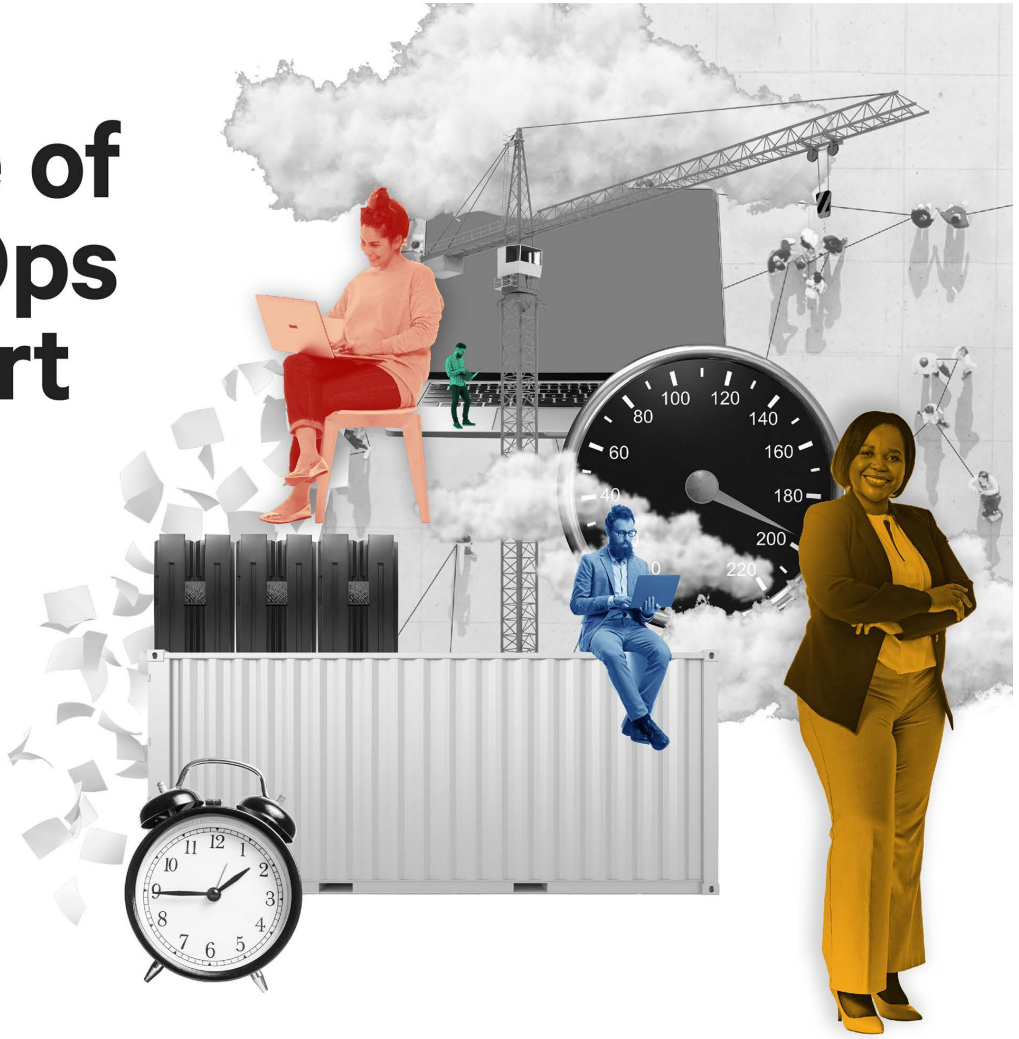
2020 State of DevOps Report



Presented by

puppet

circleci

Sponsored by

servicenow sysdig

armory New Relic. splunk>

Contents

Executive summary

Scaling DevOps practices with internal platforms

Internal platform usage is widespread.

- Sixty-three percent of respondents say their company has at least one self-service internal platform.
- Of those with internal platforms, 60 percent have between two and four.
- Almost a third of respondents have 25 to 50 percent of developers using an internal platform

High DevOps evolution correlates strongly with high use of internal platforms. Highly evolved firms are six times as likely to report high use of internal platforms as firms at a low level of DevOps evolution.

A product mindset is key to scaling DevOps and your platform. Highly evolved firms are nearly twice as likely to be highly product-oriented as firms in the middle of their DevOps evolution.

Higher levels of DevOps evolution mean more self-service offerings for developers. Highly evolved firms offer a wide range of self-service capabilities, including:

- CI/CD workflows
- Internal infrastructure
- Public cloud infrastructure
- Development environments
- Monitoring and alerting
- Deployment patterns
- Database provisioning
- Audit logging

Top challenges to providing an internal platform

Lack of time	Lack of standardization	Lack of technical skills within the team

Executive summary
Change management in the DevOps era

Change management effectiveness increases as organizations evolve their DevOps practices. Highly evolved firms are nearly three times as likely to have highly effective change management as firms at a low level of DevOps evolution.

The most effective change management is achieved by firms that emphasize:

- A high degree of testing and deployment automation
- A high degree of automated risk mitigation
- Less rigid and much less manual approval processes
- Writing changes in code
- Allowing employees more scope to influence change management
- DevOps processes and culture

Highly orthodox approval processes make change management process inefficient. Firms with highly orthodox approvals are nine times more likely to have high levels of inefficiency in their change management process than firms with low levels of orthodox approval.

Automation makes people more confident their change management is effective. Firms whose employees believe their change management is effective are three times more likely to automate testing and deployment than firms where confidence in change management performance is low.

Firms that give people a say in the change management process have better change management.

- Firms that have high employee involvement in the change management process are more than five times as likely to have highly effective change management than firms with low employee involvement.
- Firms that focus on automation the most also involve their employees the most in their change management process.
- Firms that have the heaviest and most manual process involve their employees the least in their change management process.

Top challenges to automating the change management process

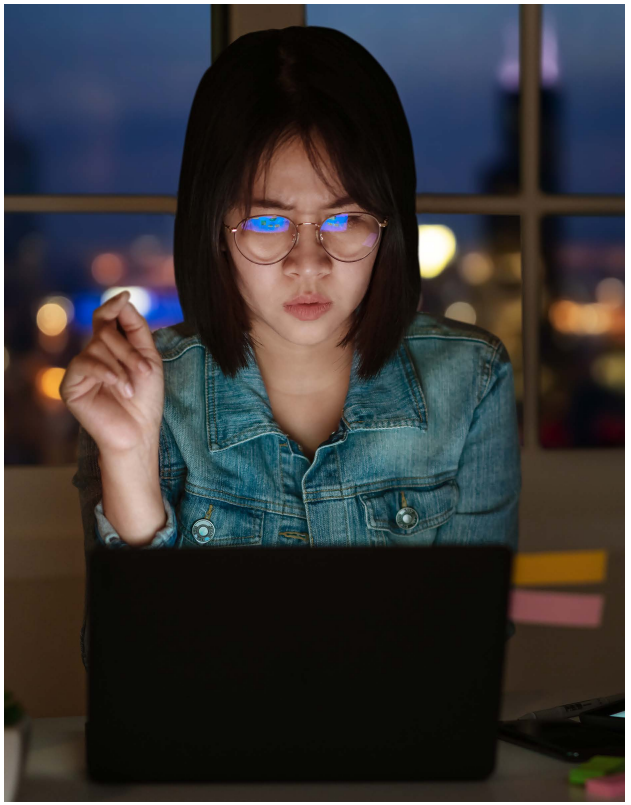
Incomplete test coverage	Organizational mindset	Tightly coupled application architecture

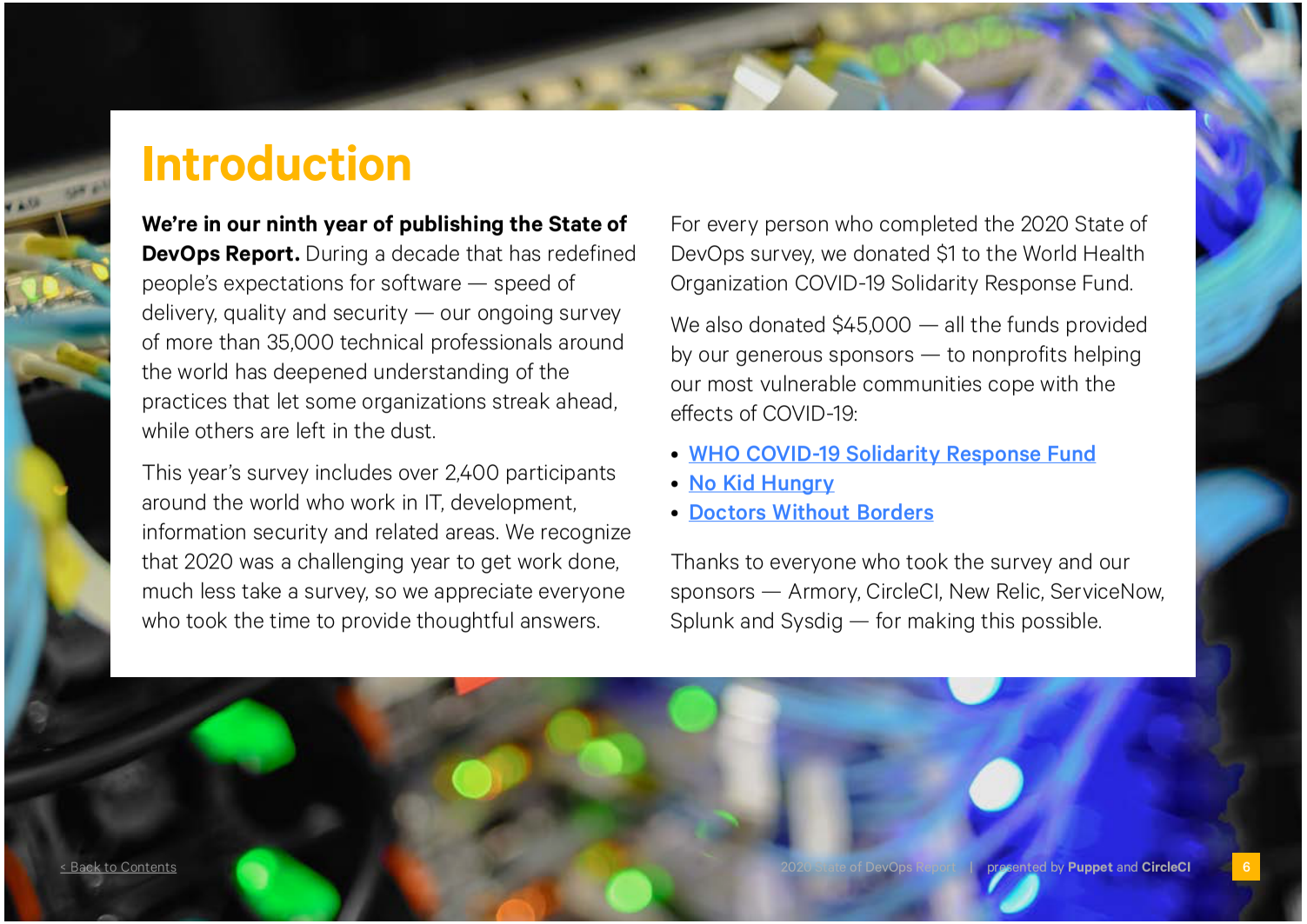
Executive summary
Security integration

Integrating security fully into the software delivery process improves your ability to quickly remediate critical vulnerabilities.

- Among companies with full security integration, 45 percent can remediate critical vulnerabilities within a day.
- Just 25 percent of those with low security integration can remediate within a day.

Integrating security fully into the software delivery process leads to providing self-service for security and compliance validation. Companies that have fully integrated security are more than twice as likely to offer self-service for security and compliance validation as firms with no security integration.





Introduction

We're in our ninth year of publishing the State of DevOps Report. During a decade that has redefined people's expectations for software — speed of delivery, quality and security — our ongoing survey of more than 35,000 technical professionals around the world has deepened understanding of the practices that let some organizations streak ahead, while others are left in the dust.

This year's survey includes over 2,400 participants around the world who work in IT, development, information security and related areas. We recognize that 2020 was a challenging year to get work done, much less take a survey, so we appreciate everyone who took the time to provide thoughtful answers.

For every person who completed the 2020 State of DevOps survey, we donated $1 to the World Health Organization COVID-19 Solidarity Response Fund.

We also donated $45,000 — all the funds provided by our generous sponsors — to nonprofits helping our most vulnerable communities cope with the effects of COVID-19:

- WHO COVID-19 Solidarity Response Fund
- No Kid Hungry
- Doctors Without Borders

Thanks to everyone who took the survey and our sponsors — Armory, CircleCI, New Relic, ServiceNow, Splunk and Sysdig — for making this possible.

Over the years, we've shown that DevOps practices lead to better performance and organizational outcomes. We have learned and shared the practices and patterns that enable organizations to evolve, and to release better software faster.

Despite the notable progress we've witnessed, we have also seen that most organizations struggle to move beyond the middle stages of their DevOps evolution. They are rarely able to scale DevOps ways of working beyond the development, operations, and (sometimes) security teams.

Yet some organizations do succeed. They expand DevOps practices beyond the initial early-adopting teams, continuing to evolve and improve across the organization. What makes the difference? The successful organizations enact deeper structural changes.

This year's DevOps survey has shown two areas of structural change that can yield excellent results: **a platform approach to software delivery** and **applying DevOps principles to change management**. When organizations successfully establish a platform model for enabling application development, or significantly improve their change management effectiveness, they achieve the goal that DevOps initiatives aim at: faster and easier delivery of better quality, more secure software.

Why did we investigate these two areas?

- **The platform model** is a fairly new approach to enabling application teams. Done right, it simply works, resulting in faster, more efficient delivery of high-quality software that meets an organization's business needs — and at scale.
- **Change management** is a common bottleneck that prevents software from being released at a pace that allows the business to achieve its goals. Efficient, effective change management improves an organization's ability to release software on schedule, at the quality and security level the business requires.

In Chapter 1, we share our survey findings about the platform approach, and describe how DevOps principles inform it. In Chapter 2, we discuss the various approaches to change management that we discovered among our survey respondents, and show how applying DevOps principles can turn change management from a blocker into an enabler of faster, safer software delivery.

