PLATFORM ENGINEERING
PULSE REPORT

How organizations implement Platform Engineering today, what's working, what isn't, and why

2025

Octopus Deploy

Contents

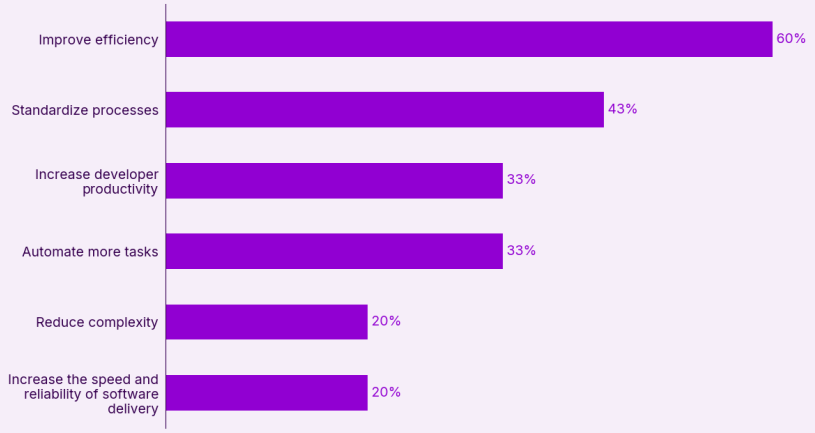
Executive summary

This Platform Engineering Pulse report examines the current state of Platform Engineering adoption across industries, analyzing how organizations implement these initiatives in practice versus theoretical ideals.

Data was collected from technical professionals across diverse roles and industries, focusing on adoption drivers, platform features in use, success metrics, and strategy effectiveness across multiple organizational perspectives.

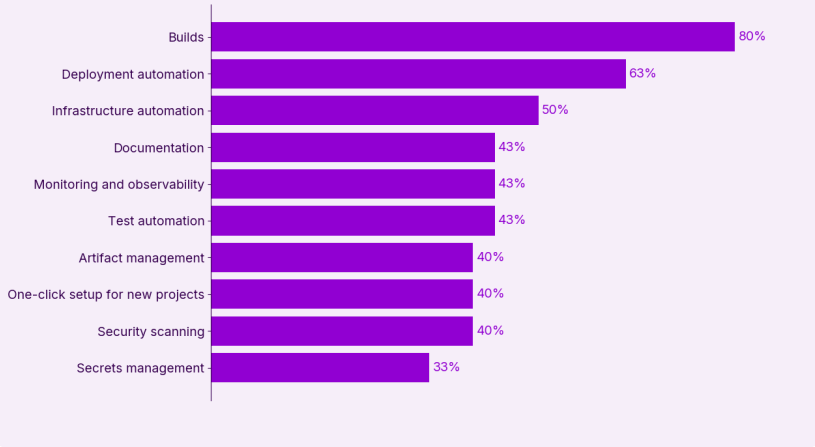
This report is a benchmark for understanding current Platform Engineering applications and provides a roadmap for organizational maturity and evidence-based decision-making during the platform adoption journey.

1. Adoption drivers



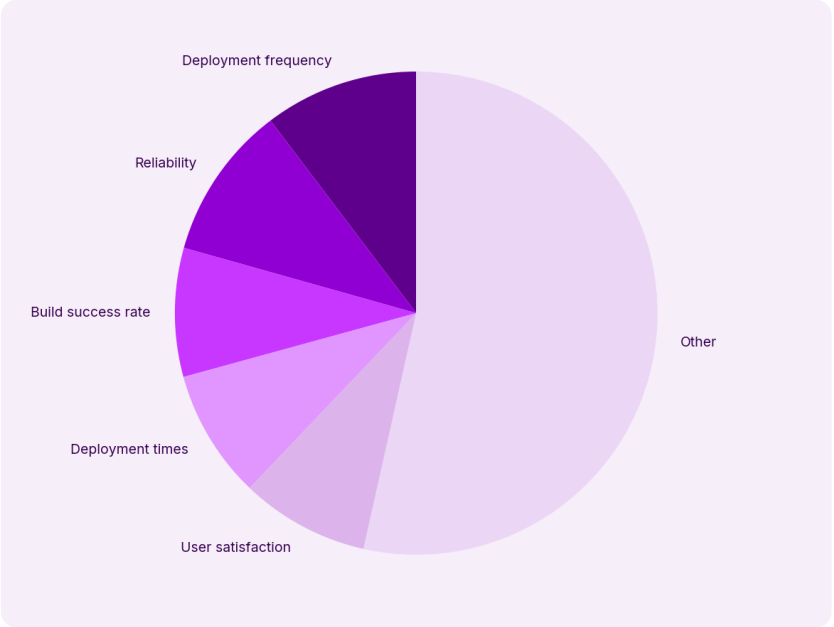
Organizations primarily adopt Platform Engineering to improve efficiency, standardize processes, and increase developer productivity, rather than reducing developer cognitive load.

2. Common platform features



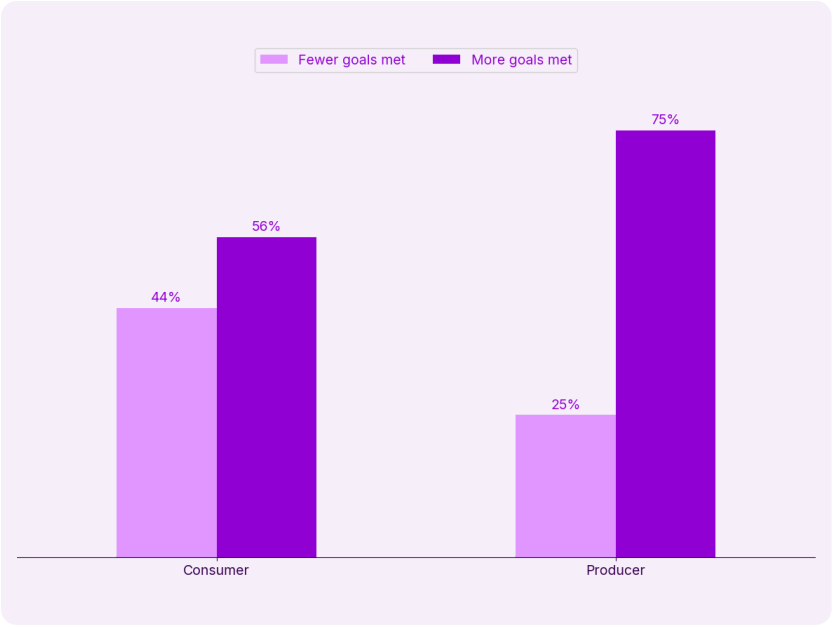
High performing platforms are likely to include deployment pipelines, infrastructure automation, and monitoring and observability.

