example, generative culture, lower burnout, more stability, and higher employee satisfaction). Cloud users scored 16% higher on these cultural outcomes.

Cloud usage continues to accelerate YoY

| | 2022 | % change over 2021 |
|-------------------------------|--------|--------------------|
| Hybrid cloud | 42.47% | 25% |
| Public and/or Multiple public | 76.08% | 36% |
| Private Cloud | 32.55% | 12% |
| No Cloud | 10.55% | -50% |



The use of hybrid and multi-cloud (and private) seems to have a **negative** impact on software delivery performance indicators – MTTR, lead-time, and deployment frequency – **unless** respondents had high levels of **reliability**.

Hybrid and multi-cloud drive organizational performance

We continue to see strong signals that the use of hybrid cloud and multiple public clouds has a positive impact on organizations. Practitioners who used multiple clouds showed a 1.4x higher organizational performance compared to non-cloud users. However, use of hybrid and multi-cloud (as well as private cloud) seem to have a negative impact on several software delivery performance indicators (MTTR, lead-time, and deployment frequency) unless respondents also have high levels of reliability. This finding further speaks to

the importance of a robust SRE practice and the role reliability plays in software delivery.

In 2021 we asked respondents to tell us their *primary* reason for utilizing multiple public clouds, whereas in 2022 we asked participants to tell us all the benefits they realize from utilizing multiple cloud providers. Availability was the number one most reported benefit, which coincides with the attention and focus we have seen in the industry around reliability — you can not have reliable services unless they are available. Over 50% of practitioners reported leveraging the unique benefits of different cloud providers.

Benefits realized by adopting multiple cloud providers

| | |
|---|--------|
| Availability | 62.61% |
| Leverage unique benefits of each provider | 51.59% |
| Trust is spread across multiple providers | 47.54% |
| Disaster recovery | 43.48% |
| Legal compliance | 37.97% |
| Negotiation tactic or procurement requirement | 19.13% |
| Other | 4.06% |

> 50%

of respondents reported using multiple cloud providers.

The use of the five characteristics of cloud computing is a crucial beginning to a long causal chain that leads to organizational performance.

The five characteristics of cloud computing

In keeping with our previous research approach, we sought not simply to learn if participants were using cloud computing technologies, but how they're using cloud computing technologies. We achieved this by asking about the five essential characteristics of cloud computing, as defined by the National Institute of Standards and Technology (NIST).

On-demand self-service – Consumers can provision computing resources as needed, automatically, without any human interaction required on the part of the provider.

Broad network access – Capabilities are widely available and consumers can access them through multiple clients such as mobile phones, tablets, laptops, and workstations.

Resource pooling – Provider resources are pooled in a multi-tenant model, with physical and virtual resources dynamically assigned and reassigned on demand.

The customer generally has no direct control over the exact location of the provided resources, but can specify a location at a higher level of abstraction, such as country, state, or data center.

Rapid elasticity – Capabilities can be elastically provisioned and released to rapidly scale outward or inward with demand. Consumer capabilities available for provisioning appear to be unlimited and can be appropriated in any quantity at any time.

Measured service – Cloud systems automatically control and optimize resource use by leveraging a metering capability at a level of abstraction appropriate to the type of service, such as storage, processing, bandwidth, and active user accounts. Resource usage can be monitored, controlled, and reported for transparency.

This report validates the previous three years of DORA research, concluding that the presence of these five characteristics in an organization positively affects software delivery and operations performance. We also found that these characteristics

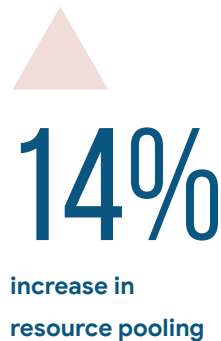


lead to better organizational performance by setting processes in motion that affect the organization in positive ways. Exhibiting the five characteristics of cloud computing is the first step in a long journey that leads to higher organizational performance.

In 2022 we see that teams are increasingly taking advantage of cloud computing differentiators. For the 4th year in a row, we are seeing growing adoption of the five characteristics of cloud computing. Resourcer Pooling saw the largest increase of 14% and Rapid Elasticity, which was the second most used feature last year, saw the smallest rise of 5%.

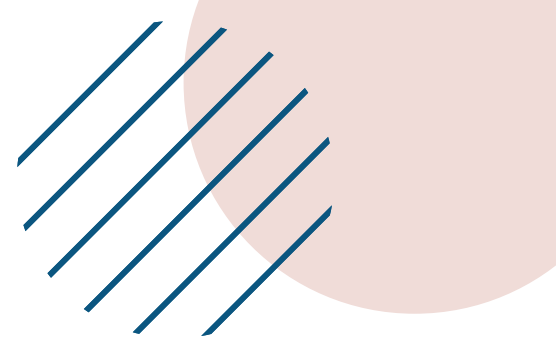


| NIST | 2021 | 2022 | Percent change |
|------------------------|------|------|----------------|
| Broad network access | 74% | 80% | 8 |
| Rapid elasticity | 77% | 81% | 5 |
| On-demand self-service | 73% | 78% | 7 |
| Measured service | 78% | 83% | 7 |
| Resource pooling | 73% | 83% | 14 |





Dave Stanke



SRE and DevOps

A successful technology team contributes more to their organization than shipping code — more, even, than shipping quality code. They also ensure that the services they deliver remain available, performant, and otherwise consistent with users’ expectations over time. *Reliability* is a multi-faceted measure of how well a team upholds these commitments, and this year we continued our explorations into reliability as a factor in software delivery and operations.

Site Reliability Engineering (SRE) is an influential approach to operations which originated at Google and is now practiced in many organizations. SRE prioritizes empirical learning, cross-functional collaboration, extensive reliance on automation, and the use of measurement techniques including Service Level Objectives (SLOs). Other modern operations practices employ similar methods but apply different naming conventions. Therefore, to assess the extent of these practices as objectively as possible, our survey takes care to use neutral, descriptive language in the text that we present to respondents. We also collect data on the outcomes of reliability engineering: the extent to

which teams are able to achieve their reliability targets. Both inputs and outputs — SRE practices and reliability outcomes — are reflected in our predictive model alongside other DevOps capabilities.

Reliability is essential

SRE adoption is widespread among the teams we surveyed: a majority of respondents use one or more of the practices we asked about. Across this breadth of teams, the data reveal a nuanced relationship between reliability, software delivery, and outcomes: when reliability is poor, software delivery performance does not predict organizational success. However, with better reliability, we begin to see the positive influence of software delivery on business success.

Without reliability, software delivery performance doesn’t predict organizational success.

Investment in SRE yields improvements to reliability, but only once a threshold of adoption has been reached.

This phenomenon is consistent with the use of the SRE “error budget” framework: when a service is unreliable, users won’t benefit from pushing code faster into that fragile context.

As Site Reliability Engineers have long asserted, reliability is the most important “feature” of any product. Our research supports the observation that keeping promises to users is a necessary condition in order for improved software delivery to benefit the organization.

Acknowledge the J-Curve

What challenges await you on the path to achieving reliability? In their O’Reilly publication “Enterprise Roadmap to SRE,”¹ DORA survey contributors James Brookbank and Steve McGhee reflect on their experiences of implementing SRE in established organizations, and recommend “acknowledging the J Curve of change.” Previously described in the 2018 State of DevOps Report, the “J Curve,” is a phenomenon in which organizational transformations tend to exhibit early success, followed by periods of diminished returns,

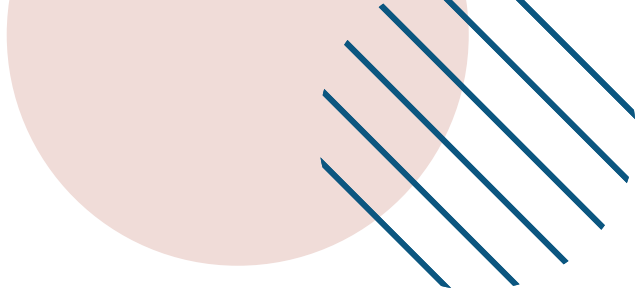
or even regressions. Those who persist through these challenges, however, often experience renewed and sustained levels of elevated achievement.

Our research this year reveals a J Curve pattern across the technology teams we studied: when teams engage in fewer reliability engineering practices — suggesting they are earlier in their journey of adopting SRE — these practices don’t predict better reliability outcomes. However, as teams adopt more SRE, they reach an inflection point where the use of SRE starts to strongly predict reliability, and in turn, organizational performance.



¹ <https://sre.google/resources/practices-and-processes/enterprise-roadmap-to-sre/>

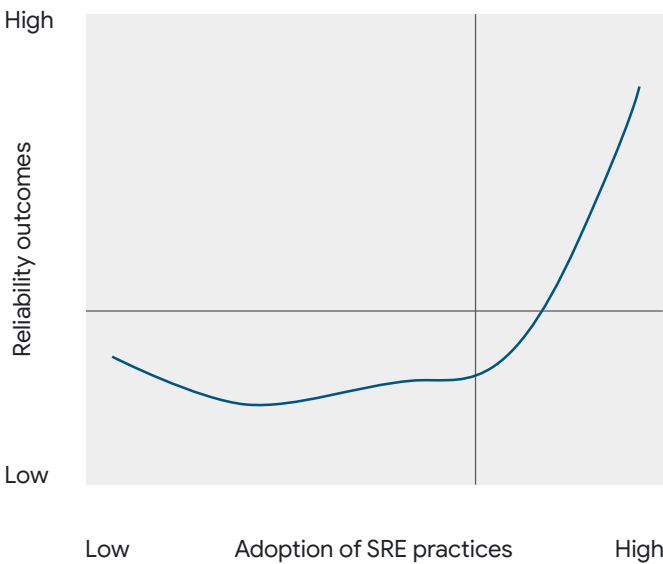
Dependable teams make dependable services: generative team culture predicts better reliability.



Teams that are in the early stages of a journey toward an SRE practice should be prepared for setbacks along the way. It can be a long road as culture, process, and tooling all realign to new guiding principles. But they can be assured that, with time and continued investment, success is likely.

Investing in people, process, and tooling

Reliability is a human endeavor, and in many ways the SRE approach exemplifies this. One of SRE's core principles is that user perception, as opposed to internal monitoring data, is the true measure of reliability. So it's perhaps unsurprising that reliability is driven by positive team dynamics. We found that teams with a "generative" culture, one that exhibits trust and collaboration, are more likely to practice SRE, and more likely to achieve good reliability outcomes. Stable teams, whose membership is consistent across time, also deliver greater reliability for user-facing services. And, as with DevOps as a whole, reliability engineering efforts benefit from augmenting human efforts with process and tooling. Practices such as the use of cloud computing and continuous integration are predictive of better reliability outcomes.



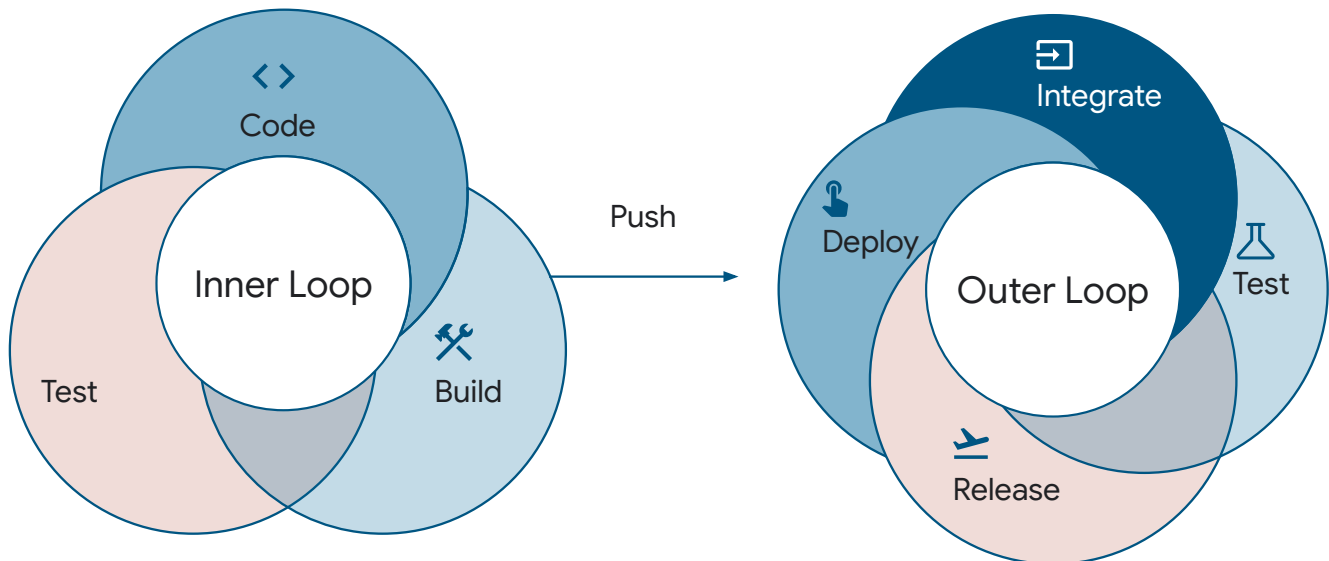
Teams that persist beyond initial steps of SRE adoption see increasing improvement in reliability outcomes

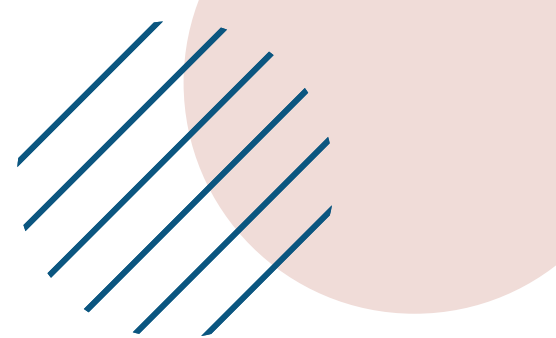


Eric Maxwell

Technical DevOps Capabilities

This year, we looked at a variety of technical capabilities to understand the outcomes that are driven by different technical practices. We considered two broad phases of software development: the “inner loop,” which comprises developer tasks such as coding, testing, and pushing to version control, and the “outer loop,” which includes activities such as code merge, automated code review, test execution, deployment, and release.