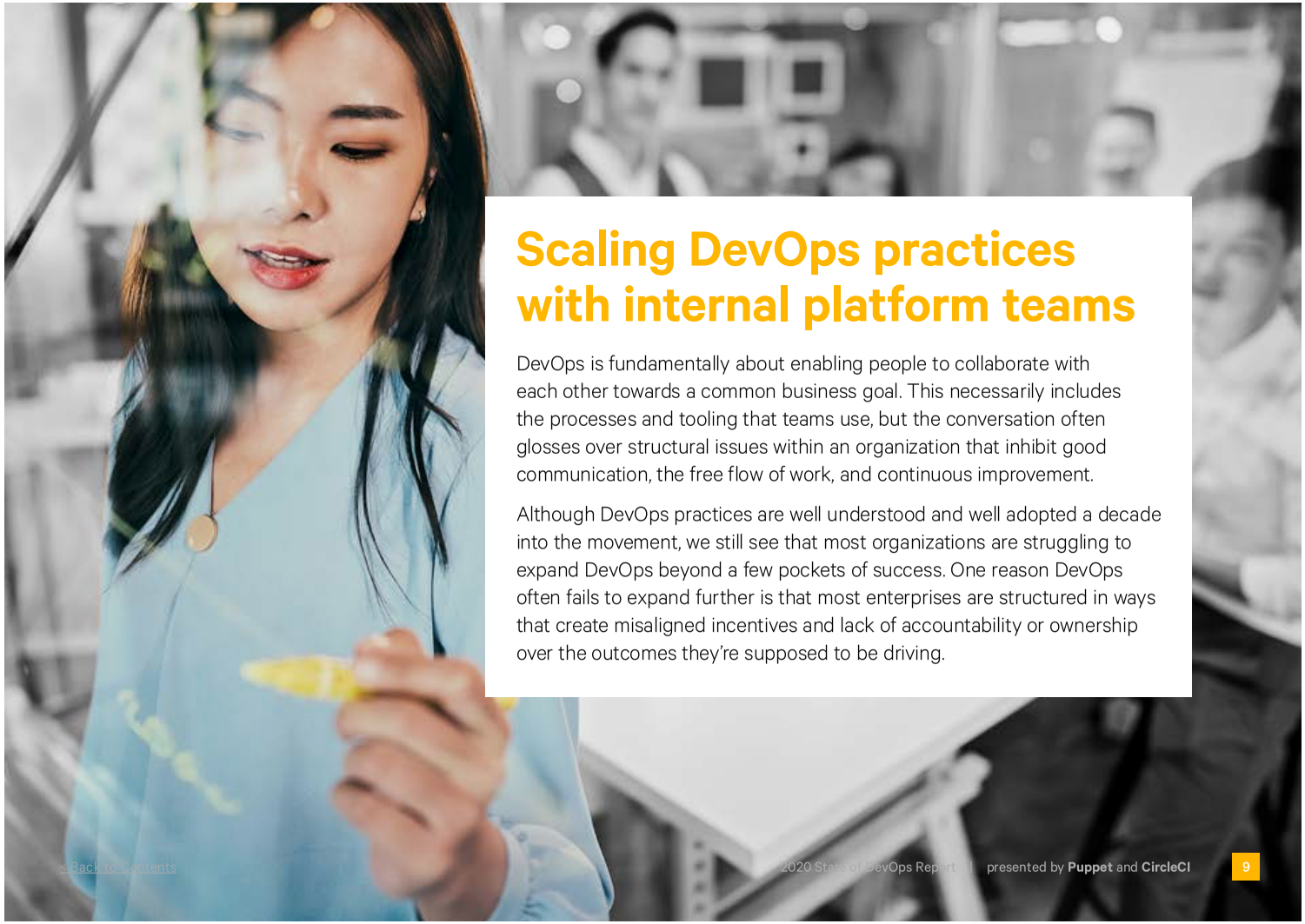
Expanding DevOps beyond Dev and Ops

In any organization, creating value through software does not depend solely on good collaboration between developers and operators. Nearly all adjacent business functions are ultimately part of the software process, and these need to evolve along with the technical delivery teams.

Agile, once the exclusive property of engineers, has evolved over the years, spreading beyond software teams to finance, human resources, executive leadership teams and more. We hope that DevOps principles and practices will likewise continue to spread beyond the dev and ops teams that first began working with them. This has already happened to some degree with DevSecOps, FinOps, and probably other new manifestations we haven't seen yet.

For DevOps principles to spread further, though, people who care about the movement need to extend their empathy and passion beyond the teams that are closest to them, and learn to collaborate with teams whose functions are further away.

Perhaps a few years from now, the term "DevOps" will sound quaint — even fade away — because so many people and organizations have fully adopted the DevOps principles of collaboration, communication, small-batch iteration, feedback loops, continuous learning and improvement. We certainly hope so.



Scaling DevOps practices with internal platform teams

DevOps is fundamentally about enabling people to collaborate with each other towards a common business goal. This necessarily includes the processes and tooling that teams use, but the conversation often glosses over structural issues within an organization that inhibit good communication, the free flow of work, and continuous improvement.

Although DevOps practices are well understood and well adopted a decade into the movement, we still see that most organizations are struggling to expand DevOps beyond a few pockets of success. One reason DevOps often fails to expand further is that most enterprises are structured in ways that create misaligned incentives and lack of accountability or ownership over the outcomes they're supposed to be driving.

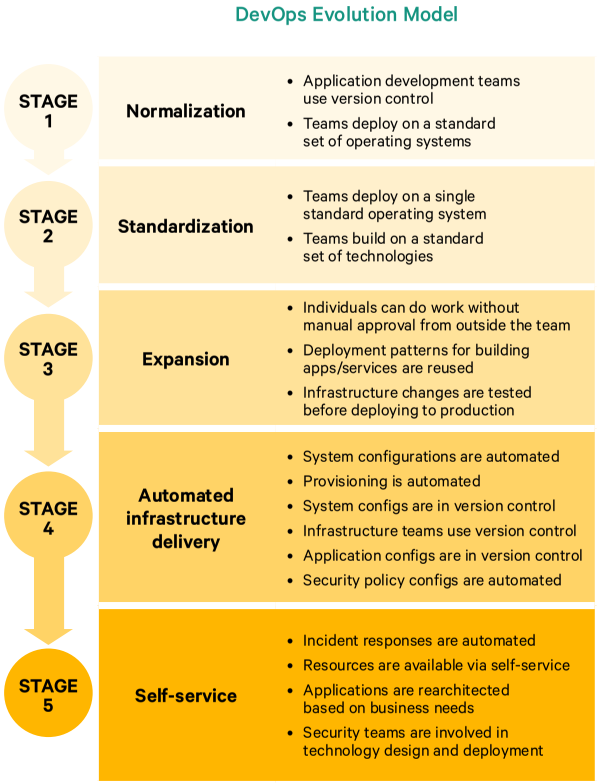
Teams adopting a set of practices alone cannot further DevOps evolution; you have to make the corresponding structural changes to optimize the way teams work.

The DevOps evolution model (see chart at right) shows that organizations do not progress to self-service and security integration until Stages 4 and 5, after individual people are given more autonomy to work without manual approval from outside the team (Stage 3).

Stage 3 is a critical point of convergence — trust has been built up in Stages 1 and 2; teams are granted more autonomy; and deployment is no longer a four-alarm fire. At this point, teams can expand their new ways of collaborating across more functional boundaries, beyond Dev and Ops.

In Stages 3 to 5 we see a loosening of one-size-fits-all rules and processes, with an underlying focus on automation. At these stages, automation has expanded beyond solving local problems for a single individual or team to an explicit — and higher — focus on creating value for the business.

This is what it means to scale DevOps practices: By empowering individuals and teams to rely on their knowledge and experience — and by automating — you are able to optimize at an organization-wide scale. You are now able to focus on eliminating waste across multiple delivery streams, and help the business achieve its goals.



Platform model: A new-ish approach to scaling DevOps

The platform model is an approach we're seeing more and more often in the field. It grew out of the idea of product teams (popularized by the DevOps movement), which are responsible for end-to-end delivery of a product or service.

This works very well if you have a single product, or just a few products. But if you have hundreds of products or services, dedicating a product team to each one is both inefficient and expensive. Imagine 10 teams, each with its own technology stack, toolchain and processes. You're going to have all these teams trying to solve similar problems, spending way too much time on evaluating technologies, integrating them, maintaining the infrastructure and more. That's time that could be better spent building and improving the actual products your teams are responsible for.

The lack of standardized technologies and processes creates other problems, too:

- Governance becomes expensive, and nearly impossible to manage.
- Separate stacks reduce knowledge sharing across the organization.
- Many of your product teams don't actually have the skills or expertise to operate a full infrastructure and application stack. Many developers regard infrastructure operations as a distraction from their real work, so they never really focus on it.

While having multiple end-to-end product teams doesn't scale well across large complex environments, a platform model defined by clear purpose, boundaries and responsibilities does. A platform, built with the customer in mind, can significantly reduce the toil and overhead of individual product teams.

Broadly speaking, the platform team provides the infrastructure, environments, deployment pipelines and other internal services that enable internal customers — usually application development teams — to build, deploy and run their applications.

Evan Bottcher's definition of a digital platform is helpful here: "...a foundation of self-service APIs, tools, services, knowledge and support which are arranged as a compelling internal product. Autonomous delivery teams can make use of the platform to deliver product features at a higher pace, with reduced coordination."

Evan points out that self-service is "a key defining characteristic for a good platform.... Specifically it should allow for self-service provisioning, self-service configuration, and self-service management and operation of the platform capabilities and assets."

The four fundamental team topologies

If you're interested in evolving your organizational design and improving team interactions, we highly recommend "Team Topologies" a website run by Manuel Pais and Matthew Skelton, and also their book by the same name. "Team Topologies" describes four fundamental team types: stream-aligned, platform, enabling, and complicated-subsystem. It also defines three team interaction patterns — collaboration, X-as-a-Service, and facilitating — and a team API, which acts as a contract between teams based on code, documentation and user experiences. "Team Topologies" brings together different frameworks, models and case studies to provide a functional and team-centered approach to building complex software systems.

The platform model is often associated with cloud native environments, but is also appropriate for many other types of architecture, ranging from modern to legacy. The primary benefits are:

Application teams can be more efficient. They don't have to be experts in infrastructure operations or have intimate knowledge of every tool in the toolchain, so they are able to focus on the product. Application developers no longer have to wait on a centralized team to provision test environments or cloud resources for them, and their resulting autonomy allows them to work much faster.

Improved governance. You can't effectively manage cost, compliance and audit if all your applications run on entirely different infrastructure stacks, using different processes. An effective platform enables efficient IT governance while empowering application teams to deliver quickly.

An end to context-switching. Constantly switching attention between an application and infrastructure operations is a huge drain on productivity (and creativity too). Both individual workers and teams are better off when they can concentrate on their own particular context. For a deeper dive into these two different contexts and how the teams interact, see the sidebar at right, *Platform and application: two different contexts*.

Continuous infrastructure improvement. A common platform that offers customer-oriented solutions rather than just raw access to infrastructure gives your organization more flexibility. Consumers of the platform are not tied to specific implementations of your infrastructure stack, so the platform team can iteratively replace and upgrade components, and needs to interact only minimally with application teams.

Platform and application: two different contexts

Most software developers and operations engineers understand that switching back and forth between two contexts is a huge cognitive drain. There's a good reason for this, apart from the normal human challenge of context-switching: the details and mindset of each realm are so different, they call on completely different knowledge and experience sets.

The platform team, as the builder and manager of the platform, has specific knowledge of infrastructure operations and the centralized tools they manage. The platform team's context includes monitoring and managing performance; current load on the platform and planning for changes to that load; all changes to storage or the network; issues with the hypervisor; working with application schedulers, database layers and more.

The application team's knowledge and context are completely different. They build, deploy and monitor application components, plus any app infrastructure that they provision themselves and deploy on the platform. An application team's context includes a wide range of considerations, including customer needs, requirements, values and dependencies. The team also has technical considerations such as the app's relationship to other applications; knowledge of the codebase and its current state; and knowledge of current features, as well as those under development or about to be deprecated.

Having a platform team that's distinct from the application team means each group is able to make decisions quickly, based on the context they have. That's a big part of why the platform model enables faster software throughput at a higher level of quality.

Use of internal platforms

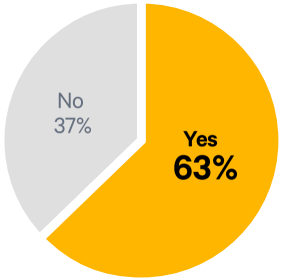
In our discussion of platforms, we use the term "internal platform" to mean one that's been built by and for the organization. We're distinguishing these from platforms that are supplied by outside vendors — for example, many people think of AWS or other IaaS offerings as "platforms." In our survey, we defined platform teams as those that are responsible for maintaining a self-service platform other teams use to build and deliver applications or services.

We asked two questions to measure an organization's use of internal platforms:

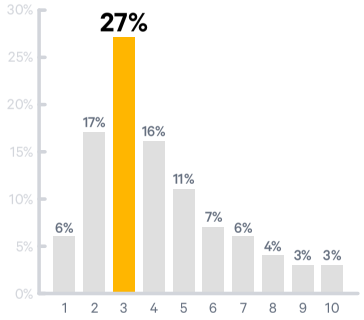
- What percentage of your developers use self-service platform(s)?
- Which services are available for self-service?

We found platform use is pretty widespread amongst our survey respondents. Sixty-three percent said they had at least one self-service internal platform. Of those who had internal platforms, 60 percent had between two and four. Almost a third of those with internal platforms had 26 to 50 percent of their developers using a platform.

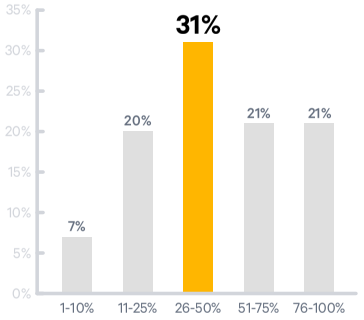
Does your organization use internal platforms?



How many internal platforms does your organization use?



What percentage of your developers use internal platforms?



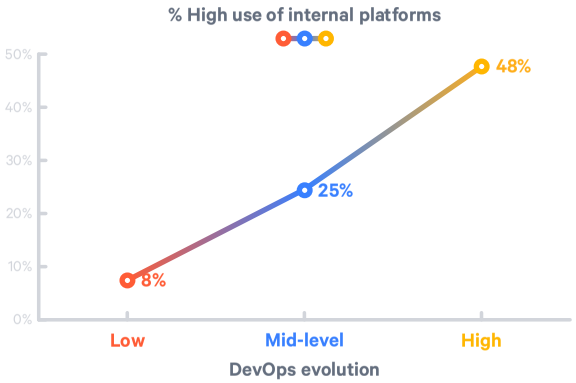
< Back to Contents

Platform use and DevOps evolution

We found a strong relationship between DevOps evolution and the use of internal platforms. Highly evolved firms are almost twice as likely as mid-level organizations to report high usage of internal platforms, and are six times more likely to report high usage than low-level organizations.