3. Success metrics



Nearly 25% of organizations rely on subjective assessments rather than formal metrics to evaluate platform success. Those using metrics employ 1 to 8 different measures.

4. Adoption strategy effectiveness



63% of platforms are mandated rather than optional. Platform producers report higher success rates (75%) than consumers (56%), revealing a persona perception gap.

Introduction

What is Platform Engineering?

Platform Engineering involves designing, building, and maintaining capabilities that enable developers to create and run applications efficiently. It focuses on creating a stable, scalable, secure platform supporting the entire software development lifecycle.

Platform Engineering aims to create an environment where developers can focus more on writing features and less on managing infrastructure. This accelerates innovation and improves overall software quality.

The community believes the key attributes of successful Platform Engineering efforts are:

- Treating platforms as products
- Developers are the customer
- Optional platform adoption (the internal developer platform should win market share)
- Making the platform self-service, with support
- Solving real developer problems, not inventing new tech stacks

The Platform Engineering Pulse report

Platform Engineering has emerged as an essential discipline for modern software organizations, yet insights into real-world implementation practices have been limited. There was a need for evidence of what works in practice, which features deliver value, and how to measure success.

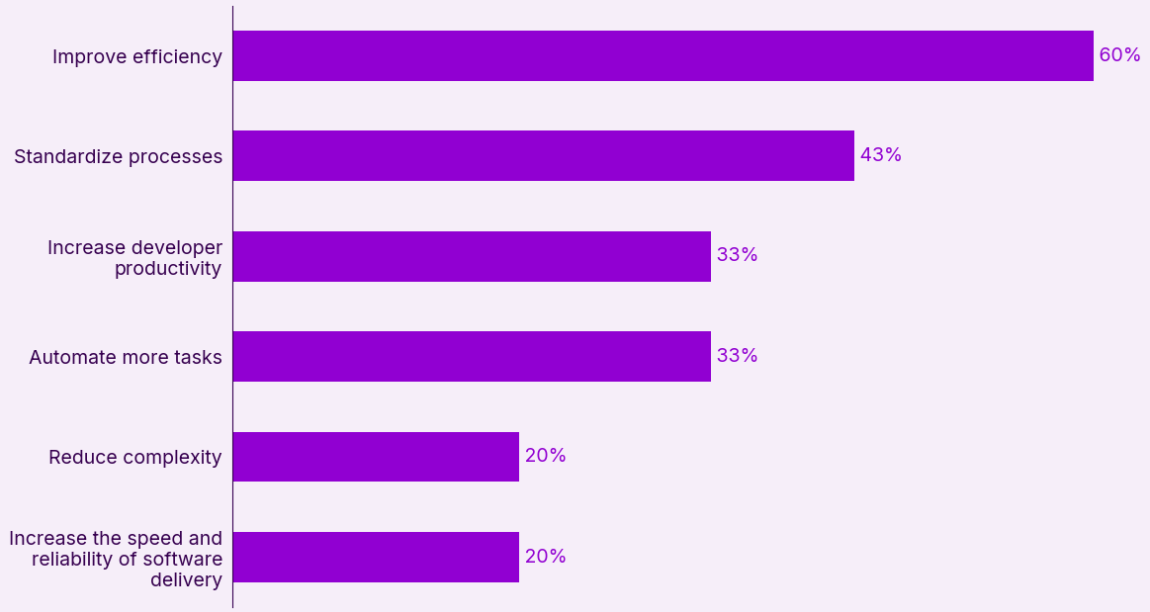
We created this report to fill that knowledge gap by establishing an initial benchmark for Platform Engineering initiatives.

Through direct engagement with technical professionals actively building and operating platform initiatives, we validated theoretical frameworks against real-world experience. We gained authentic insights into the strategies and success factors that define practical Platform Engineering.

This report enables organizations to assess their platform maturity, learn from industry practices, and make informed strategic decisions based on proven approaches rather than theoretical assumptions.

Why organizations adopt Platform Engineering

We asked respondents to provide their top 3 reasons for adopting Platform Engineering in their organization. Our expectation was that lowering developers' cognitive load and reducing burnout would feature strongly, as these are the reasons commonly cited in Platform Engineering literature, with the **CNCF White paper**, stating that the "essential goal of a platform is to reduce cognitive load on product teams"[1].



Adoption reasons by % of organizations

Contrary to our expectations, the most common reasons were for efficiency improvements, standardizing processes, productivity, and automation.

Also important, but not among the most common reasons, were increasing security, process consolidation, and standardizing tools.

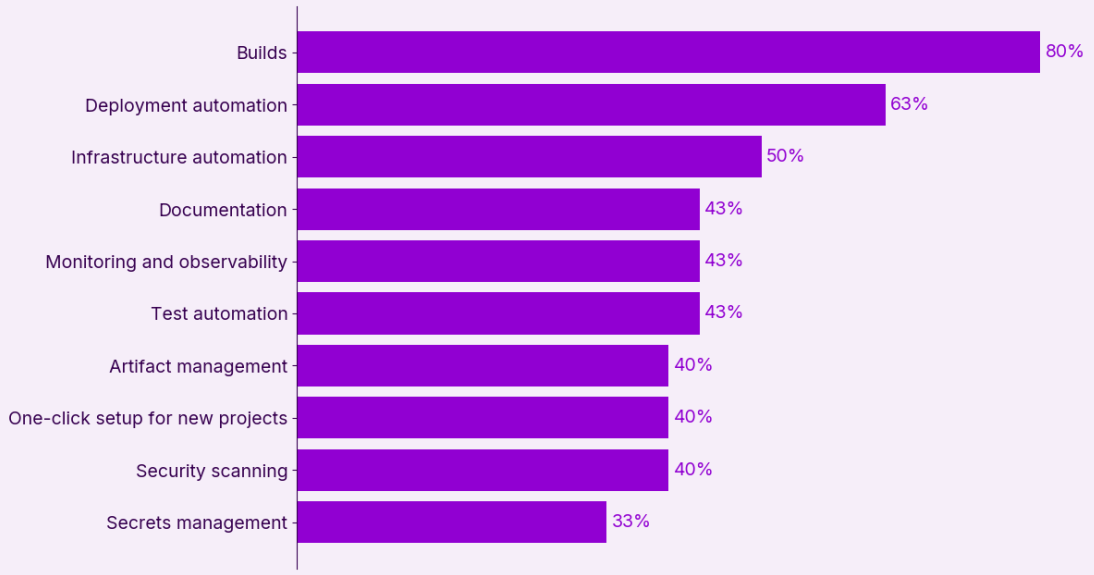
Developer-focused reasons, such as empowering developer decision-making with guardrails, improving access to institutional knowledge, and reducing developer overload and burnout, were rarely among the top 3 reasons for adoption.