High performers who meet reliability targets are **1.4x more likely to use CI.**

Our research shows that companies that excel in inner and outer loop development are able to ship code faster and with higher levels of reliability. The capabilities that contribute most to high performance are version control, continuous integration, continuous delivery, and loosely-coupled architecture.

High performers who meet reliability targets are:

33% more likely to use version control

39% more likely to practice continuous integration

46% more likely to practice continuous delivery

40% more likely to have systems based on a loosely-coupled architecture

In fact, respondents who make higher-than-average use of all of the above capabilities have **3.8x higher organizational performance** than those who do not use these technical capabilities.

Continuous integration

Continuous Integration, often referred to as CI, is a part of the outer loop development process that automatically builds an artifact and runs a series of automated tests for every code commit in an effort to assess whether code is ready to be deployed. This process provides quick, automated feedback to the developer, allowing them to operate with higher levels of confidence. CI is a key part of taking code from a developer's workstation to production. As in previous years, CI is shown to drive delivery performance. **High performers who meet reliability targets are 1.4x more likely to use CI than others.**

This year we dove a bit deeper into the other part of the outer loop development process: continuous delivery, which we will describe in a later chapter. But first, let's take a look at a complementary component to continuous integration: trunk-based development.

Trunk-based development

Trunk-based development is the practice of continuously merging code into the trunk and avoiding long-lived feature branches. This practice is considered a complement to continuous integration and has been shown for years to accelerate software delivery velocity.

Due to the shift in demographics this year around years of experience on the job, we are able to see that experience matters when implementing trunk-based development. Last year we had 40% of respondents state they had 16+ yrs on the job and this year that category represented just 13%. Continuing to validate our “Delivery Depends” theme, we see that folks with less experience overall have less positive results around trunk-based development and see:

- ▼ **Decreased** overall software delivery performance
- ▲ **Increased** amounts of unplanned work
- ▲ **Increased** error-proneness
- ▲ **Increased** change failure rate

Individuals with 16+ years of experience that use trunk-based development realize the benefits of the practice and see:

- ▲ **Increased** overall software delivery performance
- ▼ **Decreased** amounts of unplanned work
- ▼ **Decreased** error-proneness
- ▼ **Decreased** change failure rate

This is likely due to the additional practices required to successfully implement Trunk-based development. Teams that do not have rigorously enforced rules around never walking away from a broken trunk or that do not use gated code branches and auto-roll back code that breaks the trunk will certainly experience pain when trying to develop on the trunk.

However, the presence of trunk-based development shows a positive impact on overall organizational performance.



Frank Xu

Continuous Delivery

Continuous delivery (CD) is a software development practice that:


1. Enables the team to deploy software to production or end users at any time
2. Ensures the software is in a deployable state throughout its lifecycle, including when working on new features
3. Establishes a fast feedback loop that enables the team to check the quality and deployability of the system, and prioritizes fixing issues blocking deployment.

Note that continuous delivery does not necessarily imply continuous deployment, the practice in which every software build is automatically deployed. Continuous delivery requires only that a software build can be deployed at any time.

Last year, we examined the technical DevOps capabilities that predict the likelihood that a team practices CD, and discovered that factors such as loosely-coupled architecture and continuous testing / integration were among the strongest predictors. This year, in addition to examining the factors driving the use of CD, we analyzed and identified the effects of CD alone as well as its interactions with other DevOps capabilities on development outcomes.

CD drives software delivery performance

Similar to previous years' findings, the use of CD is a predictor of higher software delivery performance, both alone and in combination with other DevOps capabilities. Teams that rated higher on CD are more likely to have higher frequency of deploying code to production, and shorter lead time for changes and service restoration.



Teams that combine version control and continuous delivery are 2.5x more likely to have high software delivery performance than teams that only focus on one.


In addition, respondents are 2.5x more likely to report higher software delivery performance when their team also adopts version control practices.

CD can increase unplanned work

The data suggested that continuous delivery leads to developers spending more time on rework or unplanned work. A hypothesis for this finding is that developers are more likely to build applications iteratively when there are tighter feedback loops. As a result they may view some iterative changes as unplanned work on the same part of the system. This work may be simultaneously unplanned and driven by feedback from a previous deployment.

Technical practices and CD

Our research has regularly shown that a broad set of technical capabilities support CD. This year, we explored what happens when some of these individual capabilities are used in conjunction with CD. We found that trunk-based development and loosely-coupled architecture, together with CD, may have a negative impact on a team's performance. For example, we see evidence that teams who are adopting loosely-coupled architectures and CD together are 43% more likely to anticipate more than average error proneness (i.e., product outages, security vulnerabilities and significant performance degradation to happen to their services), compared to teams that only adopted CD. These effects require further investigation and point to some potential friction for teams who are improving. This friction may be related to the J-curve of transformation wherein teams realize early improvements but then falter as they move beyond the low-hanging fruit. Commitment to improvement is required to realize its full potential. When improving any capability, such as CD, be sure to watch after the effects on the team and overall performance.





David Farley

Loosely-coupled Architecture

Loosely-coupled systems are important to the effectiveness of teams and organizations. This doesn't just apply to cloud- or microservice-based systems — it has to do with an organization's ability to make change. The ease with which an organization can safely and confidently change its software is a marker of the software's quality.

With a loosely-coupled architecture teams can:

- Make large-scale changes to the design of their system without having to depending on other teams to make changes in their systems
- Get faster feedback through independent, on-demand testing with lower coordination costs
- Deploy code with negligible downtime

In this year's report, we asked respondents to describe whether or not the software they build is based on a loosely-coupled architecture. The results were intriguing, and showed a variety of mostly positive associations between the presence of loosely-coupled architecture and teams' performance across multiple dimensions.



The benefits of a loosely-coupled architecture

Teams that focus on building software with loosely-coupled architectures are in a better position to perform strongly across stability, reliability, and throughput. These teams are also more likely to recommend their workplace to a friend or colleague.

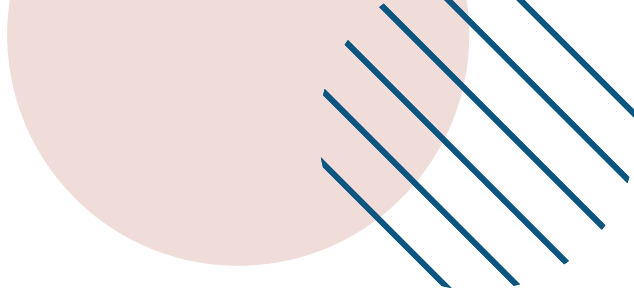
It's common to see software that uses a loosely-coupled architecture from teams that are deploying to the cloud and adopting a microservices architectural approach,

Teams that focus on building software with loosely-coupled architectures are in a better position to perform strongly across stability, reliability, and throughput.

managing hundreds of services. However, loose coupling is more than a simple measure of the count of services in a system. Components in a loosely-coupled architecture can be deployed independently. This independence allows teams to develop, test, and deploy their services without expensive coordination overhead between teams.

In the real world, loose coupling is not restricted to one architectural style; fundamentally, it's the ability to make a change in one part of the system, without that change impacting other parts. This allows organizations to divide up their work, so that individual teams can make progress without having to coordinate with other teams.

In our experience, teams that require deep integration testing with other services as a way to build confidence in their software before it's deployed have not yet achieved loose coupling; to do so, these teams would benefit from improving the interfaces and isolation between systems. One effective way to improve interfaces and isolation is by improving the 'testability' of services and components. If your design allows you to test your service in isolation, then its interface is, by definition, loosely-coupled.



We also found that cohesive, stable teams that use loosely-coupled architecture are more likely to use software development practices that encourage and support continuous improvement. For example, SRE practices such as setting reliability goals to prioritize work or performing regular reviews to revise reliability targets based on evidence, both support loosely-coupled architecture.

Loosely-coupled architectures also allow the organization to more easily add employees, as independent teams that don't need to coordinate with other teams are freer to increase the size of their teams independently.

In short, loose coupling of software services impacts more than just technical impact. It also affects the socio-technical aspects of software development. Coupling is at the root of Conway's Law — the idea that an organization's design systems mirror their own communication structure. More loosely-coupled systems mean more loosely-coupled organizations with a more distributed, scalable, approach to development.

Surprising findings

This year's research revealed that loosely-coupled architecture might contribute to burnout on teams. This is a surprising finding that contradicts findings from previous years. Our analysis shows that stable teams where information flows freely have lower levels of burnout. Westrum's generative culture and team stability both support loosely-coupled architecture and decrease burnout, so this is clearly contradictory. More research is required before we can draw definitive conclusions.

At the same time, when security requirements are defined and controlled by a consolidated security organization, it may be more difficult for teams to decouple their software from other teams. This further demonstrates the benefit of shifting security concerns to the team that is most responsible for the application (see also: [Why supply chain security matters](#)). This is one of the more subtle forms of coupling in organizations, and though we have collected the data on security, this is likely to be equally true of other centralized functions. Allowing teams to make their own decisions on security, and other frequently centralized functions, is one way to make progress toward reaping the benefits that using loosely-coupled architecture can impart to your organization.





Daniella Villalba

Culture

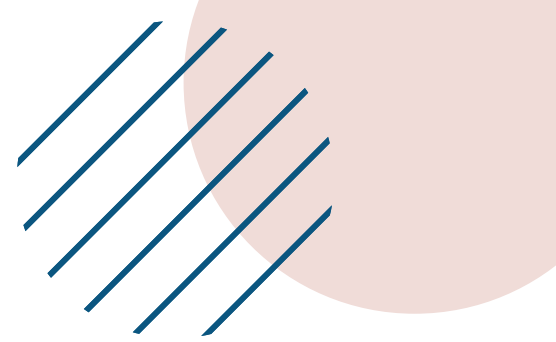
“Well, this is how things are done around here.”

People have likely uttered this phrase countless times and across a wide range of industries to describe their organization’s approach to challenges and opportunities.

Every organization has its own unique culture, and our research has consistently shown that culture is foundational to an organization’s success and the well-being of its employees.

Culture is also a necessary aspect of DevOps since, at the most basic level, DevOps is about tools, practices, and *how people work together to develop and deliver software quickly, reliably, and safely*. Understanding the factors that impact an organization’s culture can help leadership tackle culture-related challenges head-on. Therefore, fostering a healthy culture should be a priority for organizations. If left unaddressed, these culture-related challenges may hinder DevOps practices from taking hold.





This year we continued to use Westrum’s organizational typology to measure the health of an organization’s culture. In addition, we expanded our understanding of culture by measuring team churn, flexible work arrangements, perceived organizational buy-in, and burnout.

Data from this year’s research support previous findings that **organizational performance** is impacted by the type of culture that exists within an organization. Specifically, a **generative culture** is associated with higher levels of organizational performance compared to organizations characterized by a bureaucratic or

pathological culture. Employees at organizations with a generative culture are more likely to belong to stable teams, produce higher-quality documentation, and spend most of their time engaged in meaningful work.

Team Churn

We investigated team churn and found that **stable teams** — teams whose composition hadn’t changed much over the last 12 months, were more likely to exist within high-performing organizations. Constant churn can impact productivity and morale as new team members need time to onboard. And those who stay

Westrum Organizational culture

| Pathological <i>Power oriented</i> | Bureaucratic <i>Rule oriented</i> | Generative <i>Performance oriented</i> |
|--|---|--|
| Low cooperation | Modest cooperation | High cooperation |
| Messengers are “shot” | Messengers are neglected | Messengers are trained |
| Responsibilities shirked | Narrow responsibilities | Responsibilities are shared |
| Bridging discouraged | Bridging tolerated | Bridging encouraged |
| Failure leads to scapegoating | Failure leads to justice | Failure leads to inquiry |
| Novelty crushed | Novelty leads to problems | Novelty implemented |



might need to adapt to changes in their workload and team dynamics. In addition, our research showed that stable teams were more likely to report producing quality documentation compared to teams that experienced more churn. A team that is constantly dealing with change may have a harder time keeping up with practices that lead to quality documentation.

High performing organizations are more likely to have flexible work arrangements.

Flexible work arrangements

Given the shift to flexible work arrangements that many organizations have adopted since the outbreak of the COVID-19 pandemic, we investigated whether giving employees the freedom to choose between remote,

in-person, or hybrid options was associated with higher organizational performance. Findings showed that organizations with higher levels of **employee flexibility** have higher organizational performance compared to organizations with more rigid work arrangements. These findings provide evidence that giving employees the freedom to modify their work arrangements as needed has tangible and direct benefits for an organization.

Burnout

Burnout is a feeling of dread, apathy, and cynicism surrounding work. When people experience burnout, they are not just unmotivated and exhausted, they are also more likely to have lower job satisfaction, which can increase employee turnover. Burnout has been linked to a wide range of poor psychological and physical health outcomes such as increased risk for depression and anxiety, heart disease, and suicidal thoughts¹.

Last year we measured burnout in the context of the COVID-19 pandemic and found that a generative culture was associated with lower rates of employee burnout.

¹ Maslach C, Leiter MP. Understanding the burnout experience: recent research and its implications for psychiatry. World Psychiatry. 2016 Jun;15(2):103-11. doi: 10.1002/wps.20311. PMID: 27265691; PMCID: PMC4911781.