This finding mirrors Stage 5 of the DevOps evolution model, where self service is a key practice enabled by a foundation of standardization, automation and team autonomy. The underlying structural changes needed to reach Stage 5 reduce complexity in the technology stack, automate away a lot of toil, and reduce handoffs between teams — all while building a high degree of trust. These are all the necessary components for building an internal platform that can deliver higher value for the organization.

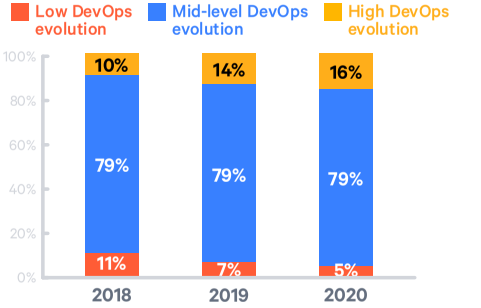
Use of internal platforms and level of DevOps evolution



Still stuck in the middle

In our 2018 State of DevOps Report, we set out to understand how organizations evolve as they progress through their DevOps journey. The analysis produced a five-stage evolution model (see page 10), with each stage composed of key practices that define it. We grouped organizations into high, mid and low levels of DevOps evolution based on how frequently the key practices were employed.

Once again, we found that the vast majority of firms surveyed this year (79 percent) are in the group we characterize as mid-level on the DevOps evolutionary scale — the same as the last two years. Sixteen percent of the overall sample were in the high group, an increase of two percentage points over last year.



Platform as product

The platform team isn't meant to be an ivory-tower-siloed cadre that prescribes all architectures, functionality, tooling and more. The platform team's job is to provide core capabilities that make it easier for their customers — that is, other teams — to get work done and achieve their goals, as well as those of the overall business.

In our experience, it rarely works to require use of the platform without first collaborating with internal customers to understand their needs. Paul Ingles, CTO at RVU, explains how his company has measured the success and effectiveness of platform teams over time.

> *We never mandated the use of the platform, so setting key results for the number of onboarded teams forced us to focus on solving problems that would drive adoption. We also look for natural measures of progress: the proportion of traffic served by the platform, and the proportion of revenue served through platform services are both good examples of that.*
>
> *— From an interview with Paul Ingles at TeamTopologies.com*

You have to make the platform a compelling option. Application teams should *want* to use the platform because it's easier and more cost-efficient than building and maintaining their own.

An internal platform is a product, not a project

The biggest mistake we see is organizations treating (and funding) platform development as a project. Just like any other product team, a platform team needs longevity, consistency and a commitment from management to be fully successful.

There's a common tendency in technical organizations to tap a limited pool of exceptionally skilled people who are well-versed in software engineering practices, infrastructure as code, continuous delivery, APIs and more. They'll be tapped to put together the platform, and then, as soon as there's another important demand, they'll be pulled off the platform team and put on the new urgent thing.

Don't do this. The platform team should not be viewed as fungible. If you want your platform approach to work for the long term, you should get your organization to commit to the platform as a product, one that will need and deserves ongoing development and funding.

Give this product time to be developed, tested, rolled out and iterated on. Over the longer term, you'll build a competency that will become a serious competitive advantage for rolling out future revenue-producing products that can drive your business forward.

So what makes a platform a product? First and foremost, a platform should be built to help deliver global optimization and efficiency at scale. Here are some suggestions for doing that.

Think self-service and API first. The key characteristic of a platform product is self-service capabilities consumed via an API. This includes infrastructure, test environments, deployment pipelines, monitoring, and more. The platform team provides an interface between the underlying infrastructure and tooling and the teams consuming those services, enabling application teams to focus on building their products instead of nitty-gritty implementation or operational details. Self-service enables developers to work at their own pace without having to make requests and wait for fulfillment.

Start locally but build globally. Rather than trying to build the entire platform in one go — based on unverified assumptions about what you think application teams need — start with a localized solution and embrace a lean product management approach. Often an application team will develop a good solution for themselves that can be used by more teams. Working with an existing solution can help drive adoption by enough teams to provide the feedback you need to further develop functionality that will eventually serve multiple teams.

Focus on developer experience and flow. We can't stress enough that empathy is a critical skill set. Empathy means understanding someone's position, and it's impossible to build a good product without having empathy for your user. We've seen platform teams adopt techniques from the UX discipline, such as empathy maps, to understand their customers' needs and pain points. Twilio's platform team surveys their developer team to ensure that devs are satisfied with the services the platform provides, and to continuously improve platform services.

Evangelize. "If you build it, they will come" is a fallacy when it comes to building products. Evangelism is critical to the success of any product. You have to demonstrate the capabilities of your platform in a way your customers can relate to. You also have to keep developers informed of changes and updates, publicize upcoming enhancements, and publicly report metrics on usage and successful outcomes of the product.

Continuously invest in the product. A platform is not a one-and-done project. Once you've assembled a platform team, commit to keeping it in place so they can continue to develop and improve the platform, meeting new organizational needs as they arise.

Turning a local solution into a global one

— From "Product for Internal Platforms" by Camille Fournier on Medium

Don't be ashamed to take over a system from a team that built it with themselves in mind, if that system seems to be the right general concept for the wider company... I did this when I built a global service discovery solution long ago. Another team had first identified the problem and created their own version of a solution using ZooKeeper. The solution was fine for their needs, but didn't solve the general needs of everyone at the company for global scaling. So I took over the idea of the project, and turned it into true platform infrastructure, built for a big company and not just one team therein. There were plenty of product decisions to make as part of that work, but the core identification of the problem as worth solving was done for me. There is a lot of interesting work in taking a solution that is locally optimized and turning it into something that can be used by a diverse set of applications.

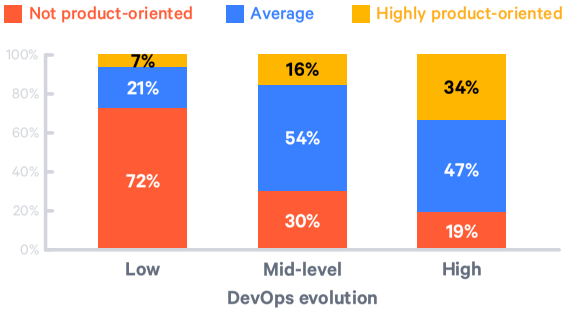
Product mindset

We wanted to test one of our core hypotheses: The more you treat your platform as a product, the more likely your platform is to succeed. In order to understand whether platform teams exhibit a product mindset, we asked survey participants whether:

- The platform team gathers requirements from internal stakeholders
- Someone on the platform team acts as a product manager for the platform(s)
- There is a roadmap for the platform
- The platform team provides onboarding help
- The platform team tests new capabilities with the teams that will use them

The more a respondent agreed with these statements about their platform team(s), the higher their score for a product mindset.

DevOps evolution and platform team behavior



We found that organizations whose DevOps evolution has reached an advanced level are nearly twice as likely to be highly product-oriented as companies that are in the middle of their DevOps evolution.

Treating your platform as a product means that from the first, you do the things any good product team does: You gather user stories and requirements; create a product road map; establish metrics for adoption and then publicize them; survey customers for additional learning; and engage in continuous improvement.

Treating your platform as a product also comes with all the rigor and advantages of the software development discipline. Beyond requirements gathering, validation, building and shipping internal features, you should also publish documentation for capabilities and APIs; offer training and onboarding materials to your customers; and make a changelog visible to all teams that use the platform.

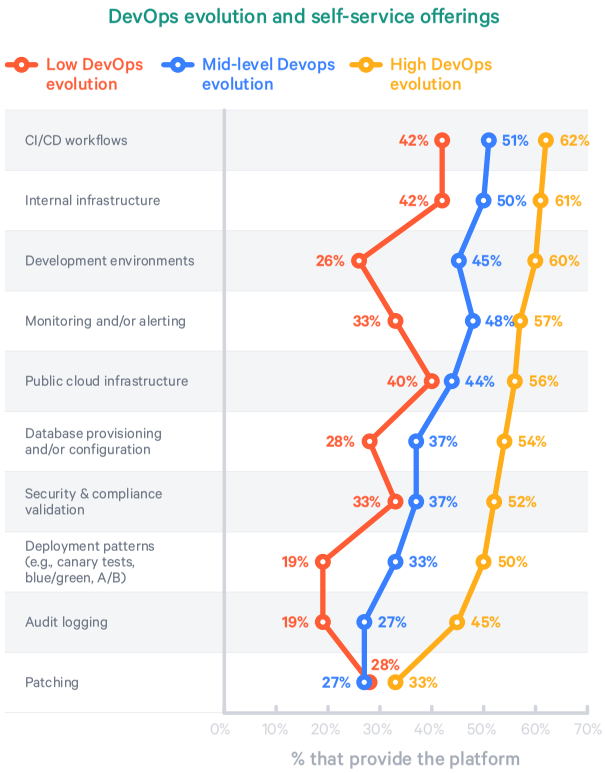
Last but absolutely not least, empathy for your customer must underlie everything else we've mentioned. All the product-development rigor in the world won't help if you don't take the time to truly listen to your customers when they tell you what pains them the most, and what they really desire that will make their jobs easier and their work go more smoothly. When empathy fills your discovery process, you'll have a much better chance of building the right capabilities for your customers from the start. You'll also get, as an organization, a much higher return on your investment in platform development.

DevOps evolution and platform evolution go together

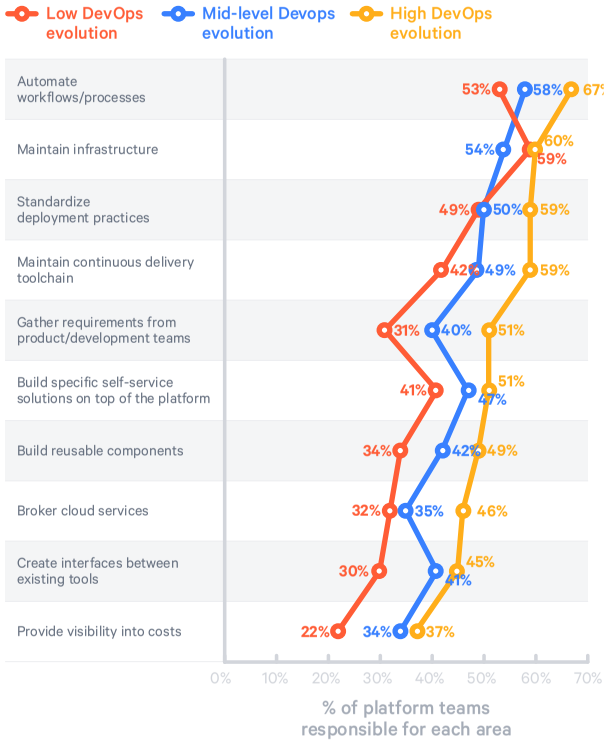
How does a team go from building a few self-service interfaces to providing a comprehensive internal platform that satisfies a wide range of organizational needs?

The graph to the right shows how platform offerings change as organizations progress through their DevOps evolution. The y-axis shows different types of self-service offerings. In each row representing a self-service offering, we've placed three colored dots, with each color representing a different level of DevOps evolution. The x-axis represents the percentage of a group that has adopted a given self-service offering. You'll notice the gaps in adoption between groups at low, mid and high levels of DevOps evolution.

- At a low level of DevOps evolution, organizations offer self-service for CI/CD workflows, internal infrastructure and public cloud infrastructure.
- Mid-evolution organizations expand their internal platforms, providing development environments, monitoring and alerting.
- High-evolution organizations tend to offer a wide variety of internal platforms. This is where you can see more self-service for deployment patterns, database provisioning and audit logging.



DevOps evolution and platform team responsibilities



In addition to self-service offerings, we looked at core responsibilities for platform teams.

- At low levels of DevOps evolution, platform teams are commonly responsible for workflow automation, standardizing deployment practices and maintaining infrastructure.
- As organizations evolve further in their DevOps practices, platform teams expand their responsibilities further, too. They move on to gathering requirements form product teams and maintaining continuous delivery toolchains.

Interfaces used for self-service as organizations evolve their DevOps practices

We asked respondents to tell us which interfaces are used in their organization for self-service. Developer-friendly interfaces expand significantly when we compare organizations at a low level of DevOps evolution to those at a high level, while use of off-the-shelf portals remains flat.

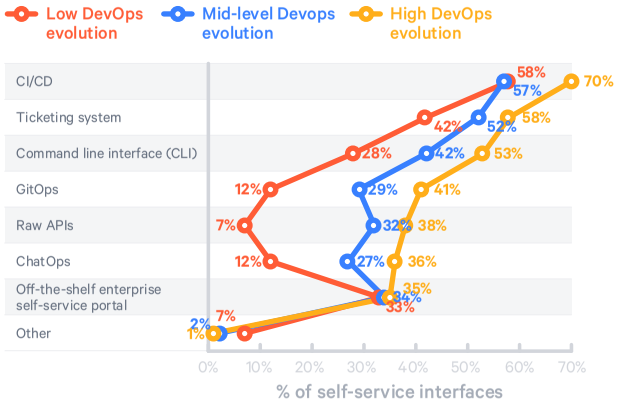
CI/CD tools are the dominant interface used at all levels of DevOps evolution. CI/CD is now common, and therefore accessible and friction-free for most engineers. Integration of CI/CD with version control, email, chat and ticketing systems allows engineers to get immediate feedback while staying in the flow of their work.

Ticketing system usage increases 16 percentage points from the low-evolution companies to those at a high level of DevOps practice. It is the second-most-used interface at all levels of evolution. We surmise the increase is due to vendors offering more integrations with DevOps tools to automate creation and approval of change tickets, based on sophisticated policies.

When we compare low-evolution companies to those at a high level of DevOps evolution, we see expanded use of raw APIs, GitOps, ChatOps and command line interfaces (CLIs). That's because these interfaces allow their users to build on them, creating the workflows they need, without having to consult anyone in authority. This means the producers of the interface and the consumers of the interface can easily collaborate; also, different consumers of the interface can collaborate with each other. For example, if you have self-service interfaces that can provision virtual machines, make firewall changes, and attach storage, it's simple for users to compose those into higher level workflows to automatically test pending code changes.