While platform initiatives like efficiency and automation may indirectly lower cognitive load and reduce burnout, the results suggest the developer experience isn't the driving force behind platform implementation.

Platform features

Top 10 features of IDPs

We asked which features were part of the internal developer platform (IDP). Respondents were able to select all answers that applied from a list we created during interviews and questionnaires. Respondents were able to provide additional answers if they couldn't find something in the list.



Platform features by % of organizations

The top 10 features of internal developer platforms are listed below.

1. Builds

Building an application from source code involves compiling code, linking libraries, and bundling resources so the software can be run on a target platform. While this may not be the most challenging task for software teams, build processes can become complex and require troubleshooting that takes time away from feature development.

When a team rarely changes its build process, it tends to be less familiar with the tools. It may not be aware of features that could improve build performance, such as dependency caching or parallelization.

2. Deployment automation

Even at the easier end of the complexity scale, deployments are risky and often stressful. Copying the artifact, updating the configuration, migrating the database, and performing related tasks present many opportunities for mistakes or surprises.

When teams have more complex deployments or need to deploy at scale, the risk, impact, and stress increases.

3. Infrastructure automation

Traditional ClickOps infrastructure is hand-crafted and often drifts from the intended configuration over time. Environments may be set up differently, which means problems surface only in one environment and not another. Equally, two servers in the same environment with the same intended purpose may be configured differently, causing problems to troubleshoot.

Infrastructure automation solves these problems and makes it easier to create new environments, spin up and tear down ephemeral (temporary) environments, and recover from major faults.

4. Test automation

To shorten feedback loops, all types of testing should be performed continuously and ideally represented by fast, reliable test automation suites. You should cover functional, security, and performance tests within your deployment pipeline.

Consider how test data is managed as part of your test automation strategy. The ability to set up data in a known state will help you make tests less flaky. Test automation lets developers know early if their change causes a fault, reduces team burnout, and increases software stability.

5. Monitoring and observability

While test automation covers a suite of expected scenarios, monitoring and observability help you expand your view to the entirety of your real-world software use. Monitoring implementations tend to start with resource usage metrics, but mature into measuring software from the customer and business perspective.

The ability to see information-rich logs can help you understand how faults occur so you can design a more robust system.

6. Documentation

To provide documentation as a service to teams, you may supply a platform for storing and finding documentation or use automation to extract documentation from APIs to create a service directory for your organization.

For documentation to be successful, it needs to be clear, owned, updated, and accessible[1].

7. Security scanning

While everyone should take responsibility for software security, having automated scanning available within a deployment pipeline can help ensure security isn't forgotten or delayed. Automated scanning can be a source of fast feedback for developers.

Automated security scanning should be supplemented with security reviews and close collaboration with information security teams.

8. One-click setup for new projects

When you set up a new project, several boilerplate tasks are needed to configure a source code repository, set up a project template, configure deployment pipelines, and set up associated tools. Teams often have some standards, but manual setup means projects drift unintentionally from the target setup.

One-click automation helps teams set up a walking skeleton with sample test projects, builds, and deployment automation. This ensures a consistent baseline and speeds up the time to start writing meaningful code.

9. Artifact management

Your Continuous Integration (CI) process creates a validated build artifact that should be the canonical representation of the software version. An artifact repository ensures only one artifact exists for each version and allows tools to retrieve that version when needed.

10. Secrets management

Most software systems must connect securely to data stores, APIs, and other services. The ability to store secrets in a single location prevents the ripple effect when a secret is rotated. Instead of updating many tools with a new API key, you can manage the change centrally with a secret store.

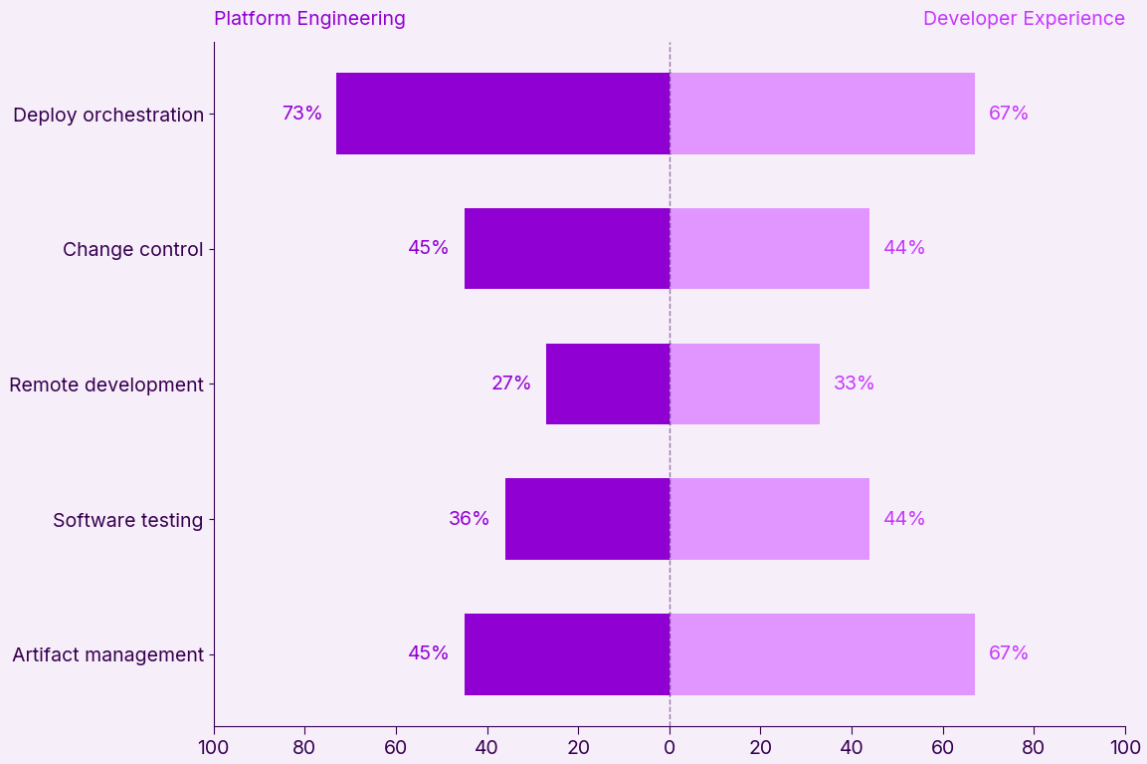
When you deploy an application, you usually have to apply the correct secrets based on the environment or other characteristics of the deployment target.