Flexible work models are associated with decreases in employee burnout and increases in employees' likelihood of recommending their team as a good place to work.

This year we replicated this finding and expanded our understanding of burnout by showing that stable teams and flexible work arrangements are also associated with less burnout. In addition, this year we measured team Net Promoter Score (NPS), which indicates whether people would recommend their team to a friend or colleague. We found that team NPS was associated with perceived leadership buy-in. And mirroring the burnout findings, we found that a generative culture, a stable team, and a flexible work arrangement are associated with people being more likely to recommend their team to others.

How employees perceive their organization

Lastly, we investigated perceived leadership buy-in by asking people to predict how much support they expected their team to receive over the next 12 months. Results showed that higher perceived leadership buy-in (for example more financial support, more allocation of resources, sponsorships) was associated with high performing organizations.

We also asked people to predict the likelihood that a security breach or a complete outage would occur over the next 12 months. Results showed that people working at high performing organizations were less likely to expect a major error to occur - they had a more positive outlook on their organization. Similarly, we found that people working in organizations with high software and delivery performance were **less likely** to feel that their current practices needed to be changed to improve business outcomes.

A few words on representation

Our findings showed that employees from underrepresented groups were more likely to report spending more time on unplanned work regardless of whether they belong to high or low performing organizations. We also found that employees from underrepresented groups reported higher levels of burnout compared to employees who do not belong to underrepresented groups. Team leads should be aware of the risk for workload imbalance and ensure work is allocated fairly among team members.

Taken together, these findings underscore the importance of creating a healthy and inclusive environment for employees both at the organizational and team level.

While we continue to emphasize the importance of culture, we acknowledge that changing or even improving an organization's culture is no easy task. We recommend that organizations seek to first understand their employees' experiences and subsequently invest resources in addressing culture-related issues as part of DevOps transformation efforts.



04

Why supply chain security matters



John Speed Meyers



Todd Kulesza

In November 2020, relatively few technology professionals suspected that a software supply chain security crisis was brewing. The [Open Source Security Foundation](#), a successor to past efforts, had been founded to focus on open source software security, and while there were a few [bright spots](#) towards addressing this problem, the topic was not on the front page of major newspapers. A major attack, [SolarWinds](#),

changed all of that. When attackers can silently penetrate thousands of major companies and government networks on the back of trojan software updates, the times change fast.

Today, the topic of software supply chain security has become widely recognized as urgent — if not over family dinner, certainly in the boardroom.



There are numerous initiatives, and large parts of the software industry have committed to reforming their own software supply chain security practices and improving the security of the open source commons.

In this chapter, we focus on two initiatives: Supply Chain Levels for Software Artifacts ([SLSA](#), pronounced “salsa”), and the NIST Secure Software Development Framework ([SSDF](#)). Each offers a range of defensive measures to make sure that attackers can’t tamper with software production processes and sail past network defenders via malicious software updates.

But how widely used are the software supply chain security practices associated with SLSA and SSDF? Which practices need help driving adoption, and which are already in widespread use? To date, there were no systematic answers to these questions. By surveying hundreds of software professionals about their use of practices associated with supply chain security, we provide some early answers. In particular, four main findings stand out:

01 Adoption has already begun: Software supply chain security practices embodied in SLSA and SSDF already see modest adoption, but there is ample room for more.

02 Healthier cultures have a head start: Organizational culture is a primary driver of software development security practices, with higher trust, “blameless” cultures are more likely to establish SLSA and SSDF practices than lower-trust organizational cultures.

03 There’s a key integration point: Adoption of the technical aspects of software supply chain security appears to hinge on the use of CI/CD, which often provides the integration platform for many supply chain security practices.

04 It provides unexpected benefits: Besides a reduction in security risks, better security practices carry additional advantages, such as reduced burnout.

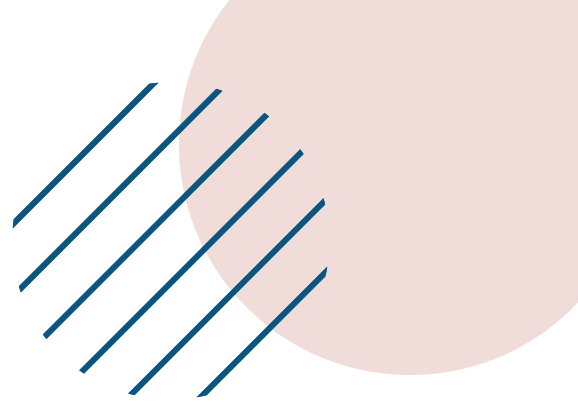
What companies do today to avoid security vulnerabilities

To better understand what organizations are doing today to identify and resolve security vulnerabilities in the software they build, we added over two dozen questions to this year's survey. These questions broadly fell into two categories:

- Questions that asked respondents to agree or disagree with a statement (for example, “My organization has an effective method for addressing security threats” or “I have access to the necessary tooling to execute security tests”).
- Questions that asked respondents how established or not established security practices are at their organization (for example, “Builds are defined through build scripts and nothing else” or “Production releases are built by using a centralized CI/CD system, never on a developer's workstation”). We used the “established/not established” scale because early tests found that respondents were biased towards agreeing with some security-related questions. The questions about SSDF, however, were more naturally phrased with the agree/disagree response scale.

The **SLSA framework**, at v0.1 at the time of writing, describes a series of software supply chain integrity practices associated with SLSA “levels,” with higher levels corresponding to higher levels of software supply chain security assurance. We asked respondents about many of the particular practices associated with SLSA. In particular, the survey asked, “How established are the following practices for the primary application or service you work on?” Table 1 lists the wording of the SLSA-related practices covered in the survey.

The **SSDF**, currently at v1.1, focuses on practices to help organizations ship software with fewer vulnerabilities, and to minimize the potential impact of remaining vulnerabilities. Instead of SLSA's “levels,” SSDF practices are grouped into four categories: preparing the organization, protecting the software being developed, producing well-secured software, and responding effectively to discovered vulnerabilities. The survey asked respondents how much they agree (or disagree) with statements describing several SSDF practices; these questions are summarized in Table 2.



| SLSA Practice | Survey Definition |
|-------------------------------|--|
| Centralized CI/CD | Production releases are built by using a centralized CI/CD system, never on a developer's workstation |
| History Preserved | Revisions and their change history are preserved indefinitely |
| Build Script | Builds are fully defined through the build script and nothing else |
| Isolated | Builds are isolated; they cannot interfere with concurrent or subsequent builds |
| Build Text Files | Build definitions and configurations are defined in text files stored in a version control system |
| Parameters Metadata | Build metadata (e.g. dependencies, build process, build environment) about an artifact includes all build parameters |
| Dependencies Meta-data | Build metadata (e.g. dependencies, build process, build environment) about an artifact documents all dependencies |
| Metadata Generated | Build metadata (e.g. dependencies, build process, build environment) is either generated by the build service, or by a build-metadata generator that reads the build service |
| Prevent Inputs | When running builds, build steps are prevented from loading any build inputs dynamically (i.e. all required sources and dependencies are fetched upfront) |
| Users No Edit | Build metadata (e.g. dependencies, build process, build environment) about an artifact cannot be edited by build services users |
| Metadata Available | Build metadata (e.g. dependencies, build process, build environment) is available to the people who need it (e.g. via a central database), and is delivered in a format that they accept |
| Two Person Review | Every change in a revision's history must be individually reviewed and approved by two trusted persons prior to submission |
| Metadata Signed | The build metadata (e.g. dependencies, build process, build environment) about how an artifact was produced is signed by my build service |

Table 1. SLSA-Related Survey Questions

Note: Respondents had five possible responses to each question: not established at all, slightly established, moderately established, very established, and completely established.

| SSDF Practice | Survey Definition |
|---|---|
| Security reviews | A security review is conducted for all major features on the applications I work on |
| Continuous code analysis / testing | We continuously engage in automated or manual code analysis and testing for all supported releases, in order to identify or confirm the presence of previously undetected vulnerabilities |
| Early security testing | Security tests are run early in the software development process, either by me or by another team |
| Effectively address threats | My organization has an effective method for addressing security threats |
| Integrated with development team | Security roles are integrated into our software development team |
| Documents requirements | Our org has processes in place to identify and document all security requirements for the software our organization develops or acquires (including third-party and open source) |
| Regularly reviews requirements | Security requirements are reviewed at regular intervals (annually, or sooner if required) |
| Metadata Generated | Build metadata (e.g. dependencies, build process, build environment) is either generated by the build service, or by a build-metadata generator that reads the build service |
| Integrated with development cycle | At my company, the software-security protocol is seamlessly built into our development process |
| Standard process across projects | At my company, we have a standardized process for addressing software security across projects |
| Monitor security reports | We have ongoing efforts to monitor information coming from public sources regarding possible vulnerabilities in the software we use and its third-party components |
| Have necessary tools | I have access to the necessary tooling to execute security tests |

Table 2. SSDF-Related Survey Questions

Note: Respondents had seven possible responses to each question: strongly disagree, disagree, somewhat disagree, neither agree nor disagree, somewhat agree, agree, and strongly agree.

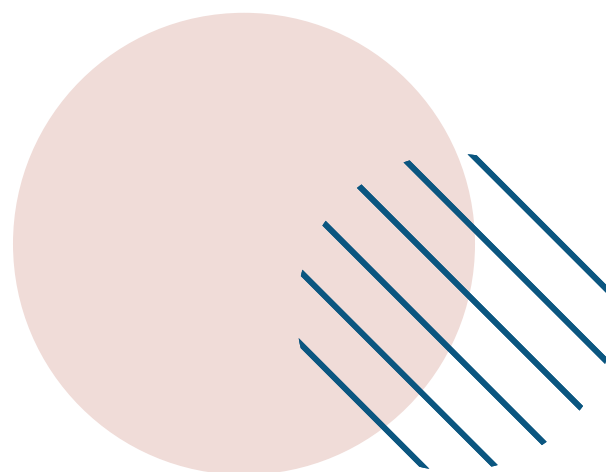


Overall, we found relatively broad adoption of emerging industry practices, though with plenty of room for these to become more established. For example, while 66% of respondents agreed with the statement, “At my company, the software-security protocol is seamlessly built into our development process,” only 18% strongly agreed. Figures 1 and 2 summarize participant responses to our security questions.

We found that using continuous integration/continuous delivery (CI/CD) systems for production releases was the most commonly established practice, with 63% of respondents saying this was “very” or “completely” established. That CI/CD tops this list aligns with prior security research, which found that [most organizations implement application-level security scanning as part of their CI/CD process](#). In addition, a separate set of security-focused qualitative interviews suggested that most developers were unable to run such tooling locally during development. The SLSA framework similarly builds upon CI systems as a central integration point for supply chain security. Our model analysis, described in the next section, found that the presence of CI in

an organization was a predictor of the maturity of its security practices. Thus, we believe that without this critical piece of infrastructure, it is very difficult for an organization to ensure a consistent set of scanners, linters, and tests are run against the software artifacts they create.

In addition to CI/CD, other commonly established practices included indefinite preservation of code history (60%), builds that are solely defined via scripts (58%), keeping builds isolated from one another (57%), and storing build definitions in source control (56%). On the lower end, the two least commonly established practices were requiring two or more reviewers to approve each code change (45%) and signing build metadata to prevent/detect tampering (41%).



Alongside questions about established practices, we also asked participants to agree or disagree with a set of statements about security at their organization. The statement with the highest level of agreement was, “We have ongoing efforts to monitor information coming from public sources regarding possible vulnerabilities in the software we use and its third-party components,” with 81% of respondents agreeing. On the opposite side, the statement with the lowest proportion of agreement regarded negative impacts of security practices on software development – 56% of respondents agreed that, “The software security processes that exist at my company slow down the development process for the applications I work on.”

While it’s encouraging that this had the lowest level of respondent agreement, the fact that a majority of respondents said current security processes slow down development suggests a great deal of room for improvement in security tooling and approaches. Our model analysis also supports this interpretation, showing mixed (though minor) effects on software delivery performance.