Use of off-the-shelf, enterprise-grade self-service portals neither increases nor decreases when we compare low-evolution to high-evolution companies. Our takeaway: The interfaces that get used the most broadly are those that take advantage of existing skills, and don't require learning new tools or new working methods. Platform teams should therefore choose to standardize on tools that integrate well with the tools their customers already use, to avoid introducing friction into development workflows. The best tools to introduce are those with rich APIs that enable composability.

DevOps evolution and interfaces for self-service



Providing an internal platform: the challenges

Globally, the top three challenges are lack of time, lack of standardization, and lack of technical skills within the team.

The differences between responses for companies at different levels of DevOps evolution are predictable. At a low level of evolution, respondents selected lack of time as the top challenge, followed by lack of empowerment from leadership, lack of technical skills and lack of standardization. Lack of awareness of what the platform would deliver and lack of automation to build the platform tied for fourth place.

Many of these challenges are mutually reinforcing. Lack of empowerment from leadership is often due to a team's inability to express the benefits of a platform in terms that leadership cares about. Lack of time is often a symptom of too much manual work and not enough standardization to create economies of scale.

At the mid-level of DevOps evolution, lack of time is again the top challenge, followed by lack of standardization. Lack of clearly defined processes and lack of technical skills are tied for third. To make sense of this, note that as the platform team scales, it's important to continually balance standardization and autonomy. As Galo Navarro explains, "To make meaningful impact, platform teams depend on having standards in their organization. Trying to support every possible language ecosystem, framework, DB, messaging system, and whatnot spreads platform teams too thin to be effective." Clearly defined processes are also important; these act as a contract between the platform team and its internal customers.

Like less-evolved companies, the highly evolved organizations cite lack of time as their top challenge, though it's a significantly lower percentage reporting this than in the least-evolved organizations. We think this means lack of time to do all the things their customers are asking them to do. These teams have proven their value at this point, and their services are in high demand. Tying for second place among highly evolved teams: lack of standardization and onboarding development teams to the platform.

Challenges to providing an internal platform

Low evolution	Mid-level evolution	High evolution
• Lack of time	• Lack of time	• Lack of time
• Lack of empowerment from leadership	• Lack of standardization	• Lack of standardization (tie with below)
• Lack of standardization (tie with below)	• Lack of clearly defined practices (tie with below)	• Onboarding dev teams to the platform (tie)
• Lack of technical skills (tie)	• Lack of technical skills (tie)	

How do I know if my platform is successful? A case study in platform engineering

Mike Stahnke,
Vice President of Platform
CircleCI

A successful platform is normally measured by adoption and usage — important metrics to track to ensure you're serving your internal customers. From a business perspective, though, you also need to show that your platform delivers a worthy return on the investment your organization makes to build, run, maintain and evolve it.

At CircleCI, our platform engineering organization is measured on four things: **availability, cost, security,** and **developer productivity.**

Our platform engineering organization was created with **availability** as a key objective. We standardized our tools and capabilities so we could thoroughly understand how they work, what the failure modes are, the lifecycle expectations of a component, and to make sure components can handle millions of requests per hour.

On the other side of the coin is **cost**. You can improve availability by overprovisioning capacity or buying additional tooling, but that may cause you to exceed your cost targets and depress your margins. You can lower costs by turning off services or capabilities, but that can hurt your resilience and availability. That's why these two items are inextricably coupled.

We call out **security** as a measure on its own, both to highlight its importance and to make sure people beyond the dedicated security team keep security in mind when adding and modifying capabilities. We also measure conformance to internal and external security policies.

The final measure is **developer productivity**. A platform team that builds shared components, libraries and tooling allows developers to move demonstrably faster than they would without such a team. At some scale, you can get to the point where hiring an engineer to work on developer tooling has the equivalent impact on output of hiring a product team engineer on multiple teams. For more on thinking about developer productivity, this is my favorite resource: www.gigamonkeys.com/flowers.

The ROI of your platform

Your firm (or your team) probably has some idea of what downtime costs. If you don't, that is the right place to start. Once you know the approximate cost of downtime, you can begin calculating the value of shrinking your downtime and increasing your uptime. Apart from actual transaction value, make sure you do your best to quantify improved user trust and confidence.

Next you should figure out the costs associated with running disparate systems. Can you achieve the same uptime by consolidating these systems into a platform? The consolidation effort could be quite expensive, so you'll need to factor the transition time and effort into your ROI model.

Another element in your ROI calculations is developer or application team productivity. What is the value of those engineers' time? What percentage of their time is spent on tasks that a platform could handle? You may be able to show a good return on investment from saved developer time alone, but you should also take into account what those developers could be doing once their time is freed up. They could be getting features to market faster, integrating a new capability, and reducing technical debt elsewhere in the technology stack.

Once you've made a decision to invest in a platform, other metrics may become important to display and share. Here are some metrics we make available for our internal teams at CircleCI:

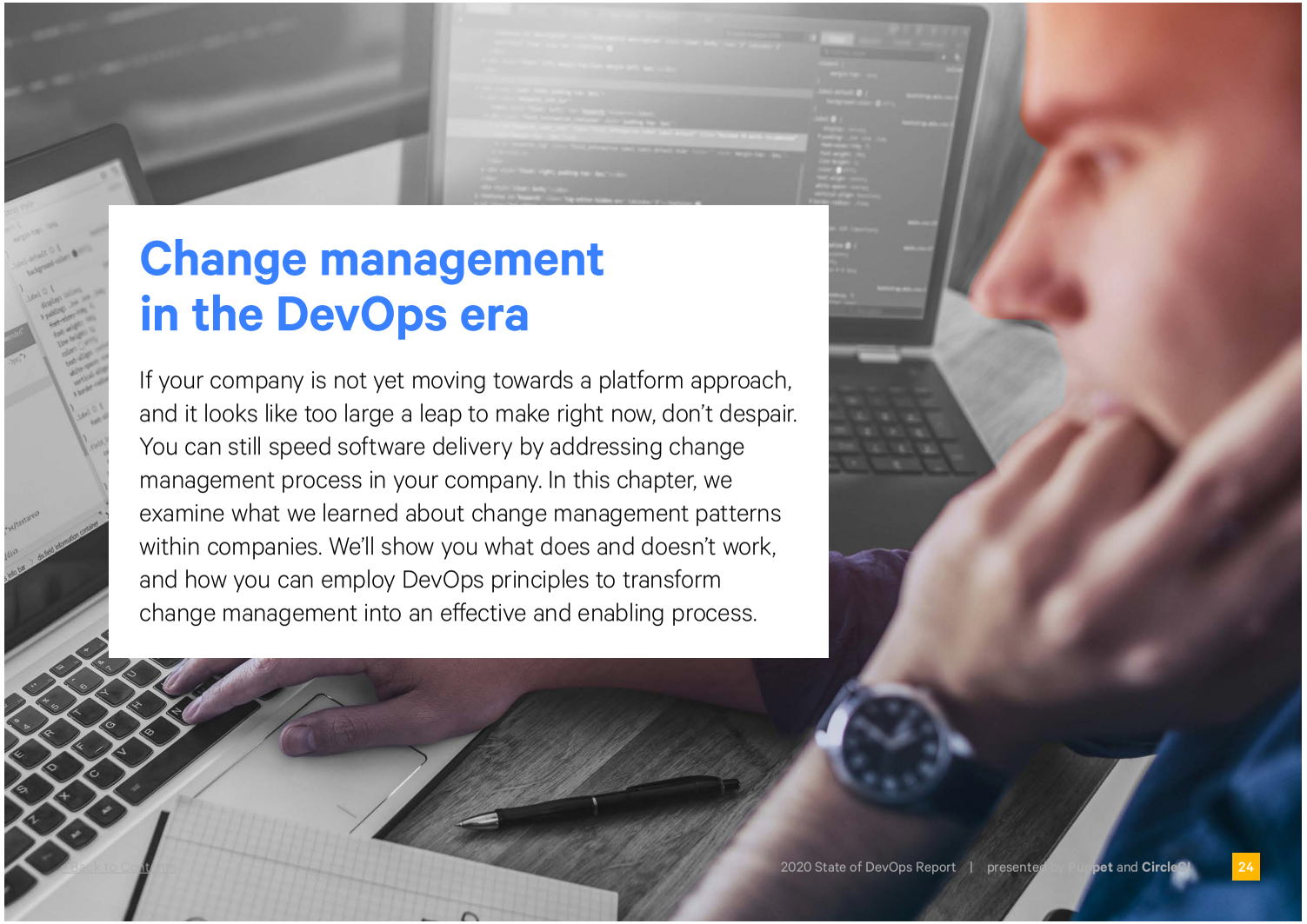
- 28-day rolling and/or seven-day rolling uptime
- Incident frequency
- Mean investigation time per incident (e.g., how long it takes to find out what went wrong)
- Percentage of services or capabilities with vulnerability-patching SLAs
- Cost per work unit
- Developer throughput rate
- Deployment rate
- Rollback rate
- Conformance metrics (how close a service is to using "paved roads," i.e., standards provided by the platform team)

Cost per work unit will always be pretty specific to the engineering or business goals, so they vary from company to company. For CircleCI, conformance metrics tell us how close a service is to using our most desired configurations, including paved roads for development, documentation, deployment, testing, rollback, canary deployments, dependency updating and a few other things.

When services have a higher conformance score, that means they are taking fuller advantage of the capabilities of our platform. This is the direction we want every service to move in.

— Mike Stahnke, Vice President of Platform, CircleCI

< Back to Contents



Change management in the DevOps era

If your company is not yet moving towards a platform approach, and it looks like too large a leap to make right now, don't despair. You can still speed software delivery by addressing change management process in your company. In this chapter, we examine what we learned about change management patterns within companies. We'll show you what does and doesn't work, and how you can employ DevOps principles to transform change management into an effective and enabling process.

In the past decade, we've seen DevOps practices upend the way software release teams work. Here are the most prominent changes.

From	To
Waterfall projects and big heavy releases.	Small batches delivered frequently, leading to more frequent deployments and faster cycle times.
Slow feedback cycles with a lot of manual reviews and approvals; long wait times.	Real-time feedback and metrics driven by automated workflows.
Process-heavy and time-intensive management of change requests. Having to provide context to approvers who are not directly involved with the work.	Collaborative software development, automated delivery pipelines and decisions made by teams doing the work.
Teams are organized by technology or functional boundaries. Manual handoffs between siloed teams. Misaligned incentives.	Early stakeholder involvement across the value stream at every stage of the delivery lifecycle: design, build, deploy, monitor and maintenance. Stakeholders include auditing, compliance, change management, security, network, storage, middleware and enterprise architects. Teams are aligned to business goals.

> **"The problem isn't change, per se, because change is going to happen; the problem, rather, is the inability to cope with change when it comes."**
>
> — Kent Beck, *Extreme Programming Explained: Embrace Change*

Even as we see delivery teams successfully shift their thinking and practices, it remains much harder to change deeply ingrained structures and processes across a large organization. Change management is one of the processes that is hardest to shift.

Pivoting to a new way of doing things requires leadership support, organizational discipline, and a ton of collaboration and alignment across every layer of the organization. But the large legacy environments that have evolved in most big organizations are not easy to pick apart and rework. They are often maintained by many different teams, each owning a piece of the technology stack. The teams that understand the work usually lack authority to approve their own changes; instead, change approval is often assigned to a committee (such as a change approval board) that is not involved with the actual work, and does not have as much understanding of it.

All these layers exist because the large legacy environment is where the organization's primary business lives. So any changes feel risky, and having lots of process and bureaucracy feels like you're keeping that business safe.

Unfortunately, all this process holds organizations back. They simply can't release software — whether it's for external or internal customers — fast enough to meet the needs of the business. Meanwhile, competitors who've made their change management more effective are able to release quickly and repeatedly, putting them way out in front.

Change management and ITIL

Many organizations that have been around awhile based their change management processes on the ITIL framework. Developed by the British government's Central Computer and Telecommunications Agency (CCTA) during the 1980s, ITIL (Information Technology Infrastructure Library) was a response to the agency's growing dependence on information systems. According to Axelos, the organization that took over ownership of ITIL in 2013, at least 90 percent of the Fortune 500 have adopted ITIL.

The purpose of ITIL is to help the business manage risk by:

- Improving alignment between IT and the business
- Increasing the quality of IT services while decreasing their cost

Ironically, in their efforts to implement ITIL, large organizations often created complex processes that require entire teams just to traffic and manage changes. Rather than improving alignment between IT and the business, many companies built cumbersome bureaucracies that take enormously long times to approve any change. Of course, this actually decreases the effectiveness of IT services while increasing their cost.

Interestingly, the ITIL world seems to have acknowledged these issues. The latest ITIL version, ITIL 4, departs significantly from previous versions with its focus on change enablement and guiding principles drawn from the key DevOps themes of collaboration, centering on value, rapid iteration and feedback.

DevOps evolution and change management effectiveness

We wanted to see whether change management effectiveness correlated with DevOps evolution. To measure change management effectiveness, we looked at three dimensions:

Implementation success. We looked at change failure rate and deployment frequency. Ideally, firms should be able to make changes much more frequently, recover quickly from failures and learn from them.

Level of efficiency. We wanted to know how efficient the change management process is, based on the following:

- A mandatory waiting period of less than two weeks
- Changes require only one approval
- Changes are implemented correctly and do not need to be backed out
- Approval by someone who has the right skills to make a proper assessment
- Little time required for documenting changes

Performance sentiment. As a proxy for objective evaluation of each respondent's organization, we developed this metric. We asked respondents whether their company's change management procedures:

- Reduce risk
- Reduce downtime related to service incidents
- Provide information that is useful to the organization
- Ensure that knowledge and information are shared with appropriate stakeholders
- Facilitate the rate of change our business needs
- Provide an appropriate level of review and approvals based on the evaluated risk level of a change

Together, these three dimensions — **implementation success, level of efficiency** and **performance sentiment** — make up our measure of **change management effectiveness**.

We found that change management effectiveness increases as organizations evolve their DevOps practices. While the differences aren't enormous, they are statistically significant.

We speculate that the differences are not dramatic because effective change management requires many components that stretch far beyond the specific DevOps practices we've mapped in our DevOps evolution model. These additional components include supplier relationships, incident management, risk management, value stream optimization, and more.

Most of the key practices in the DevOps evolution model revolve around standardization of tools, technology, configurations, infrastructure and patterns — work that is done by technology-oriented teams to lay the foundation for greater business agility and innovation. Now let's consider the 34 practices described in ITIL 4. There are just three technical management practices: deployment management; infrastructure and platform management; and software development and management. There are 13 *additional* service management practices, not to mention the 14 general management practices! Of course, there is overlap between many of these practices, but generally speaking, the DevOps evolution model represents a small slice of overall service management as it is commonly practiced in enterprise environments.