Platform versus DevEx features

DX provides a platform that helps organizations improve developer productivity and impact through metrics and insights.

They recently looked in depth at Platform Engineering and developer experience (DevEx) across **14 organizations**. They found that "Platform teams tend to have a wider scope of responsibilities than DevEx teams"[2], but we can also use the data to see whether a feature is more likely to be featured by Platform Engineering teams or DevEx teams.

The figure below shows the most popular features and the degree to which they are used by both kinds of team.



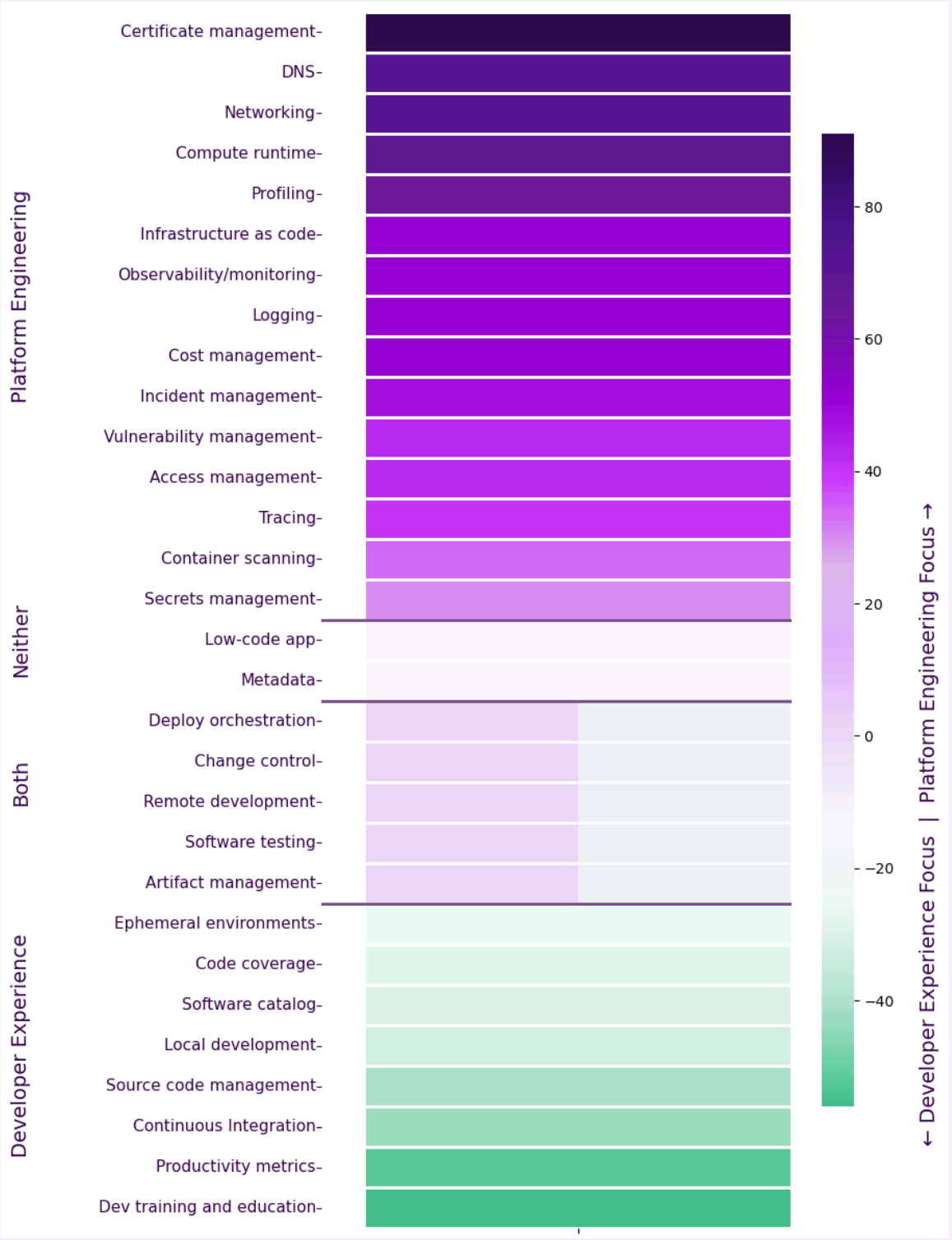
Popular features for Platform Engineering and DevEx teams

While most features appeared in both Platform Engineering and DevEx to some extent, some features were exclusive to a single discipline.

Platform only	DevEx only
Certificate management	Developer training and education
DNS	
Networking	

> While many people consider a feature to be either a platform concern or a DevEx concern, the reality is that both types of teams look after cross-cutting capabilities. This shows that terminology, roles, and responsibilities in this area are still evolving.
>
> *Ships Mahindra*

All other features appear along a scale from being mostly a platform feature to mostly a developer feature. In the figure below, the heatmap displays features the are more platform focussed (purple) to more developer focussed (green), with "Low-code app" and "Metadata" (white) found to have limited presence in either discipline.