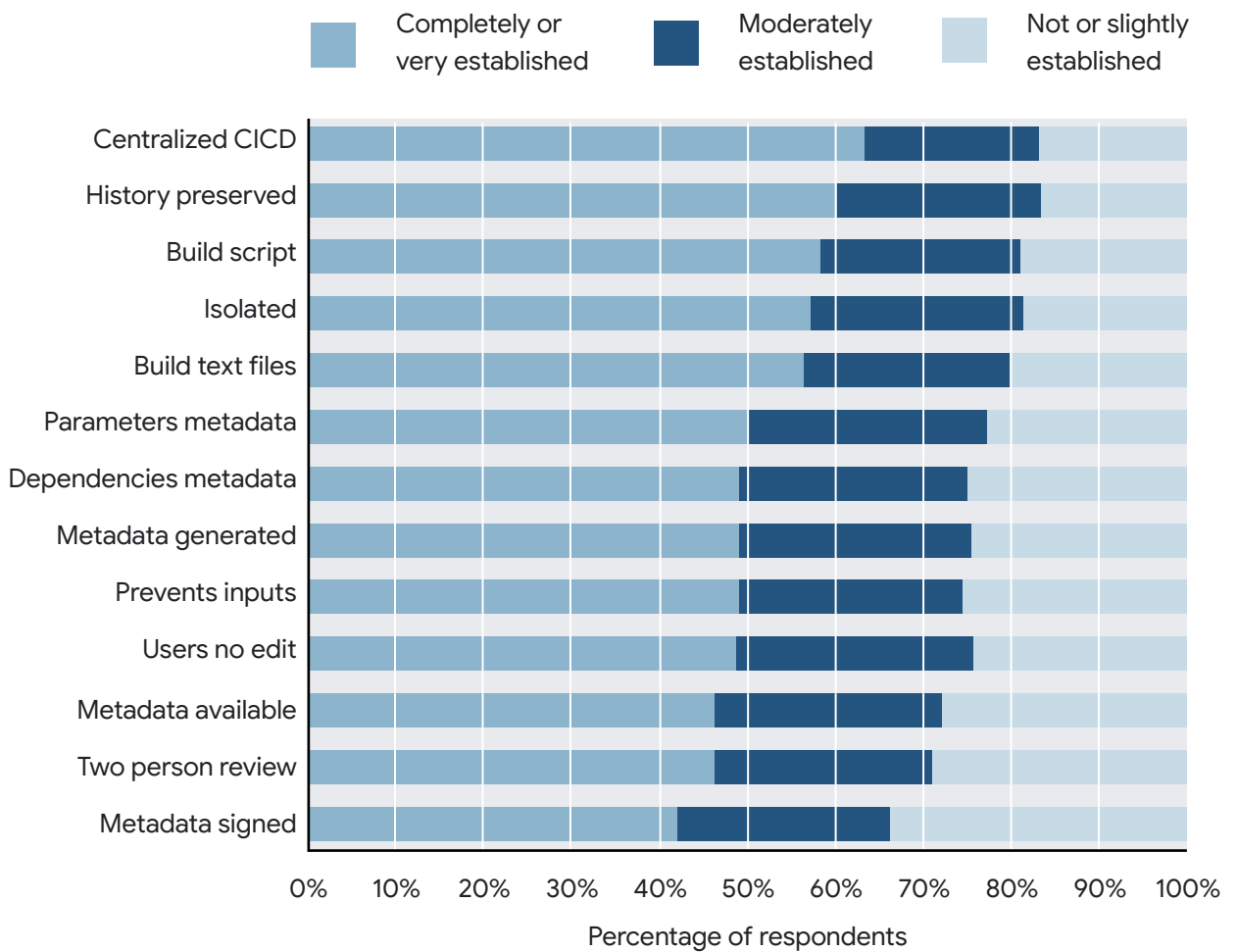


Figure 1. Establishment of SLSA practices

Survey responses about the establishment of SLSA practices. A majority of respondents indicated some establishment of all of these practices, but relatively few said they were “completely” established yet.

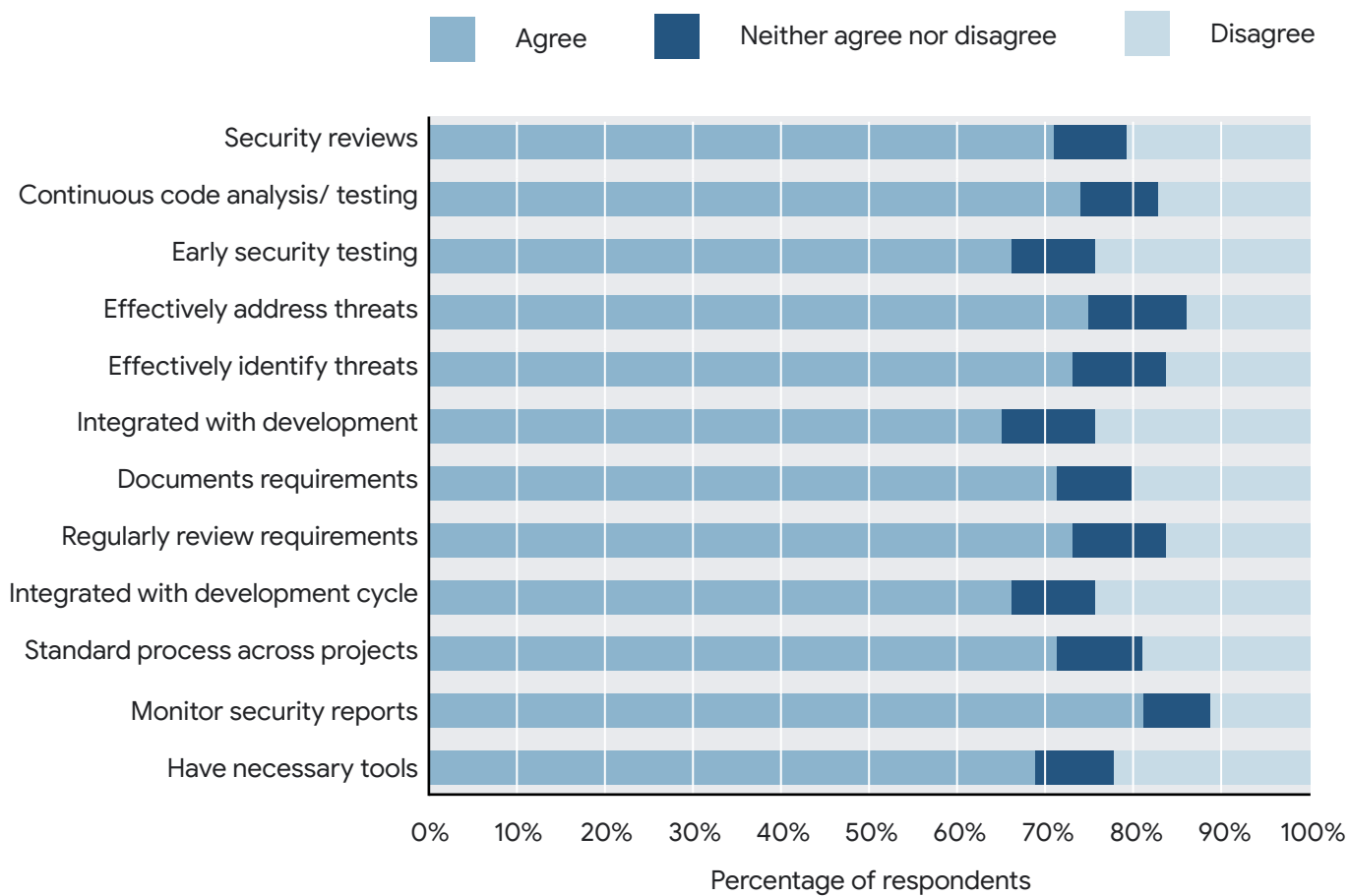


Figure 2. Establishment of SSDL practices

Survey responses about the establishment of SSDL practices. Similar to SLSA, a majority of respondents agreed that their organization followed all of these practices.

What helps companies follow good security practices?

Application security is just one aspect of software development, and thus one of many competing demands on developers' time and attention. High-friction approaches to security can be frustrating for developers and ineffective overall, as people try to avoid the friction points. For example, a set of research interviews with professional software engineers found that their touchpoints with security teams were limited to either the start or end of a project, and the teams could be difficult to engage with. In the words of one participant, "We have an application security team, but I have never had my code reviewed by them... I am like most engineers, I avoid them usually."

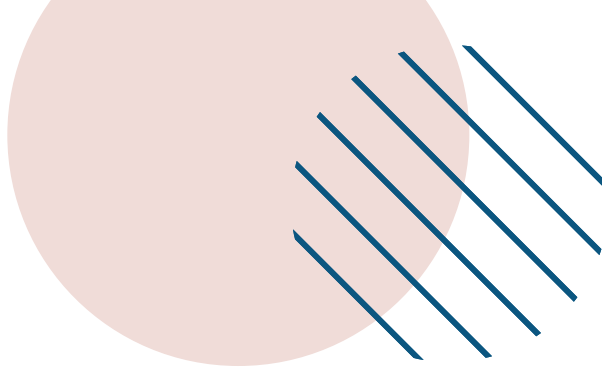
One approach to improving software security is to reduce barriers to following security practices. The developers we spoke with *wanted* to do the right thing, and often discussed frustration that shipping features or fixes consistently took priority over potential security issues. For example, one respondent to a separate security-related survey described their biggest security challenge as, "making it a priority in the first place. It's not sexy, it doesn't sell more product, [and] it's not a problem until it becomes one."

Our survey data suggest that there are several factors which make it easier for developers to, "do the secure thing."

The biggest factor we found was not technical at all, but rather cultural: **organizations closest to the "generative" Westrum culture group were significantly more likely to say they had broadly established security practices**, as defined by the SLSA framework¹. Aspects of generative cultures include being highly cooperative, sharing risks and responsibilities, and learning from past mistakes. We hypothesize these traits manifest in healthier security practices in numerous ways, such as encouraging software engineers to be more proactive about supply chain security, rewarding people for their efforts around security regardless of their job role, or reducing perceived risks of reporting potential security issues.

Technologically, three of the most important factors driving security relate to infrastructure. Intuitively this makes sense: if your infrastructure makes tasks like vulnerability scanning or manual code reviews easier to conduct, then it becomes more likely your engineers will use them. Specifically, we found that **having systems for source control, continuous integration, and**

¹ Interestingly, these same respondents were not more likely to agree with the NIST SSDF questions. While SLSA and SSDF discuss different aspects of application development security, we expected to find overlap between these sets of questions. As mentioned earlier, it's possible that the response scale for SSDF was biased towards "agreement" responses, which would explain this difference.



continuous delivery were all linked with also having more firmly established SLSA practices. A key part of this is likely *when* security issues get developer attention, which a separate survey found was primarily during CI. Typically CI directly precedes code reviews, and is when vulnerability scanners and other code analysis tools are run, as it guarantees that all code commits are subject to the same security requirements. The lack of a centralized build system makes such consistent scanning far more challenging, and the lack of source control in turn makes it challenging to have a centralized build system in the first place.

Security scanning as part of CI/CD, however, may not be early enough for software engineers. In a set of security-related interviews with application developers, they consistently told us security scanning on their development workstation would help to save time and effort. Two situations were commonly cited: 1) wanting to know in advance if they were building upon a dependency with known vulnerabilities, so they could re-evaluate using that dependency before building on top of it, and 2) avoiding long CI wait times, sometimes measured in hours, just to confirm whether their current changes resolved a security issue. In both cases, software engineers said that while a CI “backstop” was necessary, the ability to run the same

security tools locally would help them work faster and more efficiently.

The cultural and technological factors discussed above were the biggest drivers of security, but not the only ones. Other notable factors included:

- Flexibility in work arrangements (for example, does the organization support working from home?)
- Cloud use (either public or private)
- Working on a “cloud-native” application or service
- Feeling that the business values and invests in your team
- Low turnover on team
- Organization size, with larger organizations reporting higher security scores

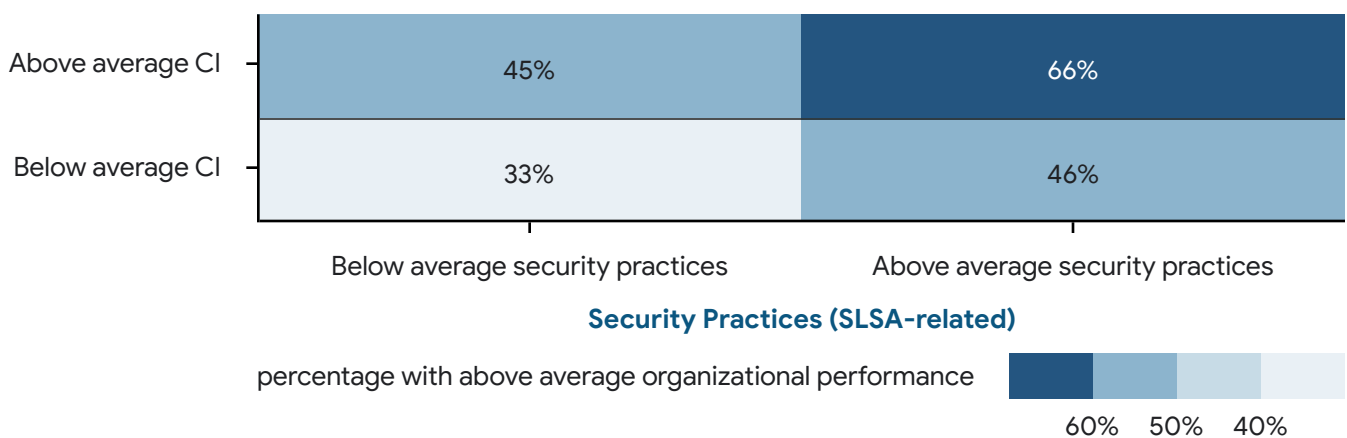
These factors, however, mostly seem to correlate with either the generative Westrum culture (for example, flexible work arrangements, feeling valued by your organization, or having low team turnover), or with CI/CD usage (for example, working on a cloud-native application, or working at a large organization). This data leads us to believe organizational culture and modern development processes (such as continuous integration) are the biggest drivers of an organization’s application development security, and the best place to start for organizations looking to increase their security posture.

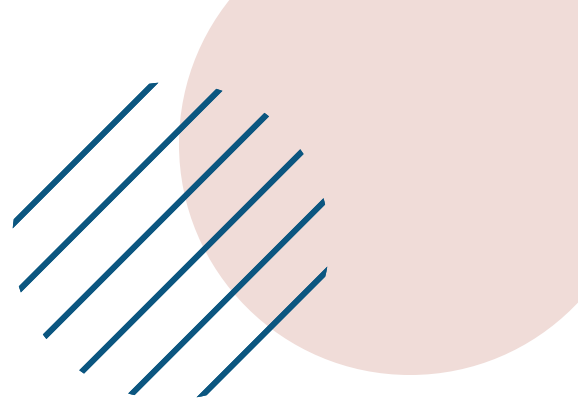
What outcomes do good security practices lead to?

As organizations improve their security practices around software development, what benefits might they expect to see? Our survey data confirms that participants anticipate a **lower chance of security breaches, service outages, and performance degradation as companies increase establishment of supply chain security practices**. Similarly, we conducted separate research in the first half of 2022, finding that running tools like vulnerability scanners during CI significantly increased the probability of identifying vulnerabilities in software dependencies: respondents who used such tools were nearly twice as likely to report identifying a security vulnerability in their own code or in one of its dependencies. In short: SLSA and SSDF practices appear to work as intended. We don't claim that they

can eliminate security threats, but our evidence suggests they do reduce an organization's security risk.