Nonetheless, there is a statistically significant relationship between DevOps evolution and change management effectiveness. In particular, the DevOps values of allowing employees a high degree of autonomy and involvement, sharing and communication, and collaboration, all play a role in effective change management.

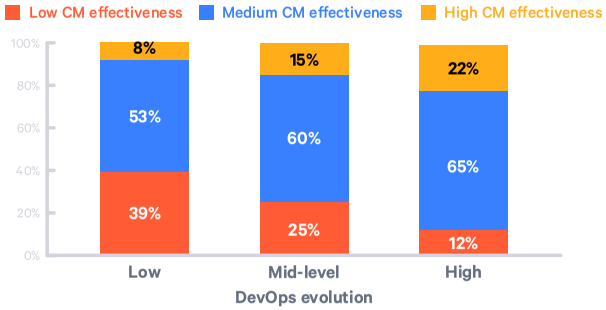
Approaches to change management

To investigate change management, we asked our survey respondents about a number of different practices in their workplaces. These can be sorted into two buckets: change approval processes, and the degree to which change implementation has been automated.

As we analyzed these answers, we discovered significant differences between survey respondents, and were able to categorize them into four clusters. Each cluster has a distinct approach to change management:

- **Operationally mature.** High levels of both process and automation.
- **Engineering driven.** High emphasis on automation.
- **Governance focused.** High emphasis on manual approvals and low emphasis on automation.
- **Ad hoc.** Low emphasis on both process and automation.

Change management effectiveness and level of DevOps evolution



Change approval processes

The questions we asked about change approval revealed two different approaches: orthodox and adaptive. We also looked at how frequently organizations evade their change management processes.

Orthodox change approval is based on strict adherence to established practices:

- Changes are approved by a committee (e.g., a change approval board).
- Approval is required from multiple levels of management.
- Changes can be made only in predefined windows.
- The person requesting the change cannot implement the change (separation of duties).

Adaptive change approval is based on input from teams that are close to the work. We call this "responsible autonomy."

- Changes are approved by the team implementing the change.
- Post-implementation reviews identify opportunities for improvement.
- Pre-approval is based on proof from the delivery team that the change can be made safely.

Evasion of change approval process happens in a couple of ways:

- Changes are approved without proper consideration (i.e. rubber-stamped).
- Team members explicitly bypass the change management process and there are no consequences.

What does "separation of duties" really mean?

Some companies implement controls to limit access to IT systems or require manual approvals, believing that regulations — for example, the Sarbanes-Oxley Act or SOC 2 — mandate separation of duties.

This is often interpreted to mean that people who can commit to a code repository must not be allowed to deploy that same code to production. Indeed, many auditors and security professionals are convinced that this is what the regulations say.

In reality, regulations can frequently be satisfied with the combination of:

- Automated deployment
- A requirement that someone other than the code author must review and approve the change
- Supporting controls such as strong audit logs and access control

If your automation efforts are being hamstrung by controls such as these, we suggest you focus on building a collaborative relationship with your auditors and risk management teams. Work together on *genuinely* satisfying regulatory requirements in an efficient and secure manner. We've seen very few people actually reach out to their risk teams to collaborate, but the ones that do nearly always succeed.

Degree of automation in change implementation

Change implementations can be highly automated, or not automated at all. Methods range from a manually implemented change with a secondary review — no automation — to a fully automated deployment of the change, with automated testing providing risk assessment during the rollout, and automated progression when the tests pass.

Below are the areas we asked about to determine the degree of automation applied to changes. The possible answers are below the bolded text.

Does the CI/CD process model traditional change management processes, or is it independent?

- Work going through the CI/CD system requires a ticket.
- Changes using CI/CD pipelines are not subject to traditional change reviews and processes.

Test and deployment automation: how changes move through the assurance process.

- Changes are run through automated acceptance tests.
- Changes are deployed automatically after automated tests pass.

Manual risk mitigation

- A person must manually review actions to be performed.

Automated risk mitigation includes more advanced deployment techniques to compartmentalize risk and enable changes while keeping services online. These include:

- Feature flags
- Blue/green deployments
- Canary deployments
- Active-active high availability clusters

How changes are typically written: Writing changes in code is a foundational practice that enables capabilities such as automated testing, automated deployment and automated risk mitigation. We asked survey respondents how their changes are normally written.

- Changes are written primarily in code.
- Changes are written primarily in prose.

Changes as code

When your changes are written in code, they may have the same properties as changes written in prose. But because they follow coding practices, your changes can be authored, tested, reviewed and deployed like code.

The real power here is that your changes can be subjected to any validation techniques that are available for code. Plus, you can roll forward to a previously stable implementation if the change does not go as planned.

Changes implemented in code also have the advantage of sometimes escaping the traditional — and slow — change management processes that most companies use. That's because they follow development deployment guidelines rather than an ITIL-based review process. Going through the development process instead of the change-review process usually results in greater velocity, and often more consistency.

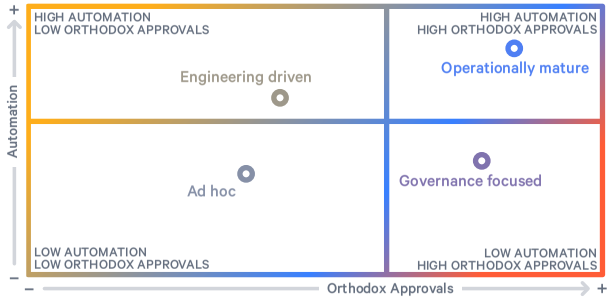
Common changes produced in code include changes through infrastructure automation tools, delivering new capabilities in software, or updating software packages.

The four approaches to change management

We plotted the four change management approaches on a matrix (see below). The x-axis represents orthodox approvals, and the y-axis represents automation, meaning a combination of automated test, automated deployment and automated risk mitigation.

Note that the junction of the four quadrants represents the average score of all four clusters. The top right quadrant represents "above average" for both orthodox approvals and automation, while the bottom left quadrant represents "below average" for both dimensions.

Change management approaches by level of automation and orthodox approval



What is toil?

In "Site Reliability Engineering: How Google Runs Production Systems" (essays written by a number of authors at Google), toil is defined as "work tied to running a production service that tends to be manual, repetitive, automatable, tactical work, devoid of enduring value, and that scales linearly as a service grows." Google's goal is to keep toil at less than 50 percent of each SRE's work hours, and for engineering work to account for at least 50 percent.

Engineering work includes both software engineering (writing or modifying code, including infrastructure code and automation scripts) and systems engineering (configuring production systems including monitoring setup, load balancing configuration, server configuration, etc).

For our survey respondents, the top four sources of toil were interrupts (non-urgent service-related messages and emails); urgent on-call response; deploying patches, releases and pushes; followed by building testing environments and logging. Respondents reported roughly the same sources of toil, regardless of the overall time they dedicated to engineering.

Our analysis also revealed that toil varied across teams. Site reliability engineering, platform engineering, DevOps teams and application development/software engineering teams spend more of their time on engineering — just 25 percent of respondents from these teams reported more than 40 percent of their work time goes to toil. By contrast, 40 percent of respondents from application security, cloud, and compliance and audit teams spend over 40 percent of their time on toil.

Information security, infrastructure/IT operations and quality assurance/quality engineering fall in the middle — about a third of these respondents spend over 40 percent of their work time on toil.

Each cluster has a distinct profile based on specific elements we asked our respondents about. These include the size of the companies and the industries they operate in; how they manage approvals; what gets automated and the degree of automation; and how much of the work week goes into toil (manual, repetitive work that can be automated, see also the sidebar on page 31, ***What is toil?***) versus engineering work. We've summarized each cluster's distinguishing features in the chart below, and further down, and you'll find a fuller discussion of each group.

Four approaches to change management: distinguishing features

	Operationally mature	Engineering driven	Governance focused	Ad hoc
Approvals	High orthodox approvals High adaptive approvals	Low orthodox approvals High adaptive approvals	High orthodox approvals Low adaptive approvals	Low orthodox approvals Low adaptive approvals
Automation	High test and deployment automation High automated risk mitigation	High test and deployment automation High automated risk mitigation Highest number of changes written in code	Low test and deployment automation Low automated risk mitigation	Low test and deployment automation Low automated risk mitigation
Industries	Technology Industrials & manufacturing Financial services Energy & resources	Technology Financial services	Technology Financial services Healthcare, Pharmaceutical and Life Sciences	Technology Education
Company size	Primarily mid-market organizations	Primarily smaller organizations	Primarily larger organizations	Primarily smaller organizations
Annual revenue: Small = under $100M Mid-market = $100M - $1B Enterprise = $1B+	 11% small 78% mid-market 10% enterprise	 37% small 24% mid-market 26% enterprise	 24% small 27% mid-market 33% enterprise	 43% small 17% mid-market 19% enterprise
Department	46% IT 37% InfoSec 17% Engineering	53% Engineering 39% IT 5% InfoSec	55% IT 37% Engineering 5% InfoSec	48% Engineering 44% IT 4% InfoSec
Engineering effort	63% report toil is over 30% of work hours 46% report engineering is more than half of work hours	67% report toil is 30% or less of work hours 47% report engineering is more than half of work hours	53% report toil is 30% or less of work hours 34% report engineering is more than half of work hours	64% report toil is 30% or less of work hours 37% report engineering is more than half of work hours

Distinguishing features of operationally mature companies

> ***"Even well-meaning gatekeepers slow innovation." — Jeff Bezos***

This group has the highest levels of both orthodox approvals and adaptive approvals. It also has the highest level of automation. These companies employ sophisticated risk mitigation techniques, yet also rely heavily on manual reviews. This may seem paradoxical, and indeed we were initially surprised by the high levels of both orthodox and adaptive approvals, as well as the seemingly contradictory high manual reviews and high degree of automation.

When we looked at the firmographic makeup of this group, these unexpected findings began to make more sense. We found that:

- The majority of respondents in this group work at larger mid-market organizations — 78 percent are at companies with revenue between $100 million to $1 billion.
- Almost two-thirds of respondents in this group (58 percent) work in large organizations (more than 500 employees). Half are at companies employing 500 to 1,000 people, and half at companies with 1,000 to 5,000 employees.
- They are more likely to work in old-line industries than in technology companies: energy and resources; financial services; and industrials and manufacturing.

Operationally mature organizations that have been around a while — such as larger, higher-revenue companies in traditional industries — have mature processes in place to safeguard their production systems. They are also big enough to have multiple business models and revenue streams.

Older industries have been disrupted by digital innovation, so digital transformation is recognized as a necessary strategy for remaining competitive. We think this helps explain why there's high use of automation, as well as manual approval, and why there's a mix of orthodox and adaptive approvals for this group.

Faced with the pressure to move fast and compete on one side, and the weight of old processes on the other, it's not surprising that respondents from operationally mature companies report it's common to evade change management procedures. Many changes get rubber-stamped, and teams regularly bypass their change management procedures without consequences.

When it comes to toil, operationally mature companies continue to present an interestingly mixed picture. Nearly two-thirds (63 percent) of respondents said they spend over 30 percent of their time on toil. However, nearly half (46 percent) spend more than half their weekly work hours on engineering.

Distinguishing features of engineering-driven companies

The engineering-driven group reported high adaptive approvals and low orthodox approvals. These companies scored high on automated testing and sophisticated risk mitigation, though a bit lower on deployment automation than we expected.

This group stands out from the others because the majority of their changes are written in code versus prose. This means they have the skills to automate repetitive operational tasks.

The majority of respondents work in an engineering or development department, with 34 percent part of an application, software development or engineering team. Twenty-five percent work in a DevOps team.

Not surprisingly, the majority of respondents work in technology companies — primarily small businesses with under $100 million in annual revenue. They reported less separation of duty than both the governance-focused and operationally mature groups, and also that teams in their companies have more autonomy to approve their own changes.

Given higher levels of automation and less cumbersome approvals, we weren't surprised to find that respondents in engineering-driven companies reported the lowest level of toil and highest level of engineering work of all our clusters. Two-thirds (67 percent) said toil accounts for 30 percent or less of their weekly work hours, and almost half (47 percent) report spending more than half their hours on engineering.

Distinguishing features of governance-focused companies

Respondents in this group skew towards larger organizations, with 40 percent working in companies that employ more than 5,000 people. Thirty-three percent of respondents are at organizations with annual revenue above $1 billion. The highest industry representation is technology, at 29 percent, with financial companies next at 20 percent.

This group relies heavily on orthodox approvals and manual reviews. These companies scored low for automation across the board — testing, deployment and risk mitigation. They make up for lack of automation with human oversight, which slows them down even more.

Unlike the operationally mature group, which also relies heavily on process, the governance-focused companies tend not to evade the process — they accept it. Few changes are reviewed post-implementation to identify opportunities for improvement, and few changes are pre-approved, suggesting a culture where continuous improvement is not valued.

Toil is high for the governance-focused companies. Nearly half (47 percent) of respondents in this group said more than 30 percent of their work is toil. They spend the least amount of time on engineering work, with just a third (34 percent) reporting more than 50 percent of weekly work hours spent on engineering.

Distinguishing features of ad hoc companies

This group stands in stark contrast to the operationally mature companies. They score low across all dimensions: automation, approvals and engineering work. Results for this group show their approach to change management is indeed ad hoc: They have very little orthodox approval process, yet rely heavily on manual review.

For ad hoc companies, the team has autonomy to approve and implement changes, most likely because the structure is light and the number of stakeholders is small. As a result, change success is largely dependent on employee competencies rather than carefully thought-out policies.

Of this group, the plurality of respondents (43 percent) work at firms with annual revenue under $100 million, and 57 percent of respondents are at firms with fewer than 1,000 employees. The majority of respondents work in technology and education, so they don't operate under the same regulatory constraints that drive the governance-focused group.

Because this group is made up of smaller organizations, we surmise these respondents operate in companies where communication is easier and more immediate, simply because there are fewer engineers. It's possible these companies also are subject to less pressure from regulatory constraints, so they feel less need to formalize approval processes or automate.