Association between features and disciplines

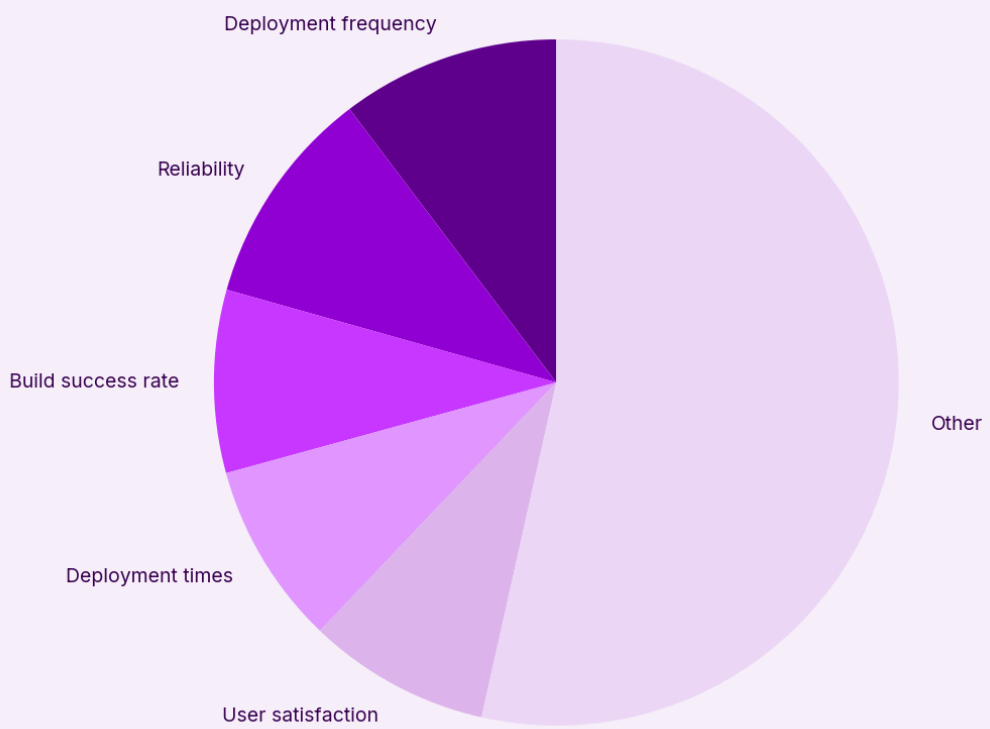
Platform teams aim to make product development more efficient. Therefore, DevEx and Platform Engineering should be kept together or at least very closely aligned. Otherwise, they will each have areas where they can't directly impact outcomes, as boundary negotiation is required to achieve their goals.

When a team is formed around a specific focus area, others outside the team often spend less time thinking about and solving issues in that area. For example, a DevEx team may result in lower developer satisfaction if line managers believe they no longer need to participate in improving it.

How success is measured

The task of measuring success for Platform Engineering has varied responses. Organizations differ in what they measure as well as the number of measures they apply.

Almost a quarter of organizations (23%) don't use metrics and rely instead on intuitive or subjective assessments. Organizations using metrics typically collect between 1 and 8 specific measures of success.



Top 5 success metrics

To measure success, we rate performance for each organization against their specific success criteria. In theory, two organizations may have identical platforms but have divergent views of their success because their goals differ.

We found that organizations focused on 3 keys areas for measurement: software delivery, operational performance, and user experience. The variety of metrics likely reflects the many contexts where platforms can be used.

The most popular measures of success, with close to 10% of organizations using each of them, are:

1. Deployment frequency
2. Reliability
3. Build success rate
4. Deployment times
5. User satisfaction

We can map the various success criteria to **DORA metrics**, which are used to assess software delivery performance[3], and **MONK metrics**, which are specifically designed for the measurement of Platform Engineering initiatives[4].

DORA metrics

The **DORA metrics**[3] are commonly used to measure success in Platform Engineering, though deployment frequency is used more than the other three metrics.

DORA metrics	Use
Change lead time	3%
Recovery time	10%
Change failure rate	13%
Deployment frequency	20%

While these metrics are widely recognized as helpful for assessing software delivery performance, applying these to Platform Engineering success needs a balanced perspective. When evaluating platform performance using these metrics alone, a positive impact may be found. However, on closer inspection, according to the **DORA 2024 Accelerate the State of DevOps report**, a platform's development may temporarily affect a developer's throughput and stability, suggesting that a broader approach is necessary for an accurate measurement of success[5].

Atlassian has found that applying the DORA metrics with a balanced team, full-time owner, clear focus on values and metrics, and managed dependencies allows initiatives to evolve into high-performing, stream-aligned teams[6].

A platform team might measure its own DORA metrics to inform its continuous improvement process, but not use them as success metrics for the platform. If you created a platform to enable developers to increase their software delivery performance, you might include the DORA metrics for the platform's consumers.

MONK metrics

The **MONK metrics** were created specifically to measure Platform Engineering[4]. While almost all organizations were able to tell us their platform's market share, only organizations who use this in their success criteria are included here.

MONK metrics	Use
Market share	3.4%
Onboarding time	3.4%
Net Promoter Score (NPS)	5.2%
Key customer metrics	10.3%

The first 3 measures are concrete and benchmarkable. They are broadly applicable to Platform Engineering teams. The last category, key customer metrics, is abstract. This should reflect the business and technical motivation for introducing Platform Engineering, such as those found in the adoption reasons section of this report.

Market share

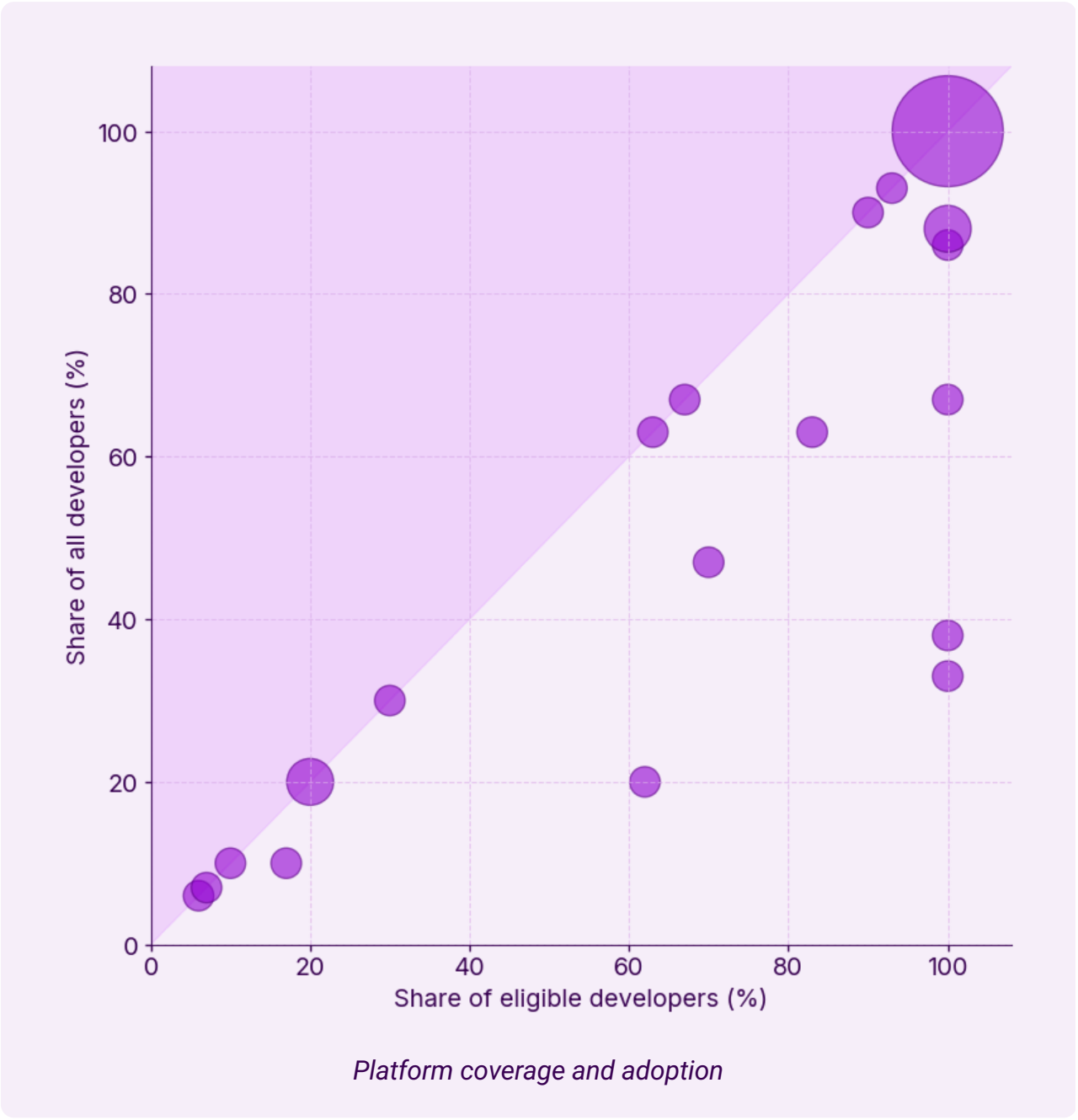
Market share refers to the number of developers adopting the internal developer platform. There are two measures of market share:

- **Eligible share**: The proportion of developers using technology stacks supported by the platform.
- **Market share**: The proportion of eligible developers choosing to use the platform.

The eligible share may be a consideration when choosing the technologies you want to support with a golden path. When faced with multiple options, the number of developers that will become eligible to use the platform may be a decision factor.

In the figure below, bubbles further right have a higher eligible share; indicating that their platform is relevant to more developers in their organizations. The bubbles closer to the top have a higher market share; indicating that more developers are using the platform.

The size of bubbles indicates the number of organizations represented at those coordinates.



Platform coverage and adoption

There are two strategies for increasing your platform's market share. You can decide between these strategies based on whether your adoption follows the diagonal line in the figure or falls below it.

Strategy 1: If your platform is below the diagonal, it means some developers could use it but are choosing not to. Your efforts should initially focus on finding out their reasons for staying off-platform.

Your approach will likely involve demonstrating the benefits other developers have gained using the platform, and improving platform onboarding to reduce the burden on teams who want to adopt it.

Strategy 2: When your platform hits the ceiling of the diagonal line, your strategy must shift to increasing the number of developers using technology the platform supports.

While this sometimes means adding support for additional technology choices, you should avoid adding support for non-strategic tech stacks. For example, if your organization wants to move away from particular languages, frameworks, or hosting targets, you shouldn't form golden paths around their use. Instead, you should find ways to smooth teams' paths onto a supported tech stack.

Don't add support for technology you don't want to standardize around. This will distract from enhancing support for the strategically best technologies for your organization. By supporting an obsolete technology, you may inadvertently encourage its adoption.

Though many teams aren't using market share as a success factor, platform teams will still find that the market share metric will help direct their efforts.

Finding yourself on the top-right of this chart doesn't mean you've finished your organization's platform initiative. Instead, it indicates the danger of failing to maintain and innovate the platform properly. If your platform stagnates, it will impact 100% of developers in your organization.

When we assessed market share by performance score, we found that high-performing platforms do not necessarily guarantee adoption. Platform initiatives meeting many or all goals can still experience low adoption rates, suggesting that success depends on factors beyond platform features (e.g., documentation, awareness, and onboarding support).

Onboarding time

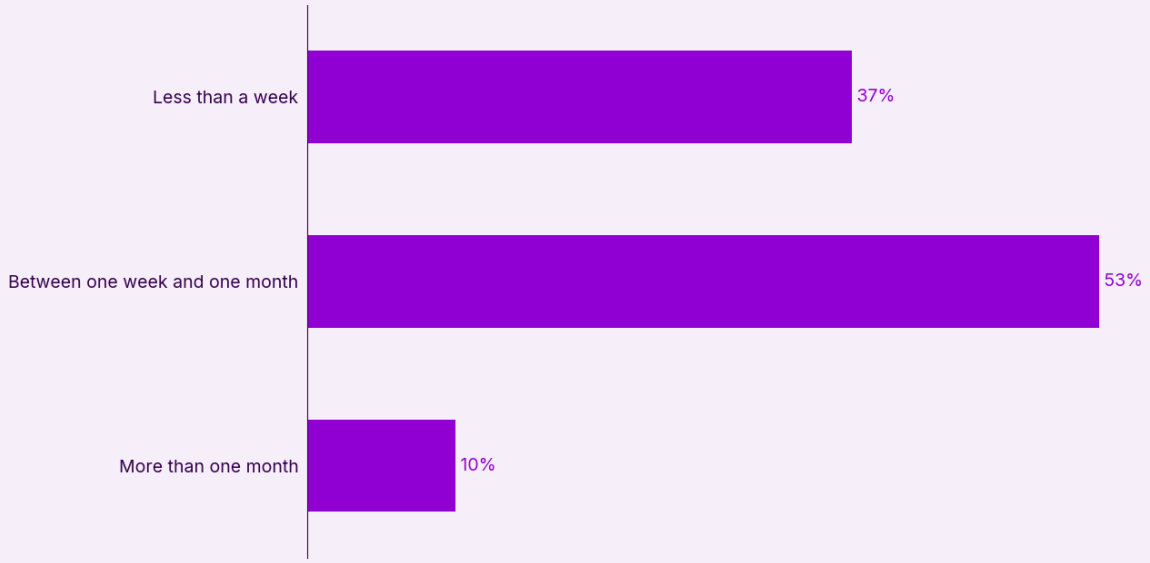
Onboarding times hide in plain sight. We often think of new hires when it comes to onboarding time, but a similar set of onboarding tasks are needed when you move between teams, pick up an application you haven't worked on recently, or upgrade your equipment.

To collect developer onboarding time, you should start the clock when a developer starts working on a new codebase and stop the clock when their first commit reaches production.