Security practices can also positively impact performance-based outcomes, but there is a twist: CI plays a pivotal role. When CI isn't implemented, security practices show no effect on software delivery performance. But when CI is implemented, security practices have a strong positive effect on software delivery performance. This essentially means that CI is necessary for security practices to positively impact software delivery performance. Further, security practices generally have a positive effect on organizational performance, and when CI is firmly established, this effect is amplified. The graph below attempts to visualize this effect.





Along with a reduction in perceived security risks, respondents also reported **less burnout** among team members and an increased willingness to recommend their organization as **a great place to work**. Both of these findings speak to the “yes, and” nature of security for software engineers: it’s one more task on their already crowded plates. Tools and processes that help them incorporate secure practices into their existing development workflow, as opposed to unplanned work or “fire drills” when a threat is discovered, provide a mechanism for reducing security risks and increasing developer joy.

Taken together, our evidence suggests that **healthy, high-performing teams also tend to have good security practices** broadly established (though, as noted earlier, there continues to be room for improvement). Following approaches such as the SLSA or SSDF frameworks might not single-handedly improve all the culture and performance metrics that we measure, but it’s clear that security need not come at the expense of other development priorities.

Tools and processes that help them incorporate secure practices into their existing development workflow, as opposed to unplanned work or “fire drills” when a threat is discovered, provide a mechanism for reducing security risks **and** increasing developer joy.

05

Surprises



Derek DeBellis

Although each year's report focuses on the corresponding year's survey responses, we do our best to understand these findings in the context of the entire catalog of State of DevOps Reports and adjacent research (for example, research on burnout and culture). Testing the reliability of these effects through replication efforts is a core tenet of the research program. This gives us an opportunity to adjust our beliefs to fit the data and understand evolving or emerging trends.

This year we ran into a few surprises. There are a myriad of potential reasons for this. For one, the sample shifted this year to include more people earlier in their careers than in previous reports. One interpretation is that we're hearing more from the people who are directly responsible for implementing the technical practices and capabilities, as opposed to people who

might be responsible for overseeing or directing the implementation of these practices. Another possibility is that something has shifted in the industry or the world; what worked yesterday isn't guaranteed to work tomorrow. Macroeconomic forces, and another year largely in the shadow of the COVID-19 pandemic, for example, may have changed the physics of DevOps. Lastly, subtle changes to what is included in our model may have changed the relationships between variables.¹



¹ Judea Pearl's "Book of Why" and Robert McElreath's "Statistical Rethinking" present us with incredible examples of how what you do and don't include in your statistical models can impact the model's output.



An unexpected or unhypothesized finding puts researchers in a tough spot when it's time to write the report. Given the risk that the finding is spurious or, at the very least, has yet to establish (or even contradicts) the empirical evidence of multiple studies, the responsible move is to do follow-up research to attempt to replicate the results and understand their cause.² By focusing on the surprises, a researcher also runs the risk of deemphasizing how many effects have reliably emerged across years of research. We statistically explore over one hundred pathways for each State of DevOps report. In doing so, we invite the risk of spurious findings simply by chance. We try to counteract that with yearly replication efforts.

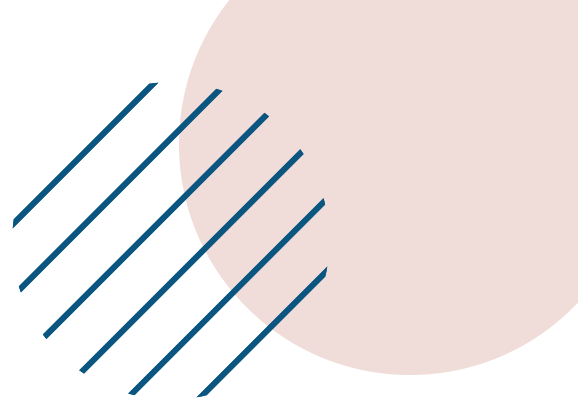
On the other hand, not reporting the finding risks creating a file drawer effect³, which effectively results in the expected or palatable becoming well-known, and the unexpected or difficult to stomach being hidden away. We seek to strike a balance: we don't want to sensationalize nascent findings, but we also feel it is crucial to share them. Here are the things that surprised us the most, and what we think they mean:

01 We have consistently observed that trunk-based development practices have a positive impact on software delivery performance. In fact, this has been observed in every year of the study since 2014. Trunk-based development capabilities were behaving out of character this year. For one, trunk capabilities had a negative impact on software delivery performance. The opposite was true in previous research reports. Given how aberrant this finding is, we are eager to see if it is replicated in upcoming research and hear if the community has any explanations.

02 We only found that software delivery performance is beneficial to organizational performance when operational performance is also high, and many respondents did not have high operational performance. This contradicts previous iterations of our research where the connection between software delivery performance and organizational performance was much clearer.

² Kerr, N. L. (1998). HARKing: Hypothesizing after the results are known. *Personality and social psychology review*, 2(3), 196-217.

³ Rosenthal, R. (1979). The file drawer problem and tolerance for null results. *Psychological bulletin*, 86(3), 638.



03 Documentation practices negatively impacted software delivery performance. This is at odds with previous reports. One hypothesis is that documentation is becoming an increasingly automated practice, especially among high-performing teams. Until we collect additional data, we have little evidence to either support or refute this belief.

04 Some tech capabilities (i.e., trunk based development, loosely-coupled architecture, CI, CD) seemed to predict burnout. As mentioned above, many respondents in this sample were notably earlier in their careers than participants in previous years' samples. Hence, we might have been talking to people responsible for implementing the capability as opposed to those responsible for creating or overseeing the initiative. The implementation process may be notably more challenging than its oversight. We want to do further research to better understand this finding.

05 Reliability engineering practices had a negative impact on software delivery performance. One explanation is that these are not necessarily causally intertwined. This year we noticed in a new clustering analysis (see "How do you compare") that a subset of clusters seemed to focus on reliability while ignoring software delivery performance. We believe that these are decoupled, in the sense that you can do one without doing the other, but ultimately, to make software delivery performance count in terms of organizational performance, reliability needs to be in place.

06 We added SLSA-related practices to understand whether teams are adopting these approaches to maintaining a secure software supply chain. While we expected there to be some association between implementation of security practices and performance (e.g., use of technical capabilities, better software delivery performance, and better organizational performance), we were surprised to see that security practices were actually the mechanism through which technical capabilities impacted software delivery performance and organizational performance.

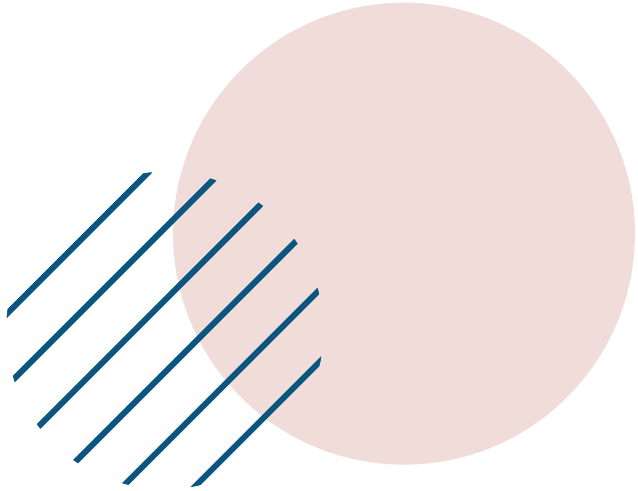




The inclusion of SLSA-related practices seems to account for much of the effect of continuous integration, version control and continuous delivery on both software delivery performance and organizational performance. Put differently, there is a causal chain that is being detected in the data where many technical capabilities have a positive impact on SLSA-related practices and through this positive impact on SLSA-related practices have a positive impact on both software delivery performance and organizational performance. We used mediation analyses to detect this result.^{4 5} This is pushing us to explore whether our measure of SLSA-related practices is tracking other features of the team (e.g., general performance) and in what way security practices lead to better software delivery performance and organizational performance.

We look forward to studying these effects again next year, to see whether we can reproduce and explain these new patterns, or whether we should lean toward dismissing them as outliers (that we should also try to explain). As always, we welcome feedback from the community.

Join the [DORA Community](http://dora.community) (<http://dora.community>) to continue the discussion about these surprises and other findings in this year's report!



⁴ Jung, Sun Jae. "Introduction to Mediation Analysis and Examples of Its Application to Real-world Data." Journal of preventive medicine and public health = Yebang Uihakhoe chi vol. 54,3 (2021): 166-172. doi:10.3961/jpmph.21.069

⁵ Carrión, Gabriel Cepeda, Christian Nitzl, and José L. Roldán. "Mediation analyses in partial least squares structural equation modeling: Guidelines and empirical examples." Partial least squares path modeling. Springer, Cham, 2017. 173-195.

06

Demographics and Firmographics



Derek DeBellis

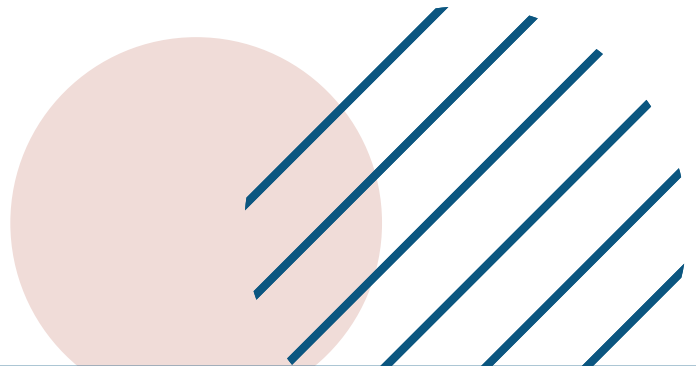
Thank you for your contributions to our research and our industry!

Who took the survey?

With eight years of research and more than 33,000 survey responses from industry professionals, the State of DevOps Report showcases the software development and DevOps practices that make teams and organizations most successful. This year, over 1350 working professionals from a variety of industries around the globe shared their experiences to help grow our understanding of the factors that drive higher performance. Thank you for your contributions to our research and our industry! In summary, representation across demographic and firmographic measures has remained remarkably consistent.

Similar to previous years, we collected demographic information from each survey respondent. Categories include gender, disability, and underrepresented groups.

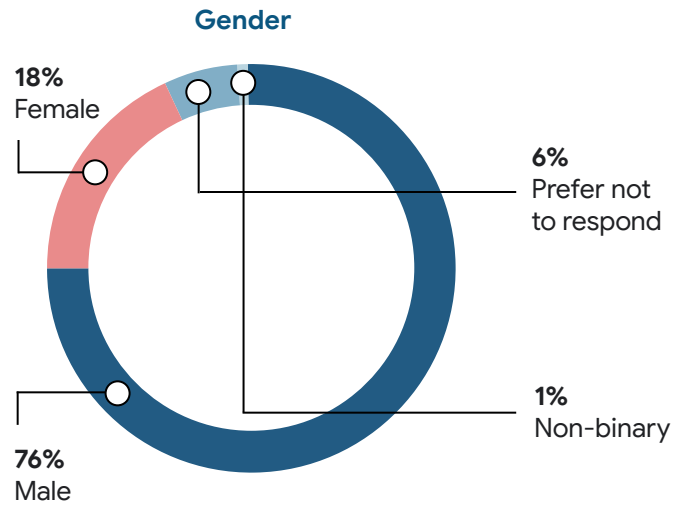
This year we saw representation that was consistent with previous reports across firmographic categories including company size, industry, and region. Again, over 60% of respondents work as engineers or managers and a third work in the technology industry. Additionally, we see industry representation from financial services, retail, and industrial/manufacturing companies.



Demographics

Gender

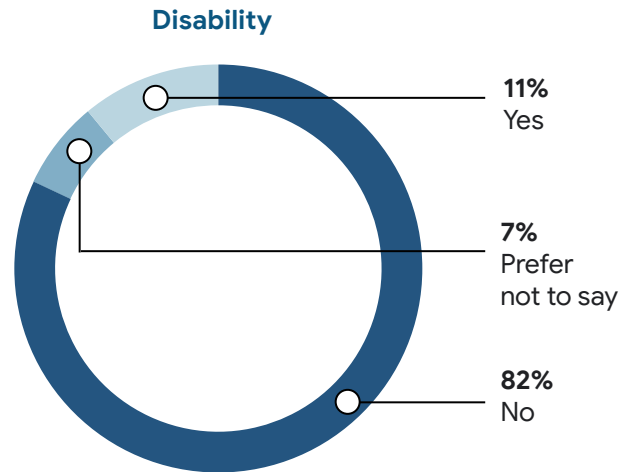
Relative to 2021, this year's sample had a higher proportion of female respondents (18% vs. 12%). The proportion of male respondents (76%) was lower than 2021 (83%). Respondents stated that women make up 25% of their teams, which is identical to 2021 (25%).



Percent women: **25% Median**

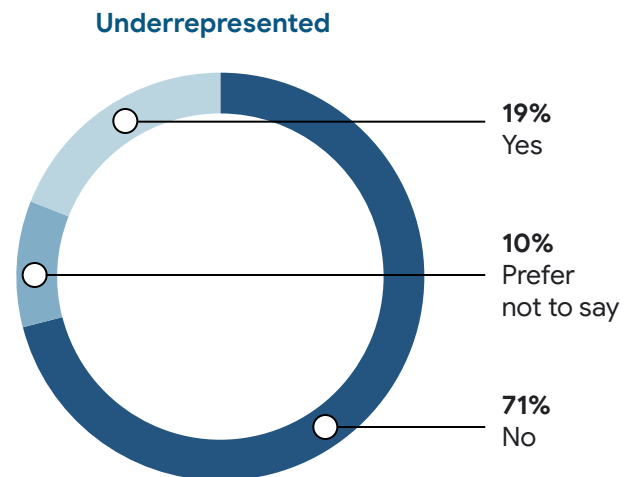
Disability

We identified disability along six dimensions that follow guidance from the [Washington Group Short Set](#). This is the fourth year we have asked about disability. The percentage of people with disabilities was consistent with our 2021 report at 11%.