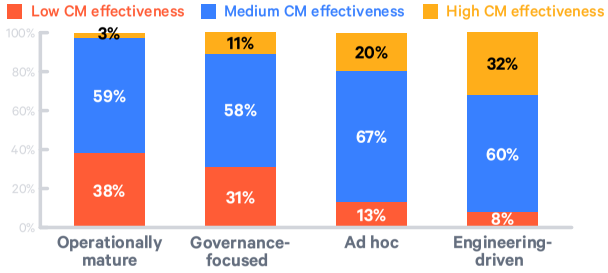
Over a third (34 percent) of respondents in ad-hoc companies reported spending more than 30 percent of their time on toil. A slightly larger number (37 percent) also reported spending over 50 percent of their time on engineering work.

What drives change management effectiveness?

When we looked at change management effectiveness in aggregate for each group, it's not surprising we discovered the engineering-driven companies had the highest level of change management effectiveness. The ad hoc companies came in second for high effectiveness. This is because their levels of inefficiency are low due to lack of process and relatively high implementation success. The remaining two groups, which rely heavily on orthodox approvals, did not score high on effectiveness.

Our data reveals several important takeaways about the factors that influence change management effectiveness and efficiency. Below these key takeaways, you'll find a fuller discussion of each cluster.

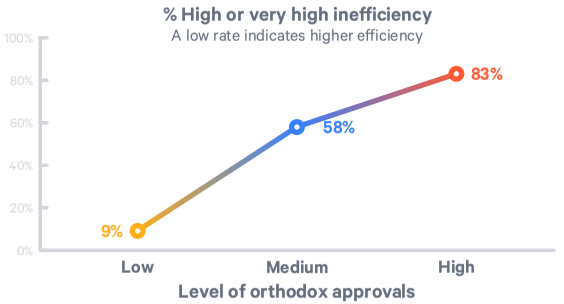
Change management approaches and level of effectiveness



Orthodox approvals make you less efficient

Orthodox approvals make the change management process less efficient. Firms with high orthodox approvals are nine times more likely to be inefficient than firms with low orthodox approvals. The correlation is clear and strong.

Level of orthodox approvals and rates of high inefficiency

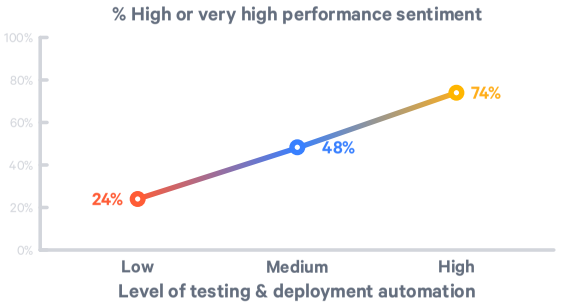


Automation gives teams confidence in change management

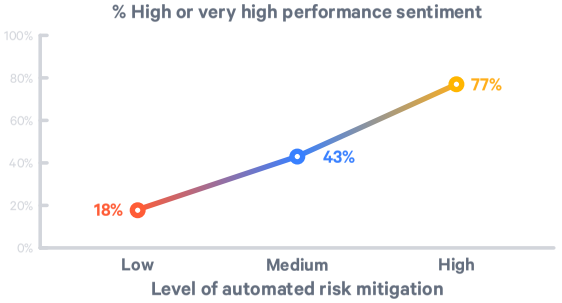
Automated testing and deployment and advanced risk mitigation techniques are strongly correlated with performance sentiment. Teams that automate and practice advanced risk mitigation believe that their change management process adds value in the following ways:

- Reduces risk, and provides an appropriate level of review and approval based on evaluated risk for any change
- Reduces downtime to services
- Provides useful information
- Ensures that knowledge and information are shared with stakeholders
- Facilitates a pace of change that advances the business

Level of test & deployment automation and performance sentiment



Level of automated risk mitigation and performance sentiment

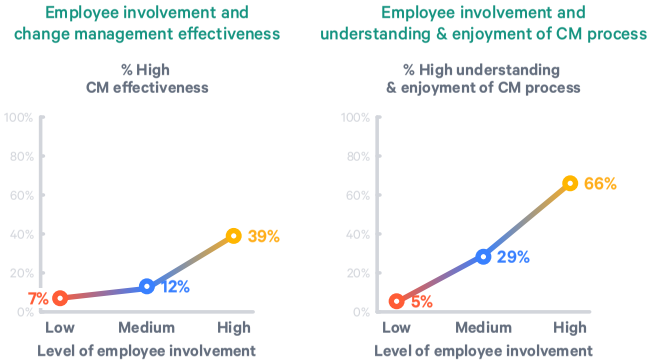


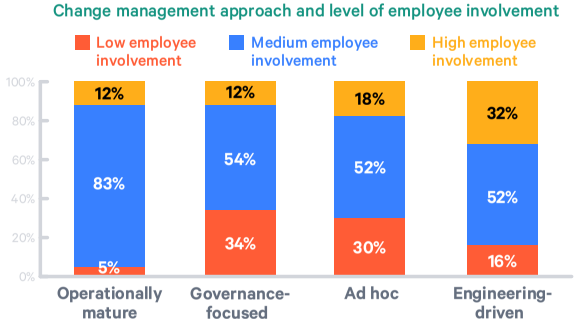
Giving people agency over the process results in higher effectiveness

Employee involvement in the change management process correlates strongly with effective change management. The more input and influence employees have over their change management processes, the more they understand and enjoy using them, which helps explain why they're seen as more effective.

We found that:

- Firms with high employee involvement in the change management process are more than five times as likely to have highly effective change management than firms with low employee involvement.
- Employees who report high involvement are also 13 percent more likely to understand and enjoy the process.
- Engineering-driven organizations involve employees the most in the change management process, and governance-focused companies the least.
- Governance-focused companies had the lowest employee involvement, reflecting their bureaucratic nature. When organizations rely on process and centralized decision-making rather than assigning more responsibility to employees, people feel disempowered and unengaged, with little or no motivation to challenge the status quo. This of course makes it much harder for an organization to change and evolve.
- Engineering-driven organizations tend to involve employees much more in the change management process — an attribute that's typical of workplaces where the DevOps principles of feedback loops and continuous learning are valued.





Operationally mature companies

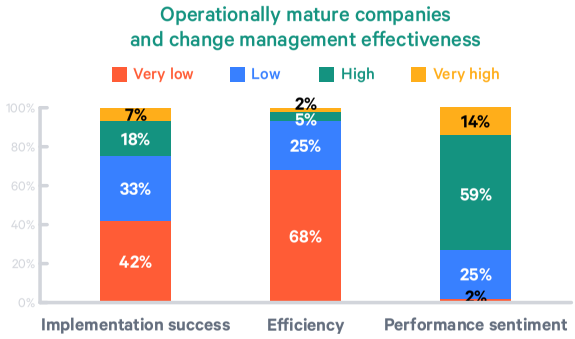
Operationally mature organizations scored low on implementation success, with lower deployment frequency and change success rate, and longer lead time for changes. They also had the lowest levels of efficiency, with 93 percent reporting low or very low efficiency.

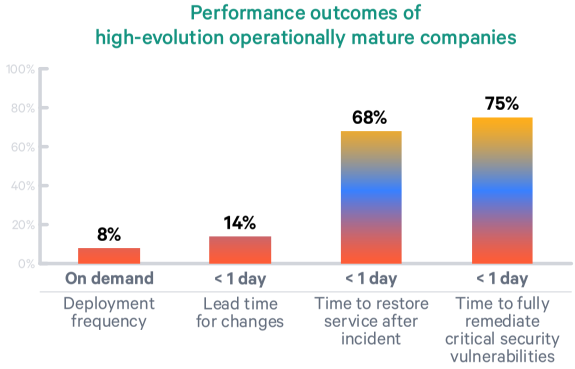
We wanted to see whether companies in this group that have reached a high level of DevOps evolution perform any differently from the rest. As it turns out, the highly evolved operationally mature organizations excel when it comes to remediating security vulnerabilities in less than one day — 75 percent of them can do this. A large majority (68 percent) are able to restore services less than a day after an incident. These teams clearly place a high value on keeping systems running, reacting to incidents and events quickly as they occur.

We surmise that companies in this highly evolved, operationally mature group have a lot of consumers and customers relying on their products or services, and therefore have a lower tolerance for change-induced risk. These organizations value uptime over new feature delivery or more frequent deployments. We shouldn't be surprised: Every deployment presents the risk of a service interruption.

We were surprised that respondents from the operationally mature companies had the highest performance sentiment of all four clusters. People in these organizations believe that their change management process really is getting the job done. Perhaps this belief in high performance — despite demonstrably lower performance — reflects a culture that deplores risk, and therefore values stability more highly than rapid throughput. It may also be the simple human tendency to believe that when you've put in a lot of effort, and have lots of documentation and artifacts to show for it, you've improved things.

The flip side of having such standardized and formal processes is that it's harder to adapt them to changing business needs. So they become a drag on progress and innovation.





Engineering-driven companies

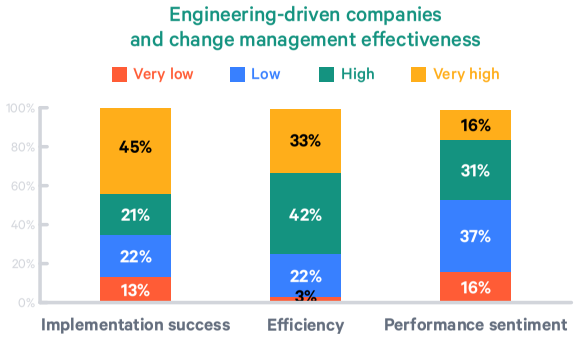
In contrast to the operationally mature organizations, engineering-driven companies have the greatest implementation success and highest levels of efficiency. Oddly, though, respondents in these companies perceive their change management processes to be less effective.

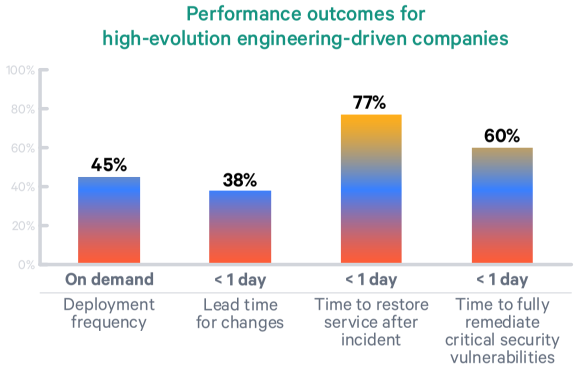
This paradox makes sense when you consider that engineering-driven companies deploy far more frequently than companies in the other clusters. If you're making 500 changes per week, for example, and even 1 percent of these changes fail, that's one incident per day for the team to deal with. Low as that percentage is, one incident per day can still feel like you're having a lot of failures.

Another observation: Unlike operationally mature organizations that accept the status quo, engineering-driven companies tend to value continuous improvement, which presupposes taking a critical view of the status quo.

Our stats on the engineering-driven organizations that are highly evolved in their DevOps journeys demonstrate that these companies value releasing new features quickly to their customers. Forty-five percent are able to deploy on demand, and 38 percent can deploy a change in less than one day.

Because they have automated so much of their delivery process, restoring service and remediating security vulnerabilities is faster, too. Seventy-seven percent restore service after an incident in a day or less, and 60 percent fully remediate critical security vulnerabilities in less than one day.

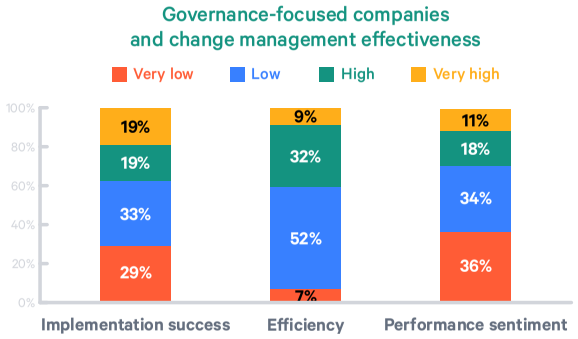


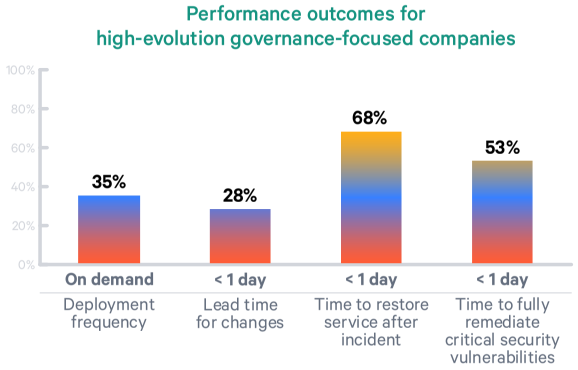


Governance-focused companies

The governance-focused organizations in our survey tended to be larger companies in financial services and healthcare, both highly regulated industries. These companies scored very low for implementation success, and their employees reported low levels of efficiency along with low performance sentiment. This gloomy view may be simply accurate. It may also be due to the risk-averse, audit-conscious culture of highly regulated companies.

The governance-focused organizations that are also highly evolved in their DevOps journeys are quick to restore service — 68 percent can restore service after an incident in less than one day. Compared to operationally mature companies, governance-focused companies don't do as well with remediating critical security vulnerabilities, though 53 percent are able to fully remediate security vulnerabilities in less than one day. However, these organizations are able to deploy changes on demand more frequently than operationally mature companies — 35 percent can deploy on demand.





Ad hoc companies

Companies in this cluster are smaller and less regulated, and many respondents in this cluster work in education. Despite good scores on implementation success and high scores for efficiency, respondents' overall performance sentiment was not positive.

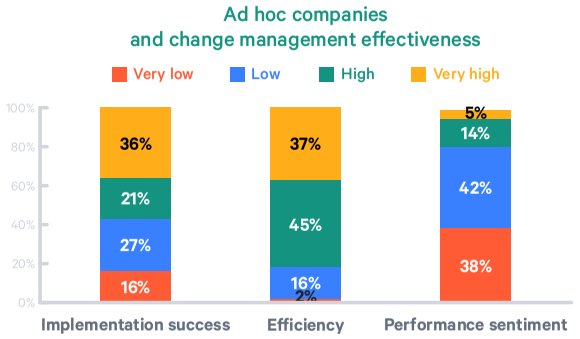
The number of ad hoc companies that were also highly evolved in their DevOps journeys was too small to make a reasonably sized sample. So we aren't reporting on this group's performance stats.