The goal of the onboarding time metric isn't just to speed up that first change. It reflects the quality of developer documentation and the level of automation in your deployment pipeline.

Platform teams may also consider the time it takes a team to move from their current tools onto the internal developer platform, as this will impact adoption rates.

When developers join the organization or work on a new codebase, quality documentation and automated deployment pipelines make it more likely they'll get their first change to production quickly. Most organizations have onboarding times between one week and one month (53%), but over a third manage to onboard developers in less than a week.



Onboarding times by % of organizations

Net promoter score (NPS)

You can use net promoter score (NPS) to gauge customer satisfaction, loyalty, and willingness to recommend a product or service, which can be influenced by factors beyond just the software's quality, like the customer's overall experience with the company[1,7]. This is relevant to Platform Engineering as platform teams typically treat developers as customers.

Surprisingly, organizations aren't using NPS, or a similar measure, to track satisfaction with the internal developer platform over time. Those who did collect NPS generally had low positive scores. As there is a cluster of NPS scores between 1 and 10, which make it likely respondents reported a 10-point rating, rather than an NPS score. This will need further investigation. While NPS scores provide a satisfaction signal, collecting them also offers an opportunity for feedback.

To assist teams who want to use this metric, revisiting how NPS is calculated may be helpful. NPS is calculated using a single 1 to 10-rated survey. These responses are collected and classified into three categories:

- **Detractors (0-6):** These unhappy customers may damage the company's reputation through negative word-of-mouth.
- **Passives (7-8):** These are satisfied but not enthusiastic, and could switch to a competitor.
- **Promoters (9-10):** These are loyal enthusiasts likely to recommend the company or its offerings.

You ignore the passives and calculate the NPS by subtracting the percentage of detractors from the percentage of promoters. This results in a score between -100 and +100. For example, if 10% of respondents are Detractors, 15% are Passives, and 75% are Promoters, your NPS score would be 75-10 = 65.

The table below shows the typical label for different scores.

Classification	Range
Bad	Less than 0
Okay	0-19
Good	20-49
Excellent	50-69
World-class	70+

NPS provides valuable insights into developer satisfaction[7], and can be used with other metrics, like the Customer Satisfaction Score (CSAT), for a more comprehensive understanding. CSAT directly indicates customer satisfaction with specific products or services[8].

CSAT surveys typically employ a Likert-style question, with a 5-point scale for overall satisfaction:

1. Very unsatisfied
2. Unsatisfied
3. Neutral
4. Satisfied
5. Very Satisfied

The results are then averaged, or the percentages are calculated to determine the CSAT score. For example, the number of satisfied customers (those who responded either 4 or 5)/the number of survey responses x 100 = the percentage of satisfied customers.

NPS and CSAT differ in focus. CSAT measures satisfaction with specific interactions, while NPS gauges a customer's long-term relationship with an organization or product. Combining NPS and CSAT offers a more comprehensive view of customer sentiment.

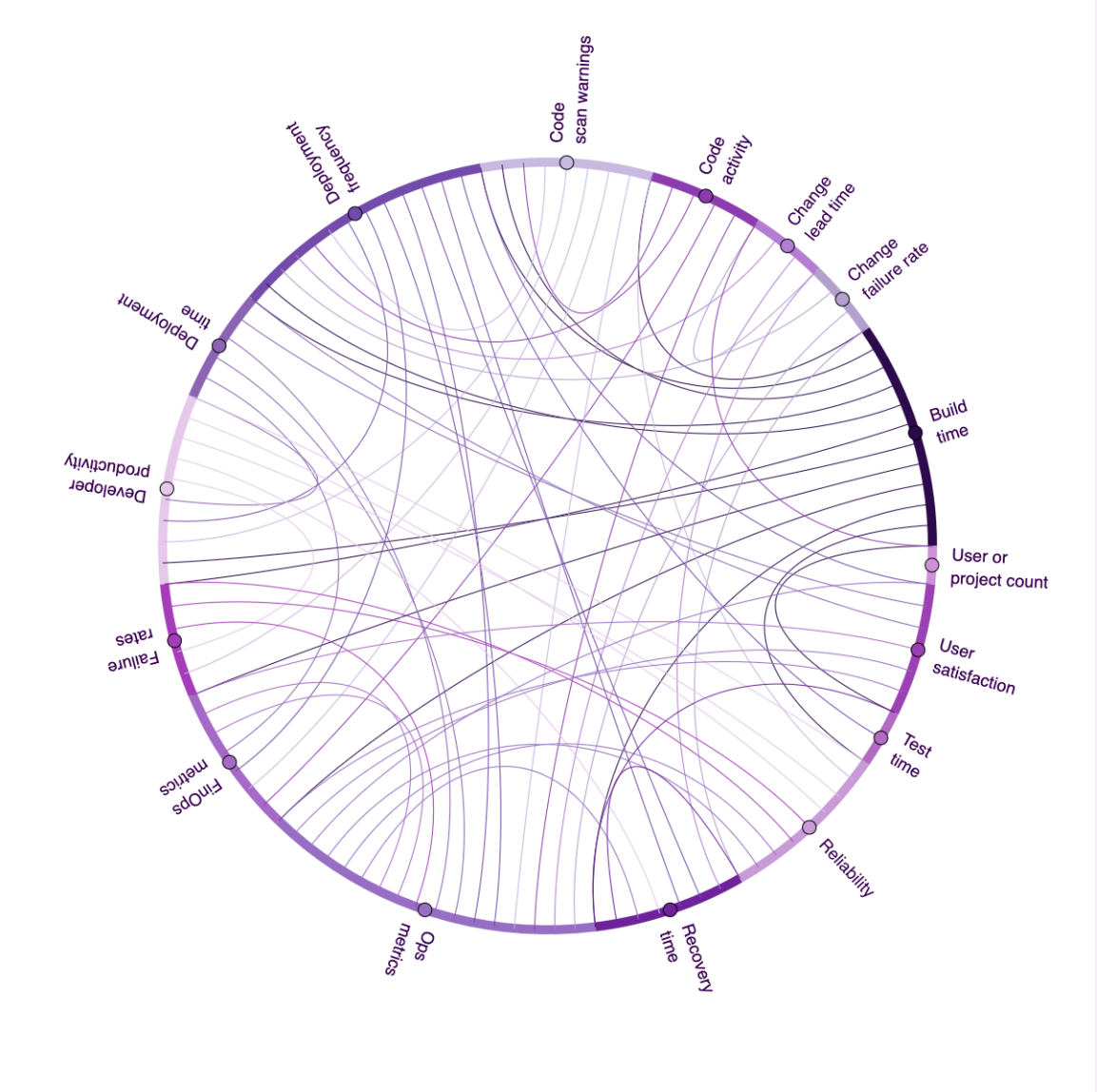
Key customer-centric metrics

The key customer-centric metrics should emerge from conversations with the developers and sponsors of the platform. A platform that captures and these goals and tracks progress against them will be more successful.