Underrepresented

Identifying as a member of an underrepresented group can refer to race, gender, or another characteristic. This is the fifth year we have asked about underrepresentation. The percentage of people who identify as underrepresented has increased slightly from 17% in 2021 to 19% in 2022.

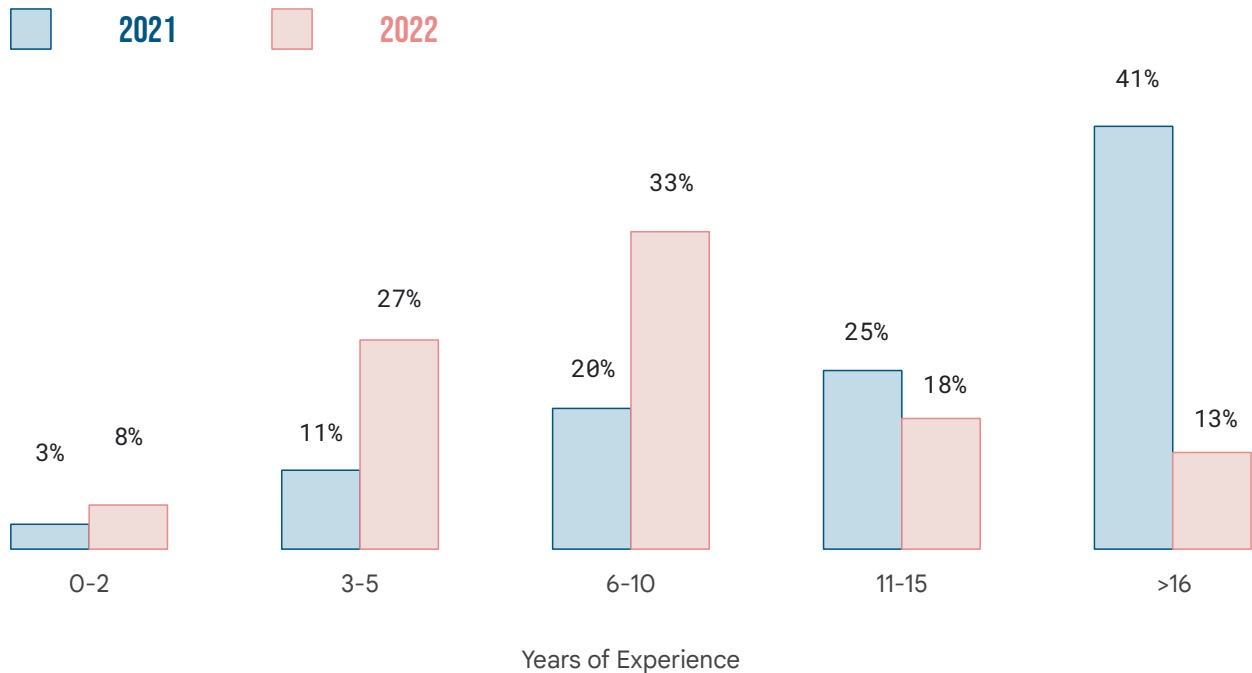


Years of experience

Notably more respondents this year had five years or less of experience (35%) than in 2021 (14%).

Perhaps unsurprisingly then, the proportion of respondents with more than 16 years of experience (13%) was a fraction of 2021 (41%).

This shift may explain some patterns that emerged in the data, and we believe is important to keep this in mind when interpreting the results, especially when comparing to last year.

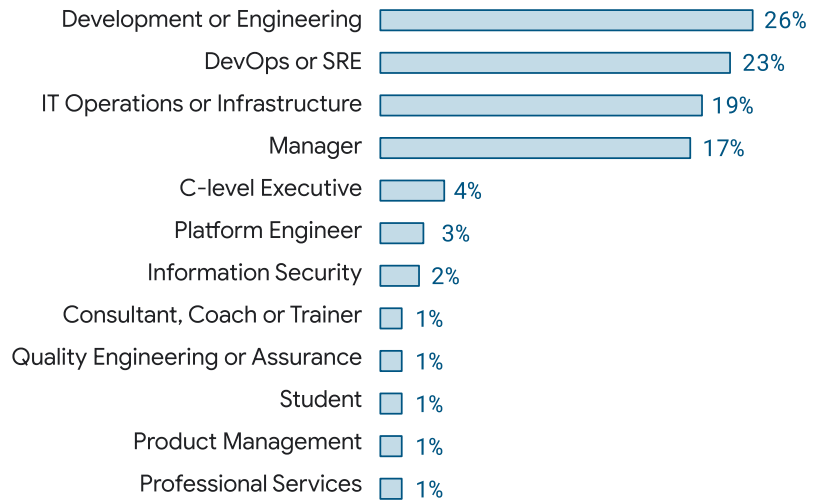




Firmographics

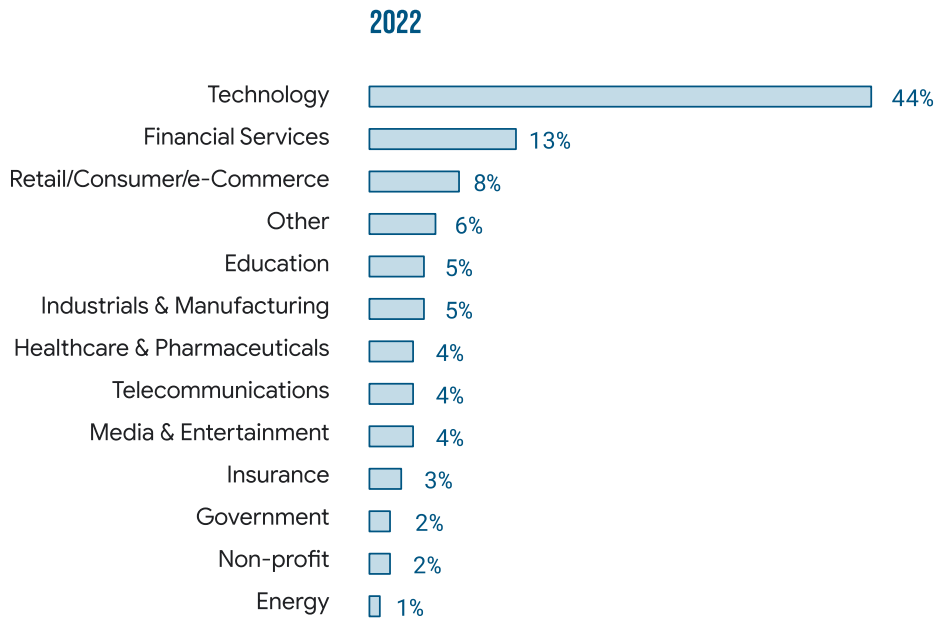
Role

85% of respondents consist of individuals who either work on development or engineering teams (26%), work on DevOps or SRE teams (23%), work on IT ops or infrastructure teams (19%), or are managers (17%). The proportion of respondents who work on IT ops or infrastructure teams (19%) more than doubled last year's proportion (9%). C-level executives (9% in 2021 to 4%), and Professional Services (4% in 2021 to 1%) are two of the more pronounced decreases relative to last year.



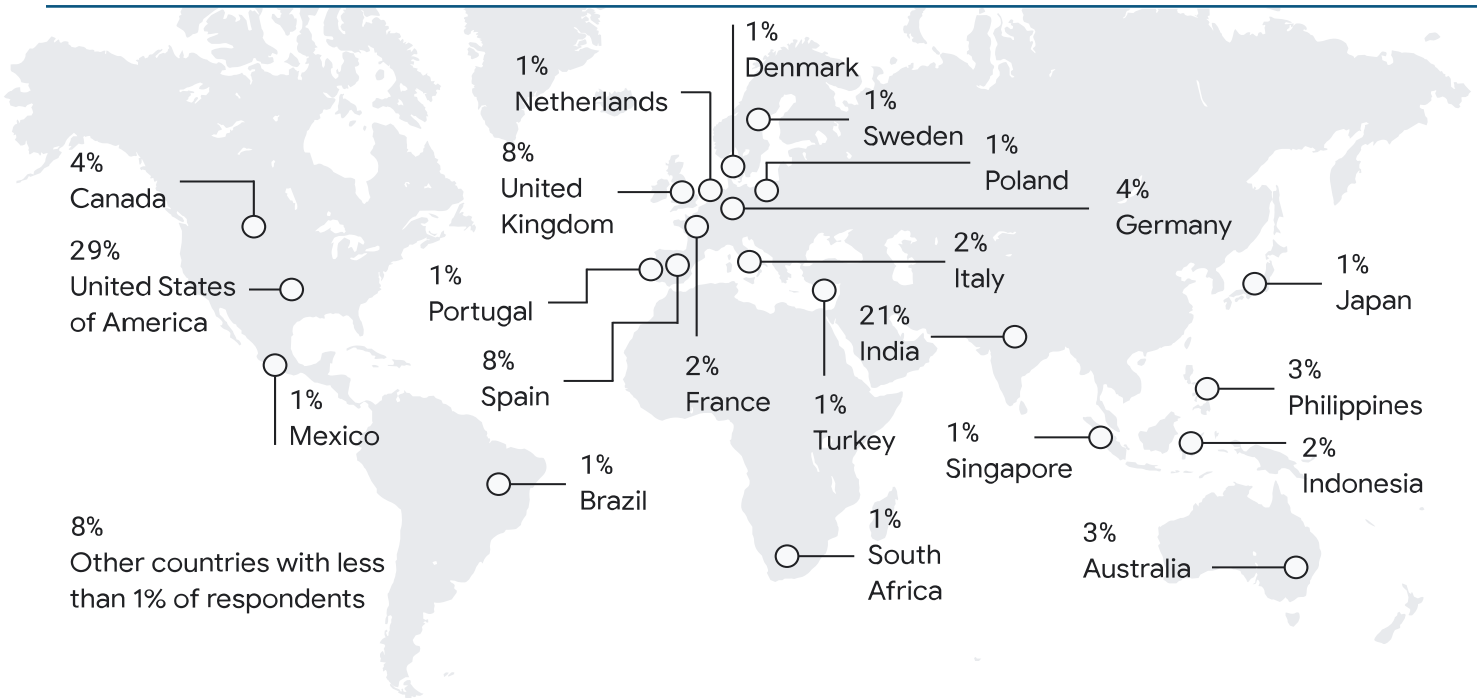
Industry

As in previous State of DevOps reports, we see that most respondents work in the technology industry, followed by financial services, other, and retail.



Region

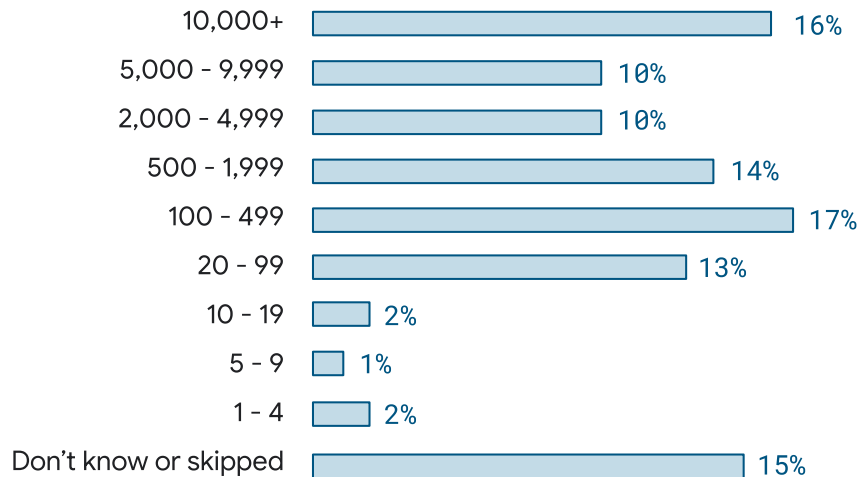
This year we asked respondents to select the country they were from instead of the region. Region, which was frequently represented by a continent, seemed a bit too coarse to understand the makeup of our respondents. We received responses from participants in over 70 countries; 89% of respondents came from 22 countries.