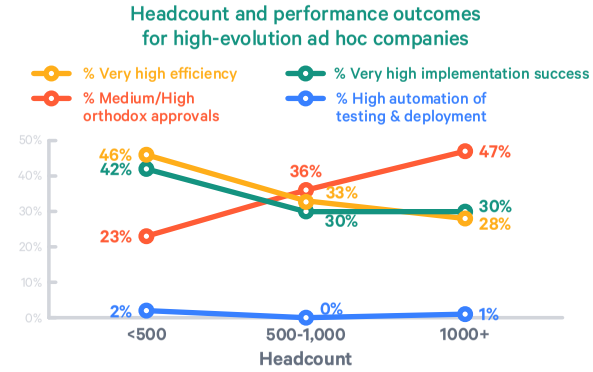
We've spent some time discussing the ad hoc companies and their behaviors, because it seems obvious to us that most companies start out as ad hoc organizations. They're new, they're moving fast, and software delivery teams are small and intimate enough that they don't need a lot of process — they can just talk to each other as they deploy changes.

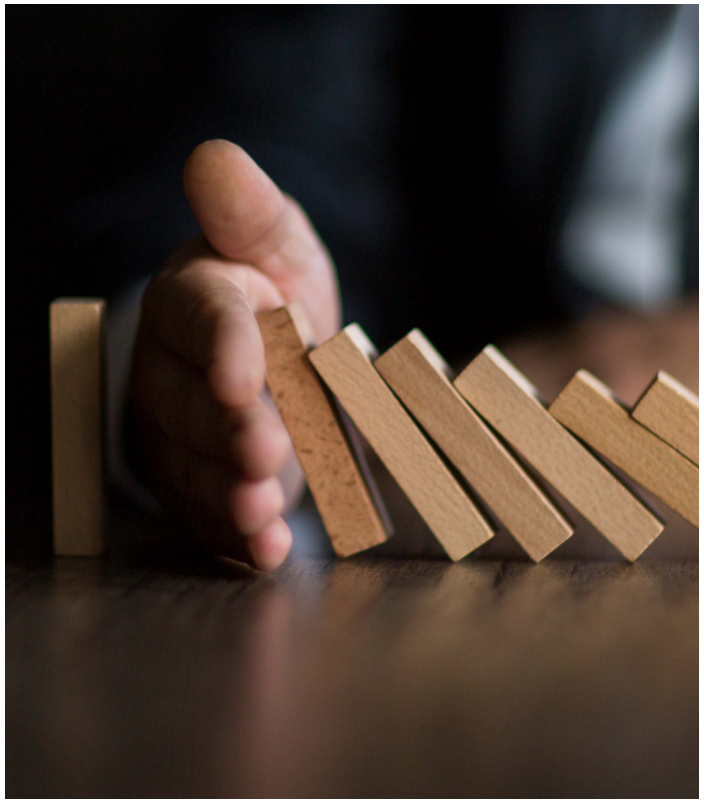
The need to move quickly may be why this group has negative feelings about their change management processes — they may feel that any process gets in the way of their work, and that most of it is unnecessary.

What happens as companies grow and become less ad hoc? Our survey data shows that companies with higher headcount have lower efficiency and lower performance.

In the chart to the right, you can see that as headcount increases, there is a corresponding decrease in implementation success and efficiency; also, orthodox approvals increase. Automation remains flat as headcount grows. It seems that the common path is to add more processes to minimize risk, instead of focusing on automation.







Challenges to automating the change management process

It's clear that automating the change management process, shifting approvals to the teams that do the work, and enabling people to have a voice in the process all make change management more effective, and improve software delivery performance (more frequent delivery, fewer failures).

Automating change management does present challenges, though. It can't be done with a simple tweak or two from one person, or even one team. Change management is, as we've seen, complex; automating it requires structural changes that are possible only when there's shared focus and collaboration across multiple teams or departments, as well as their leaders.

We asked respondents about their challenges in automating the change management process. Their answers revealed some similarities between our four groups, as well as some interesting differences.

Top challenges reported by all groups

Incomplete test coverage was the top response across all groups, followed by organizational mindset and tightly coupled application architecture.

Incomplete test coverage. Writing good tests that cover every possible scenario is nearly impossible, especially in complex environments where there is an endless number of user behaviors, dependencies, dynamic architectures and more. Authoring tests is difficult: You have to deeply understand the work the service does, what the user will do with the service and what a user who doesn't know much about the service might do with it. Teams often do a bit of testing, such as unit tests or a set of early integration tests, but beyond that, they don't have behavior defined well enough to write comprehensive tests for it.

For a fully automated deployment, teams may want a lot of tests to pass before going into staging or production: unit, integration, systems, performance and user acceptance tests. Many organizations, however, don't invest this deeply in testing, so they aren't confident enough to move to fully automated deployment.

Organizational mindset. It's no surprise that organizational mindset is a common challenge across each group. We hear this a lot in our work with companies, and it's the one thing senior leaders and practitioners agree on.

Why is it so hard to overcome organizational inertia? Often, organizations will try to copy the technical practices of DevOps leaders, neglecting the cultural aspects that make DevOps transformations successful. When we work with teams that are resistant to change, it often takes time to build trust between teams and departments; people also need evidence that the changes will be beneficial. The people who are pushing the change also need to make it easy for those adopting the change to do the right thing.

Tightly coupled application architecture. Tightly coupled application architecture is a major constraint for delivery teams. Updating their application or service requires coordination with other teams, slowing down delivery for each team due to complex dependencies. Loose coupling means that applications are more modular, so teams can deliver at their own pace using their own workflows. Testing becomes more manageable, and teams have more freedom to experiment.

Backlog coupling: Why reducing team dependencies matters

From Evan Bottcher's post, "What I Talk About When I Talk About Platforms" on MartinFowler.com

> *At an Australian telecommunications company, my colleagues did a study of hundreds of pieces of work or tasks passing through a delivery centre. Some tasks could be completed by a single team without dependency, specifically without scheduling work by members of another team. The tasks that had to wait for another team were 10x-12x slower in elapsed time. So dependencies have a real significant impact.*
>
> *This hurts us in many ways: It hurts in pure throughput and responsiveness to customer need, and drives us towards more long-term planning to more efficiently manage dependencies. It also damages a team's own accountability for outcomes, and for many teams I've observed this is a motivation-killer. Teams can find it easy to shift blame and stop seeking their own continuous improvement.*

Top challenges by change management approach

Operationally mature organizations have lower risk tolerance. Due to customer expectations and regulatory constraints, deploying multiple times a day just isn't feasible for some industries and products. By adopting CI/CD practices, you can ensure that you're always able to deploy on demand, even if your customers can't consume all of your changes. If regulatory constraints are preventing you from making changes faster, see the sidebar on page 29 *What does "separation of duties" really mean?*

Both engineering-driven and operationally mature organizations feel constrained by their application architecture. In our 2015 State of DevOps Report, we found that certain architectural characteristics correlate with high performance:

- Ability to test without an integrated environment
- Ability for devs to get comprehensive feedback from automated tests
- Ability to deploy an application independent of services it depends on
- Use of a microservices architecture

"Applications are rearchitected based on business needs" is a key practice in Stage 5 — the highest stage — of our DevOps evolution model. While rearchitecting applications is not easy, it does become significantly more achievable after standardization, automation and team autonomy have been established. It's worth noting that breaking apart a monolithic application into microservices may be what the business ultimately needs, but smaller architecture changes (such as replacing a home-grown message queue with a modern open source component or cloud-based service) can also add tremendous value.

Governance-focused organizations lack trust across functional teams. Lack of trust often goes hand in hand with fear of change. To build trust and reduce fear, we've seen organizations successfully take a two-pronged approach: automate change, and engage early with the change management team to show that changes can be made safely with automated processes. NatWest Group did just that: learn more by watching the webinar Modernising Change and Release Management: Real Life Examples with RBS Group.

Ad hoc organizations cited lack of skills as an inhibitor. These smaller organizations often lack enough people to do all the work, so they tend to prioritize urgent requests over important long-term optimizations. Systems thinking and global optimization require dedicated focus and time, which can be scarce if you're constantly fighting fires or you're the sole person in your role. When hiring is not an option, prioritizing problems based on highest potential return can help build the necessary skills, while also freeing up time to work on the most important problems. One example of this: standardizing processes around production deployments, or automating a few common repetitive tasks.

Top challenges for each cluster in priority order

Respondents in the operationally mature cluster gave equal weight to all challenges.

Engineering driven	Operationally mature
• Incomplete test coverage • Organizational mindset • Tightly coupled application architecture	• Tightly coupled application architecture • Customer risk tolerance • Incomplete test coverage • Regulatory constraints
Ad hoc	**Governance focused**
• Incomplete test coverage • Organizational mindset • Lack of skills within an organization	• Organizational mindset • Incomplete test coverage • Lack of trust across functional teams

Applying DevOps principles to change management

It's clear that improving change management presents a lot of challenges for teams. DevOps principles first arose — and DevOps practices have evolved — to deal with exactly these types of cultural and structural challenges. We've seen teams successfully modernize their change management practices through the application of fundamental DevOps principles, so here are our recommendations.

Break down silos and build empathy. Engage with your change management, release management, audit and compliance teams. Understand their fears and motivations, respect their roles, and learn to use their vocabularies to describe the capabilities your teams can provide.

People who work in these functions tend to be rational and detail-oriented. In our experience, they are more than willing to collaborate on a more efficient process that manages risk for the company. They respond well if you demonstrate that you can perform bounded experiments in low-risk environments, and provide a plan for iterating on those experiments.

Create feedback loops. Look to build feedback loops from the people who are bound by change management policies to the people who are responsible for defining them.

Feedback loops should cover not only sentiment from IT teams, but also include the introduction of new capabilities. Are you introducing significant new testing capabilities for your infrastructure-as-code deployments? Removing all manual deployments and automating them as part of a CI/CD pipeline? Inform your change management teams about upcoming plans, and offer them the opportunity to collaborate and shape your designs. You'll be making your job easier — and ultimately, theirs too.

Measure the impact of your new approach. It goes almost without saying that you must establish the metrics you'll track to prove whether things have improved. Are you saving time or money? Have you eliminated any wasteful handoffs or toil? Are your stakeholders satisfied — or even happy — with the new approach?

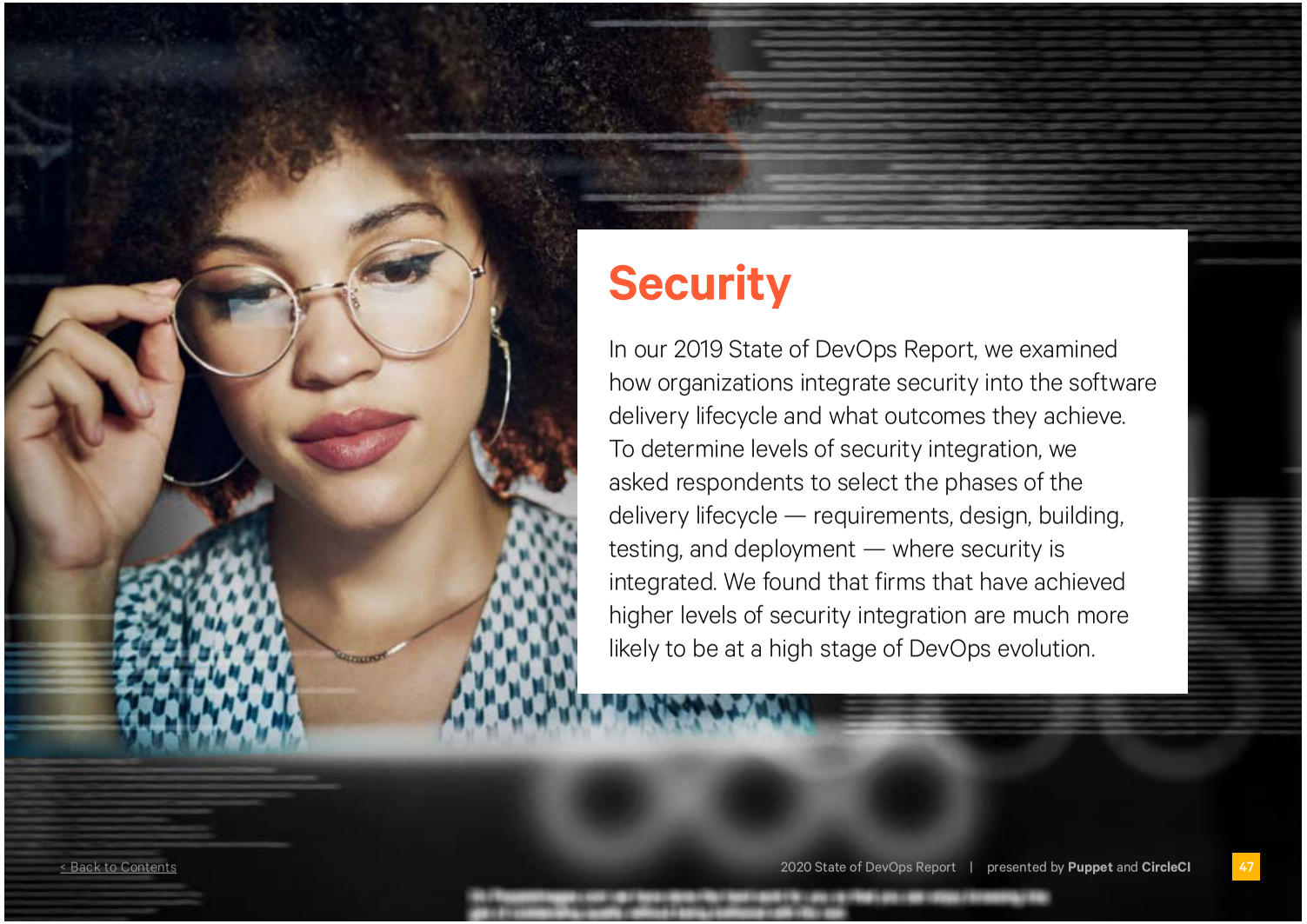
All this information should be made visible for the sake of continuous improvement — and just as important, to deliver much-deserved satisfaction and recognition to the people who have worked hard for these improvements.

As we like to say in DevOps, "Make your success visible."

Will Larson, author of "An Elegant Puzzle," gave a terrific talk titled Investing in technical infrastructure. We particularly like this quote:

> *"If you think about infrastructure teams who view themselves as just a service provider or who view themselves as a cost center... many of them don't have a vision of what they're trying to ladder up to, so they can only get incremental wins. They can never get evolutionary or order-of-magnitude wins that come from composing small series of changes into a broader vision."*

If you want pragmatic tips to help dig yourself out from firefighting and do more innovation work, be sure to check out Will's talk.