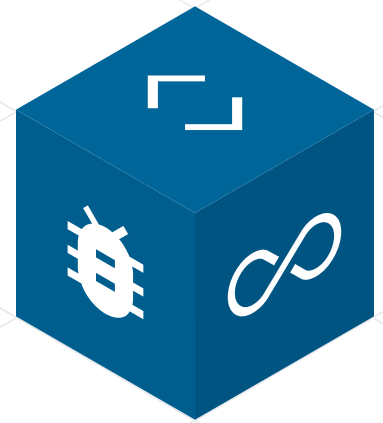


Number of employees

Consistent with previous State of DevOps surveys, respondents come from organizations of a variety of sizes. 22% of respondents are at companies with more than 10,000 employees and 7% are at companies with 5,000–9,999 employees. Another 15% of respondents

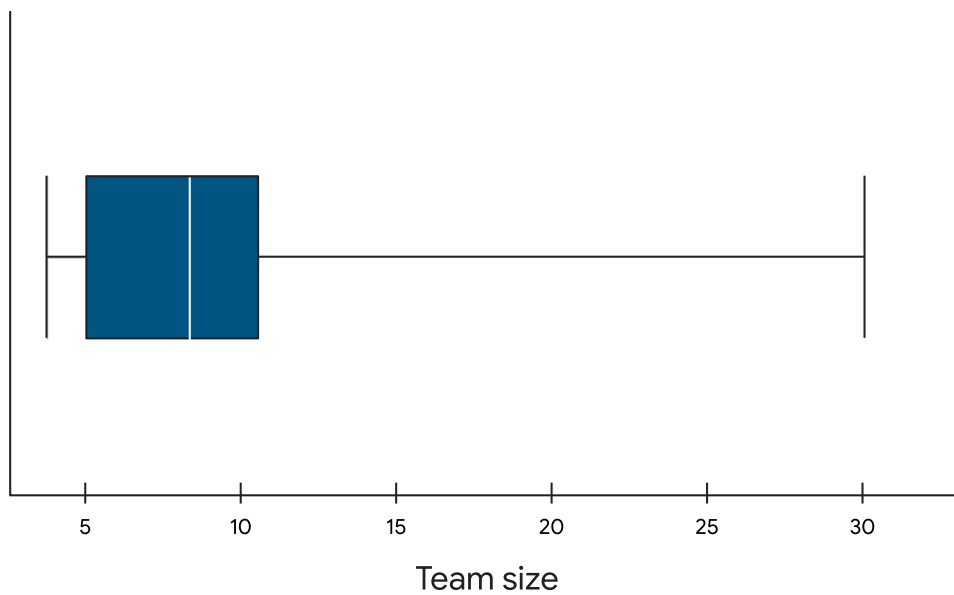
are at organizations with 2,000–4,999 employees. We also saw a fair representation of respondents (13%) from organizations with 500–1,999 employees, 15% with 100–499 employees, and finally 15% with 20–99 employees. This year, we also allowed respondents to select “I don’t know” about their organization’s size; 15% of respondents either reported not knowing or skipped the question.





Team size

This year we had participants indicate the approximate number of people on their team. 25% of respondents worked on teams with 5 people or fewer. 50% of respondents worked on teams with 8 people or fewer. 75% of respondents worked on teams with 12 people or fewer.

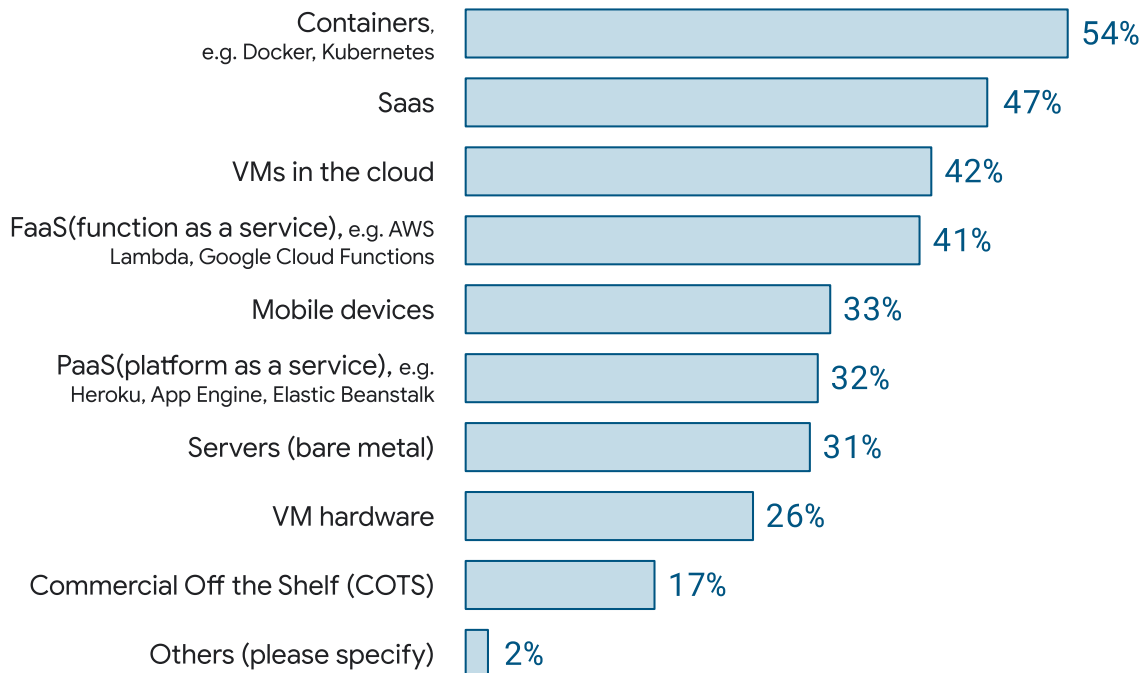


Deployment target

2021 was the first year we decided to look at where respondents deployed the primary service or application they work on. To our surprise, the number one deployment

target was containers. This year was no different, although at a lower proportion (54%) than last year (64%).

We also added more options in the hope of giving respondents more ways to reflect where they deploy.



07

Final thoughts



Derek DeBellis

Each year that we produce this report, we strive to provide a rigorous account of how practices and capabilities lead to business-critical outcomes such as organizational performance. We evaluate the replicability of many effects in previous reports, and extend the scope of our work to account for emerging priorities in the DevOps space. This year we shaped our survey and analyses to do a deep dive on security practices and altered our statistical modeling approach to explore the conditionality or dependencies of certain effects. We also explored new ways to describe the landscape of software delivery and operational performance.

In many ways, the narrative that materialized this year is an echo of previous years: technical capabilities build upon each other to create better performance; there are many benefits inherent in the use of cloud; workplace culture and flexibility lead to better organizational performance; and employee burnout prevents organizations from reaching their goals. The analysis of interactions that we explicitly added to the model helped us understand the conditions under which certain

effects can take place. For example, software delivery performance only seems to have a positive impact on organizational performance when operational performance (reliability) is high, which leads to the conclusion that you need both to thrive as an organization. There were also a few surprises, which we highlighted in a dedicated section.

We thank everyone who contributed to this year's survey. We hope our research helps you and your organization build better teams and better software — while also maintaining work-life balance.

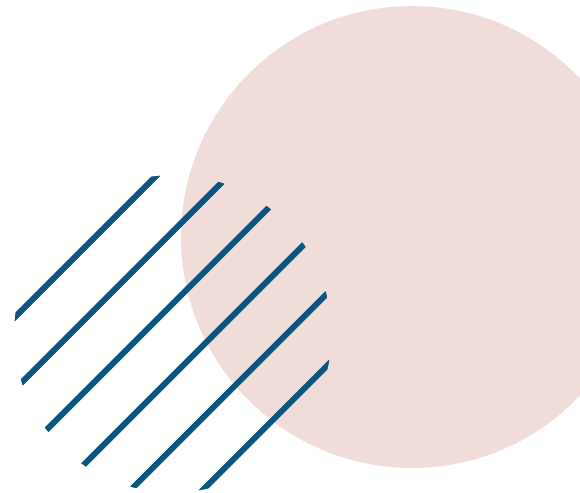


08

Acknowledgements

A large family of passionate contributors made this year's report possible. Survey question design, analysis, writing, editing, and report design are just a few of the ways that our colleagues helped to realize this large effort. The authors would like to thank all of these people for their input and guidance on the report this year. All acknowledgements are listed alphabetically.

| | |
|--------------------|------------------------|
| Scott Aucoin | Eric Maxwell |
| Alex Barrett | John Speed Meyers |
| James Brookbank | Steve McGhee |
| Kim Castillo | Jacinda Mein |
| Lolly Chessie | Alison Milligan |
| Jenna Dailey | Pablo Pérez Villanueva |
| Derek DeBellis | Claire Peters |
| Rob Edwards | Connor Poske |
| Dave Farley | Dave Stanke |
| Christopher Grant | Dustin Smith |
| Mahshad Haeri | Seth Vargo |
| Nathen Harvey | Daniella Villalba |
| Damith Karunaratne | Brenna Washington |
| Todd Kulesza | Kaiyuan "Frank" Xu |
| Amanda Lewis | Nicola Yap |
| Ian Lewis | |



09

Authors



Claire Peters

Claire Peters is a User Experience Researcher at Google. Her research comprises various aspects and expressions of DORA in applied settings. She studies Google's Cloud Application Modernization Program (CAMP), DORA-driven customer engagements, and Four-Keys-related tooling, with the goal of helping teams and individuals more effectively apply DORA principles in their day-to-day work. Claire is also a member of DORA's core research team, and constructs the annual DORA survey and the State of DevOps report. She holds an MA in Applied Cultural Analysis from the University of Copenhagen.



Dave Farley

Dave Farley, is the managing director and founder of Continuous Delivery Ltd and Creator of the Continuous Delivery YouTube Channel. Dave is co-author of the best-selling Continuous Delivery book. His also the best selling author of Modern Software Engineering: Doing What Works to Build Better Software Faster. He is co-author of the Reactive Manifesto and a winner of the Duke Award for the open source LMAX Disruptor project. Dave is a pioneer of Continuous Delivery, thought-leader and expert practitioner in CD, DevOps, TDD and software design, and has a long track record in creating high-performance teams, shaping organisations for success, and creating outstanding software. Find more of Dave on his [Twitter](#), [YouTube](#), [blog](#), and [website](#).



Daniella Villalba

Daniella Villalba is a user experience researcher dedicated to the DORA project. She focuses on understanding the factors that make developers happy and productive. Before Google, Daniella studied the benefits of meditation training, the psycho-social factors that affect the experiences of college students, eyewitness memory, and false confessions. She received her PhD in Experimental Psychology from Florida International University.



Dave Stanke

Dave Stanke is a Developer Relations Engineer at Google, where he advises customers on best practices for adopting DevOps and SRE. Throughout his career, he has worn all the hats, including startup CTO, product manager, customer support, software developer, sysadmin, and graphic designer. He holds an MS in Technology Management from Columbia University.



Derek DeBellis

Derek DeBellis is a Quantitative User Experience Researcher at Google. At Google, Derek focuses on survey research, logs analysis, and figuring out ways to measure concepts central to product development. Derek has recently published on Human-AI interaction, the impact of Covid-19's onset on smoking cessation, designing for NLP errors and the role of UX in privacy discussions.



Eric Maxwell

Eric Maxwell leads Google’s DevOps Digital Transformation practice where he advises the world’s best companies on how to be even better – a little bit at a time, continuously. 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), is a member of the DORA Core Team, and author of the DevOps Enterprise Guidebook. Before Google, Eric spent time whipping up awesome with other punny folks at Chef Software.



John Speed Meyers

John Speed Meyers is a security data scientist at Chainguard, a software supply chain security startup. John’s research projects have included topics such as software supply chain security, open source software security, and the world’s response to growing Chinese military power. Previously, John worked at In-Q-Tel, the RAND Corporation, and the Center for Strategic and Budgetary Assessments. John has a PhD in policy analysis from the Pardee RAND Graduate School, a masters in public affairs (MPA) from Princeton’s School of Public and International Affairs, and a BA in international relations from Tufts University.



Kaiyuan “Frank” Xu

Kaiyuan “Frank” Xu is a Quantitative User Experience Researcher at Google. He analyzes log and survey data to understand usage patterns and user feedback for Google Cloud products, improving product excellence for developers. Before Google, Kaiyuan conducted years of qualitative and quantitative user research at Microsoft for Azure and Power Platform products. He received his MS in Human Centered Design and Engineering from University of Washington.



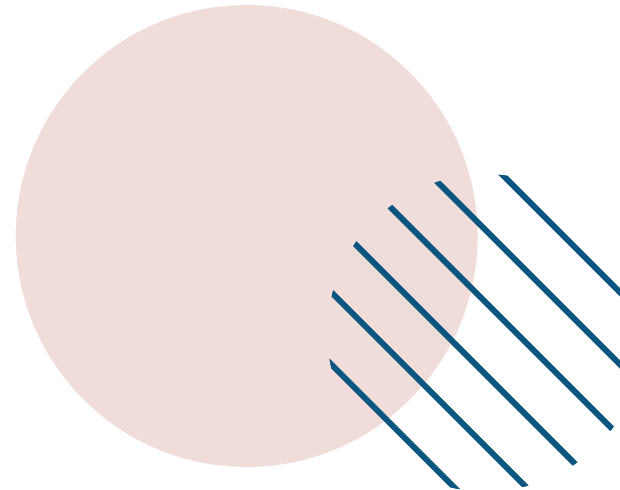
Nathen Harvey

Nathen Harvey is a Developer Relations Engineer at Google who has built a career on helping teams realize their potential while aligning technology to business outcomes. Nathen has had the privilege of working with some of the best teams and open source communities, helping them apply the principles and practices of DevOps and SRE. Nathen co-edited and contributed to *97 Things Every Cloud Engineer Should Know*, O'Reilly 2020.



Todd Kulesza

Todd Kulesza is a User Experience Researcher at Google, where he studies how software engineers work today, and how they might be able to work better tomorrow. He holds a PhD in Computer Science from Oregon State University.



10

Methodology

Research design

This study employs a cross-sectional, theory-based design known as **inferential predictive**, one of the most common types of designs conducted in business and technology research today. Inferential predictive design is used when purely experimental design is not practical or impossible.

Target population and sampling

The target population for this survey was practitioners and leaders working in, or closely with, technology and transformations, especially those familiar with DevOps. We promoted the survey via email lists, online promotions, an online panel, social media, and by asking people to share the survey with their networks (that is, snowball sampling).

Creating latent constructs

We formulated our hypotheses and constructs using previously validated constructs wherever possible.