Security

In our 2019 State of DevOps Report, we examined how organizations integrate security into the software delivery lifecycle and what outcomes they achieve. To determine levels of security integration, we asked respondents to select the phases of the delivery lifecycle — requirements, design, building, testing, and deployment — where security is integrated. We found that firms that have achieved higher levels of security integration are much more likely to be at a high stage of DevOps evolution.

With 14 percent of our 2020 survey respondents reporting working in an infosec department, we wanted to see how things had changed from last year. In comparing the percentage of respondents at each level of security integration, we are encouraged to see more organizations integrating security into two to four stages.

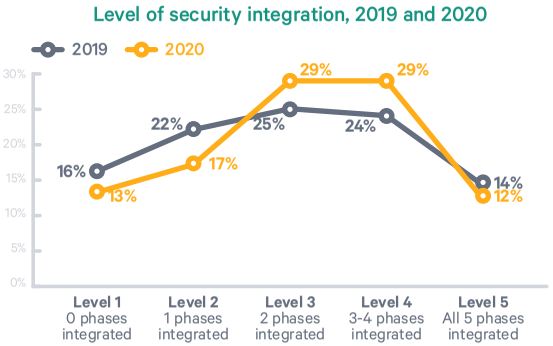
We are also happy to see that security integration is strongly correlated with the ability to quickly remediate critical vulnerabilities. Of those with low integration, 25 percent can remediate vulnerabilities within one day, compared to 45 percent of those with full security integration.

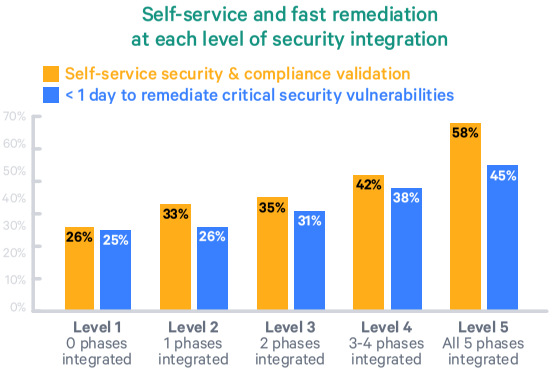
Finally, the self-service offering of security and compliance validation is positively related to level of security integration. Those with full security integration are over twice as likely as those with no security integration to offer security and compliance validation as a self-service capability.

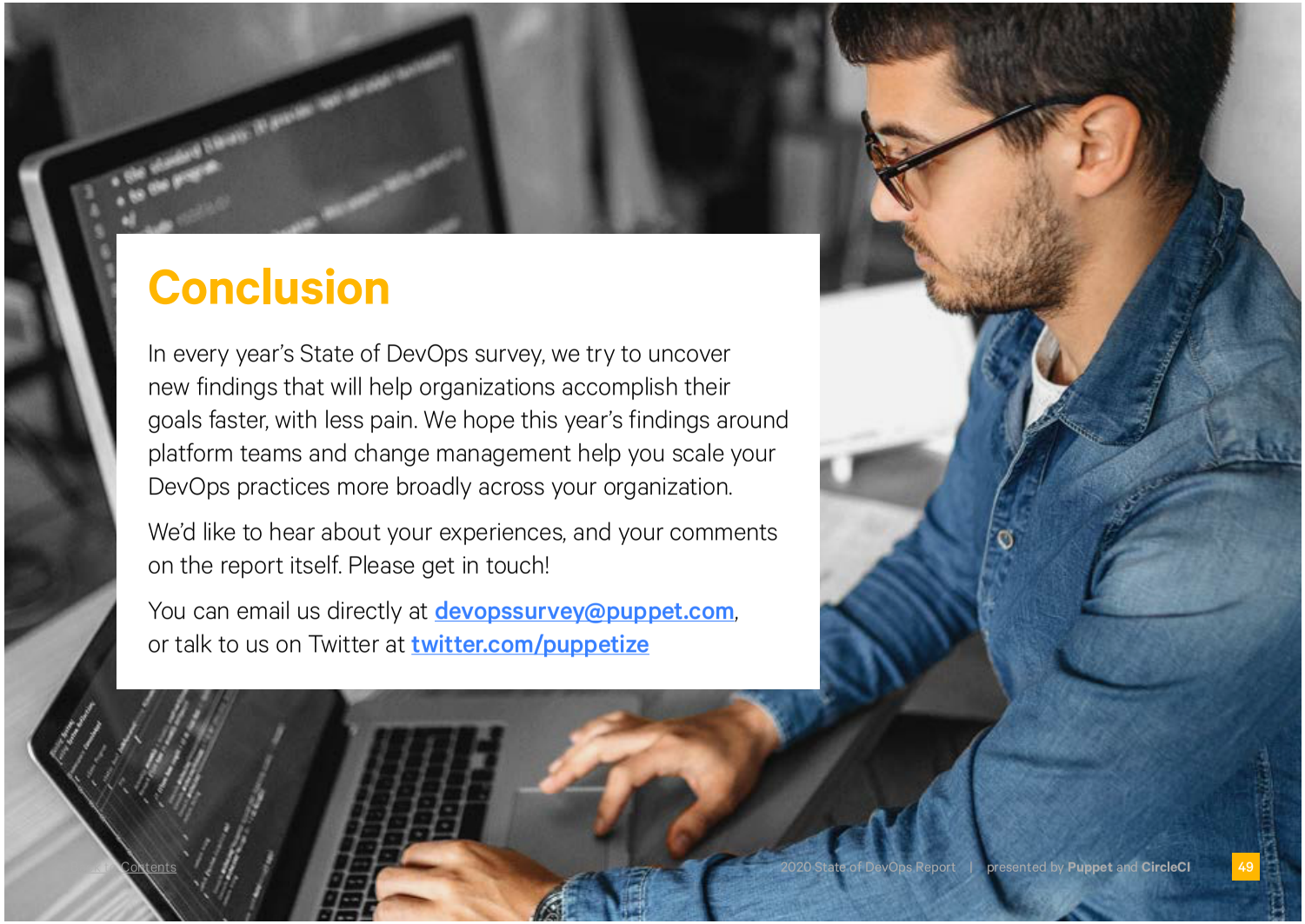
If improving security posture is a top priority for your organization — and really, who doesn't want to improve security? — then we stand by what we said in our 2019 report:

> *"Integrating security at every stage of the software delivery lifecycle is more than just shifting security checks to the left. Security integration requires a completely different approach, one that emphasizes cross-team collaboration and empowers delivery teams to autonomously prevent, discover and remediate security issues. Breaking down knowledge silos between teams, and collaborating to improve security both raise overall awareness of security concerns, making it more likely that everyone — even those outside the security team — will adopt known patterns for security protection."*

This doesn't just apply to security, of course, but to all the functions in the software process.







Conclusion

In every year's State of DevOps survey, we try to uncover new findings that will help organizations accomplish their goals faster, with less pain. We hope this year's findings around platform teams and change management help you scale your DevOps practices more broadly across your organization.

We'd like to hear about your experiences, and your comments on the report itself. Please get in touch!

You can email us directly at devopssurvey@puppet.com, or talk to us on Twitter at twitter.com/puppetize

Author biographies

Alanna Brown

Twitter: @alannapb

Alanna is senior director of community and developer relations at Puppet, where she's had the privilege of helping Puppet grow from a small startup to a global brand with thousands of customers around the world. She conceived and launched the first annual State of DevOps Survey in 2012, and has been responsible for the survey and report since then. In addition to heading up DevOps research, Alanna is also responsible for driving awareness, adoption and advocacy for Puppet's product portfolio.

Nigel Kersten

Twitter: @nigelkersten

Nigel is field CTO at Puppet, responsible for bringing product knowledge and a senior technical operations perspective to Puppet field teams and customers, working on services strategy, and representing the customer in the product organization. He also works with many of Puppet's largest customers on the cultural and organizational changes necessary for large scale DevOps implementations. Nigel has served in a range of executive roles at Puppet across product and engineering over the last nine years, and came to Puppet from the Google SRE organization, where he was responsible for one of the largest Puppet deployments in the world.

Michael Stahnke

Twitter: @stahnma

Michael is vice president of platform engineering at CircleCI. Prior to this, he was at Puppet, running engineering for Puppet Enterprise, open source Puppet, and SRE. Prior to Puppet he was an infrastructure architect, team lead and open source evangelist at Caterpillar Inc., where he spent inordinate amounts of time with auditors. He was also an author for the State of DevOps Report in 2018 and 2019. Michael helped get the Extra Packages for Enterprise Linux (EPEL) repository off the ground in 2005, is the author of Pro OpenSSH (Apress, 2005), is an organizer of Devopsdays Madison, and rants continuously about technology, humans, and computers, while striving to learn more about them.

About Puppet

Puppet is driving the movement to a world of unconstrained software change. Its revolutionary platform is the industry standard for automating the delivery and operation of the software that powers everything around us. More than 40,000 companies — including more than 75 percent of the Fortune 100 — use Puppet's open source and commercial solutions to adopt DevOps practices, achieve situational awareness and drive software change with confidence. Headquartered in Portland, Oregon, Puppet is a privately held company with more than 500 employees around the world. Learn more at puppet.com.

About ServiceNow

ServiceNow (NYSE: NOW) gives you the power to make work, work better. Our cloud-based platform and products streamline and simplify how work gets done. We help you to scale DevOps to the enterprise, leveraging what you already have to drive rapid innovation and business value through automation and improvements to the developer experience. ServiceNow works for you. To learn more, visit servicenow.com/products/devops.html

About CircleCI

CircleCI is the leading continuous integration and delivery platform for software innovation at scale. With intelligent automation and delivery tools, CircleCI is used by the world's best engineering teams to radically reduce the time from idea to execution. CircleCI was named a leader in cloud-native continuous integration by Forrester in 2017 and 2019, and has been named to multiple Best DevOps Tools lists.

About Sysdig

Sysdig is driving the secure DevOps movement, empowering organizations to confidently secure containers, Kubernetes and cloud services. With the Sysdig Secure DevOps Platform, cloud teams secure the build pipeline, detect and respond to runtime threats, continuously validate compliance, and monitor and troubleshoot cloud infrastructure and services. Sysdig is a SaaS platform, built on an open source stack that includes Falco and sysdig OSS, the open standards for runtime threat detection and response. Hundreds of companies rely on Sysdig for container and Kubernetes security and visibility. Learn more at www.sysdig.com.

Who took the survey

As we have for the past nine years, we sought survey respondents from as wide a range of geographic regions, industries and company sizes as possible. We also hoped for balanced gender representation.

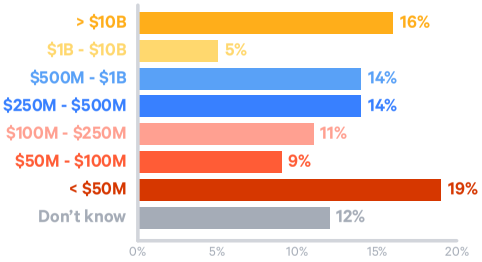
We feel lucky to have received more than 2,400 responses at a time when people have been pressured by restrictions due to COVID-19, including working from home while simultaneously supervising children's education and looking after family members.

To all 2,415 of you who responded to this year's survey, our heartfelt thanks.

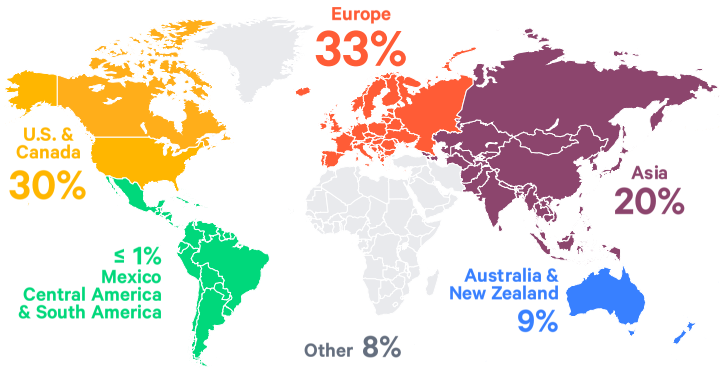
And to all of you reading this report, may you stay well and productive, wherever you are.

Organization annual revenue

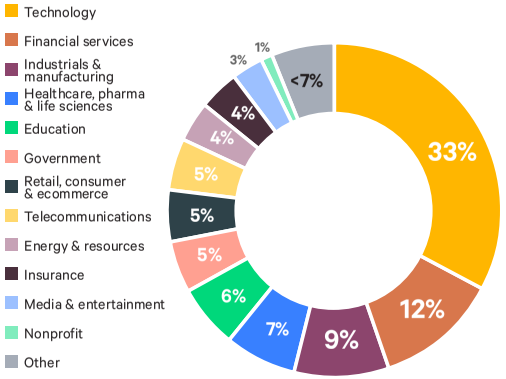
Expressed in USD



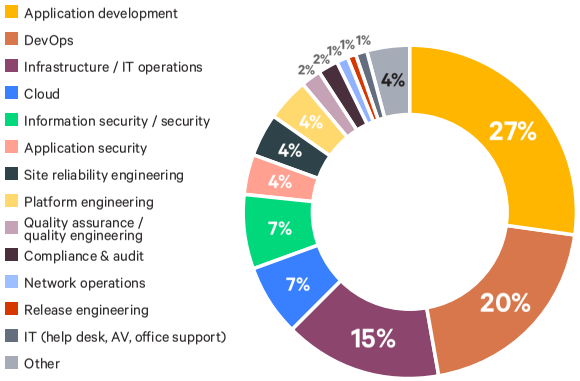
Responses by global region



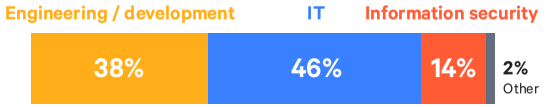
Principal industry



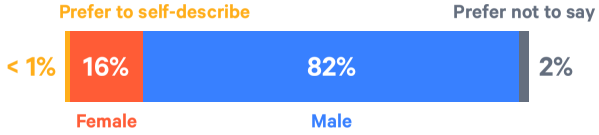
Team



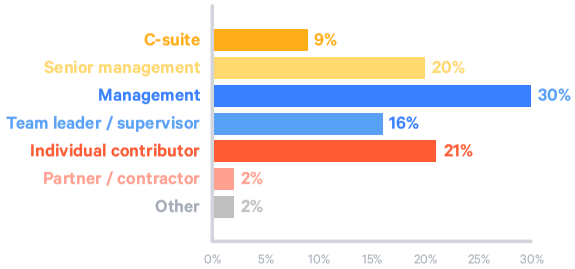
Department



Gender identity



Role within organization




2020
State of
DevOps Report