We developed new constructs based on theory, definitions, and expert input. We then took additional steps to clarify intent to ensure that data collected from the survey had a high likelihood of being reliable and valid.¹

Calculating the differences between low performers and high performers

In the “How do you compare” section we compare low performers and high performers on the four metrics of delivery performance. The method is a simple one. Let’s take deployment frequency as an example. The high cluster deploys on demand (i.e., multiple times per day). If they perform an average of four deployments per day, that ends up being 1460 deployments a year ($4 * 365$). The low performers, by contrast, deploy between once every month, to once every six months, for a mean deployment frequency of once every 3.5 months, and an average deployment frequency of about 3.4 deployments a year ($12/3.5$). This approach is generalized for the other development performance metrics.

¹ Churchill Jr, G. A. “A paradigm for developing better measures of marketing constructs,” *Journal of Marketing Research* 16:1, (1979), 64–73.

Statistical analysis methods

Cluster analysis

For both of the clustering solutions portrayed in the “How do you compare” section, we used hierarchical clustering with a version of Ward’s agglomerative method² to evaluate how well varying cluster solutions fit the data.

For the first clustering results we presented, we looked for clusters of responses across deployment frequency, lead time, time to restore a service, and change failure rate. This year we found three clusters after evaluating 14 different hierarchical clustering solutions using 30 different indices for determining the number of clusters.³

The second cluster analysis we presented was methodologically identical to the first, but was deployed over different dimensions in the data. We wanted to

look for common response patterns (i.e., clusters) across throughput (a composite of deployment frequency and lead time), operational performance (reliability) and stability (a composite of time to restore service and change failure rate). We also explored different clustering algorithms to see how sensitive our results were to our approach. Though there isn’t an established way to quantify that sensitivity (that we know of), the clusters that emerged tended to have similar characteristics.

Measurement model

Before conducting our analysis, we identified constructs using exploratory factor analysis with principal component analysis using varimax rotation.⁴ We confirmed statistical tests for convergent and divergent validity and reliability using average variance extracted (AVE), correlation, Cronbach’s alpha⁵, ρ_{AA} , heterotrait-monotrait ratio^{5 7}, and composite reliability.

² Murtagh, Fionn, and Pierre Legendre. “Ward’s hierarchical agglomerative clustering method: which algorithms implement Ward’s criterion?.” *Journal of classification* 31.3 (2014): 274-295.

³ Charrad M., Ghazzali N., Boiteau V., Niknafs A. (2014). “NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set.”, *Journal of Statistical Software*, 61(6), 1-36.”, “URL <http://www.jstatsoft.org/v61/i06/>”

⁴ Straub, D., Boudreau, M. C., & Gefen, D. (2004). Validation guidelines for IS positivist research. *Communications of the Association for Information systems*, 13(1), 24.

⁵ Nunnally, J.C. *Psychometric Theory*. New York: McGraw-Hill, 1978

⁶ Hair Jr, Joseph F., et al. “Partial least squares structural equation modeling (PLS-SEM) using R: A workbook.” (2021): 197.

⁷ Brown, Timothy A., and Michael T. Moore. “Confirmatory factor analysis.” *Handbook of structural equation modeling* 361 (2012): 379.

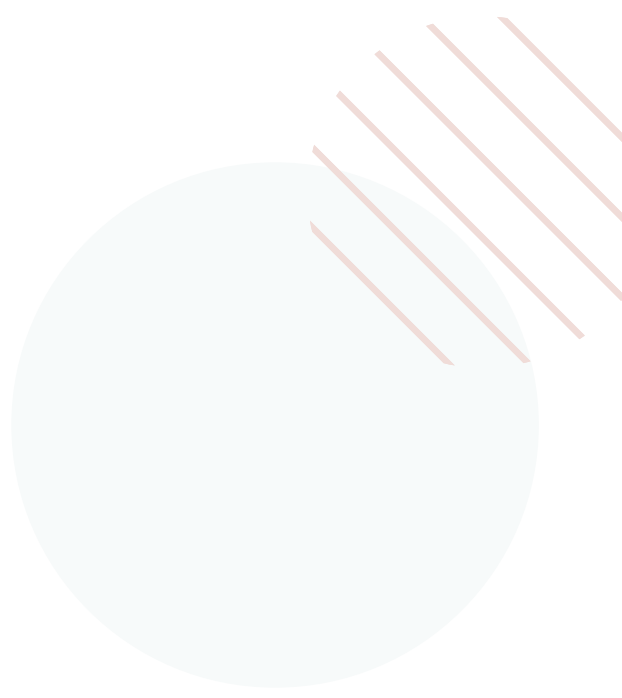


Structural equation modeling

We tested the structural equation models (SEM) using Partial Least Squares (PLS) analysis⁸, which is a correlation-based structural equation modeling.

Analysis of the second cluster model

To understand what predicts cluster membership, we employed multinomial logistic regression.⁹ We used this approach because we were trying to predict cluster membership, which, in this case, is unordered categorical data with more than two levels. To understand the outcomes that cluster memberships predicted, we used a linear regression for each outcome (burnout, unplanned work, and organizational performance).



⁸ Hair Jr, J. F., Hult, G. T. M., Ringle, C. M., & Sarstedt, M. (2021). "A primer on partial least squares structural equation modeling (PLS-SEM)." Sage publications.

⁹ Ripley, Brian, William Venables, and Maintainer Brian Ripley. "Package 'nnet'." R package version 7.3-12 (2016): 700.

11

Further reading

Join the DORA Community to discuss, learn, and collaborate on improving software delivery and operations performance.

<http://dora.community>

Learn more about the Four Keys metrics

<https://goo.gle/four-keys>

Find more information on DevOps capabilities.

<https://goo.gle/devops-capabilities>

Learn more about how you can implement DORA practices in your organization, from our Enterprise Guidebook

<https://goo.gle/enterprise-guidebook>

Find resources on Site Reliability Engineering (SRE)

<https://sre.google>

<https://goo.gle/enterprise-roadmap-sre>

Take the DevOps Quick Check

<https://goo.gle/devops-quickcheck>

Explore the DevOps research program

<https://goo.gle/devops-research>

Learn from other companies who have implemented DORA practices by reading out Google Cloud DevOps Award winner ebook

<https://goo.gle/devops-awards>

Find out about the Google Cloud Application Modernization Program or CAMP.

<https://goo.gle/3daLa9s>

Leveraging data and DORA metrics to transform tech processes


<https://goo.gle/3Doh8Km>

Read the whitepaper: “The ROI of DevOps Transformation: How to quantify the impact of your modernization initiatives” by Forsgren, N., Humble, J., & Kim, G. (2018).

<https://goo.gle/3qECllh>

Read the book: *Accelerate: The science behind devops: Building and scaling high performing technology organizations*. IT Revolution.

<https://itrevolution.com/book/accelerate>



Learn more about the Supply chain Levels for Secure Artifacts (SLSA) framework

<https://slsa.dev>

Learn more about Secure Software Development Framework (NIST SSDF)

<https://goo.gle/3qBXLWk>

Learn more about DevOps culture: Westrum organizational culture

<https://goo.gle/3xq7KBV>

Learn more about Open Source Security Foundation

<https://openssf.org/>

Learn more about in-toto

<https://in-toto.io/>

Learn more about NTIA.gov's Software Bill of Materials

<https://www.ntia.gov/SBOM>

Cybersecurity: Federal Response to SolarWinds and Microsoft Exchange Incidents

<https://www.gao.gov/products/gao-22-104746>

Learn more about application-level security scanning as part of CI/CD

<https://go.dev/blog/survey2022-q2-results#security>

Last but not least, see prior State of DevOps Reports.

All are listed at <https://www.devops-research.com/research.html#reports>:

[2014 Accelerate State of DevOps Report](#)

[2015 Accelerate State of DevOps Report](#)

[2016 Accelerate State of DevOps Report](#)

[2017 Accelerate State of DevOps Report](#)

[2018 Accelerate State of DevOps Report](#)

[2019 Accelerate State of DevOps Report](#)

[2021 Accelerate State of DevOps Report](#)



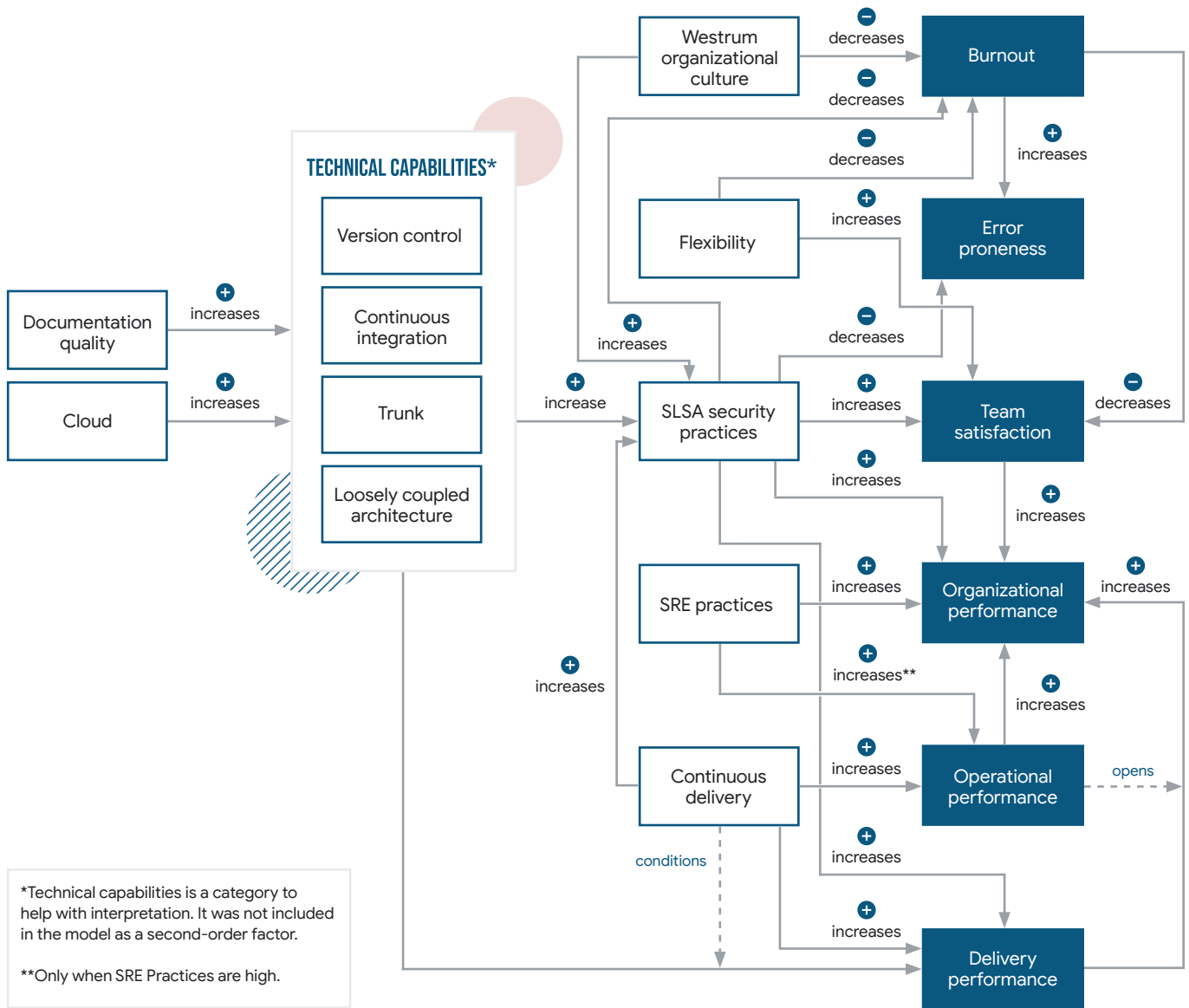
For FAQs, please see

<http://devops-research/faq.html>

12

Appendix

2022 SEM



Legend

Direct effects

We indicate whether predictor seems to increase or decrease the outcome. If there is no text in the effect, it probably because there is a moderator.

Types of variables

Often researchers differentiate between exogenous and endogenous variables. Here, however, we highlight in blue variables we often treat as ends-in-themselves or key outcomes. Gray is important and some may consider ends in themselves, but for the purposes of our work, we consider these as possible means to an end.

Conditionality

We can think of conditionality as an effect on an effect. For example, the effect of delivery performance on organizational performance is affected (read conditioned or moderated) by operational performance, such that delivery performance only shows a positive impact on organizational performance when operational performance is high.

There are a few ways we might talk about how the moderator impacts the effect between two variables:

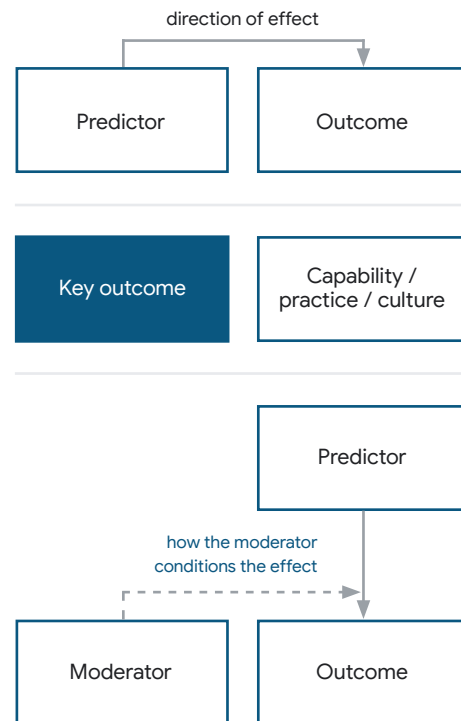
Amplifies = makes the effect stronger

Attenuates = makes the effect weaker

Opens = enables the effect to emerge

Gates = prevents the effect from emerging

Conditions = it's complicated, but it changes the effect



["2022 Accelerate State of DevOps Report"](#) by Google LLC is licensed under [CC BY-NC-SA 4.0](#)