Collecting metrics helps focus the platform team, protects long-term funding, and provides a useful internal marketing tool to help increase adoption.

The most common customer-centric metrics are:

1. Ops metrics
2. Deployment frequency
3. Build time
4. Developer productivity
5. Reliability
6. Code scan warnings



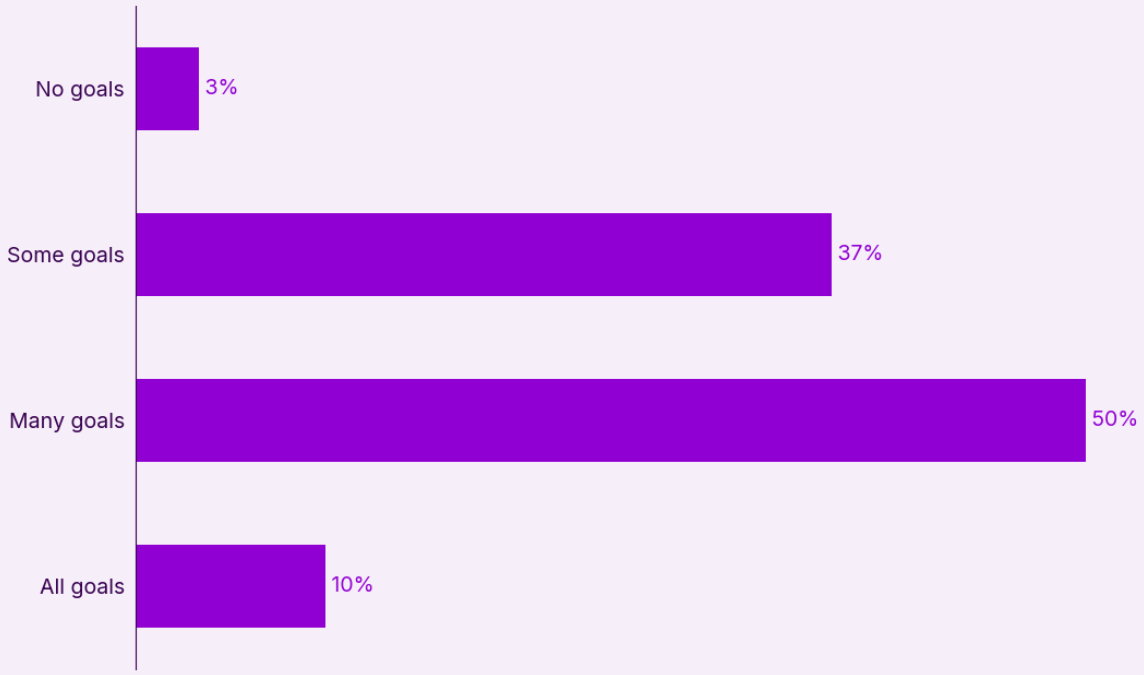
The most common customer-centric measures

You may also be inspired by these less common customer-centric measures, which have been arranged alphabetically:

- Deployment time
- Code activity
- Change lead time
- Change failure rate
- User or project count
- User satisfaction
- Test time
- Recovery time
- FinOps metrics
- Failure rates

What makes platforms successful

Overall success rates

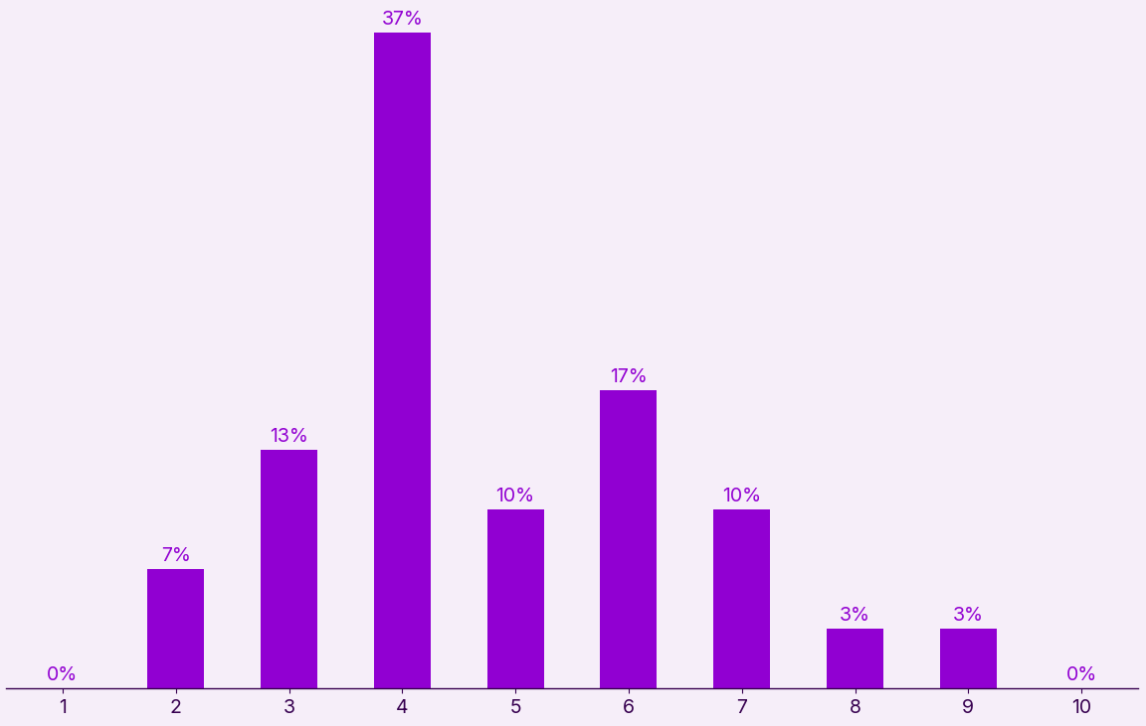


Platform success by % of organizations

Each platform's success is measured against its unique success criteria, rather than against a single definition of success. This is important as there was great variety in the objectives of platforms across different organizations. The success criteria measures the extent to which the platform meets its own goals.

The outlook is largely positive, with 60% of platforms meeting many or all goals, though there's room for improvement.

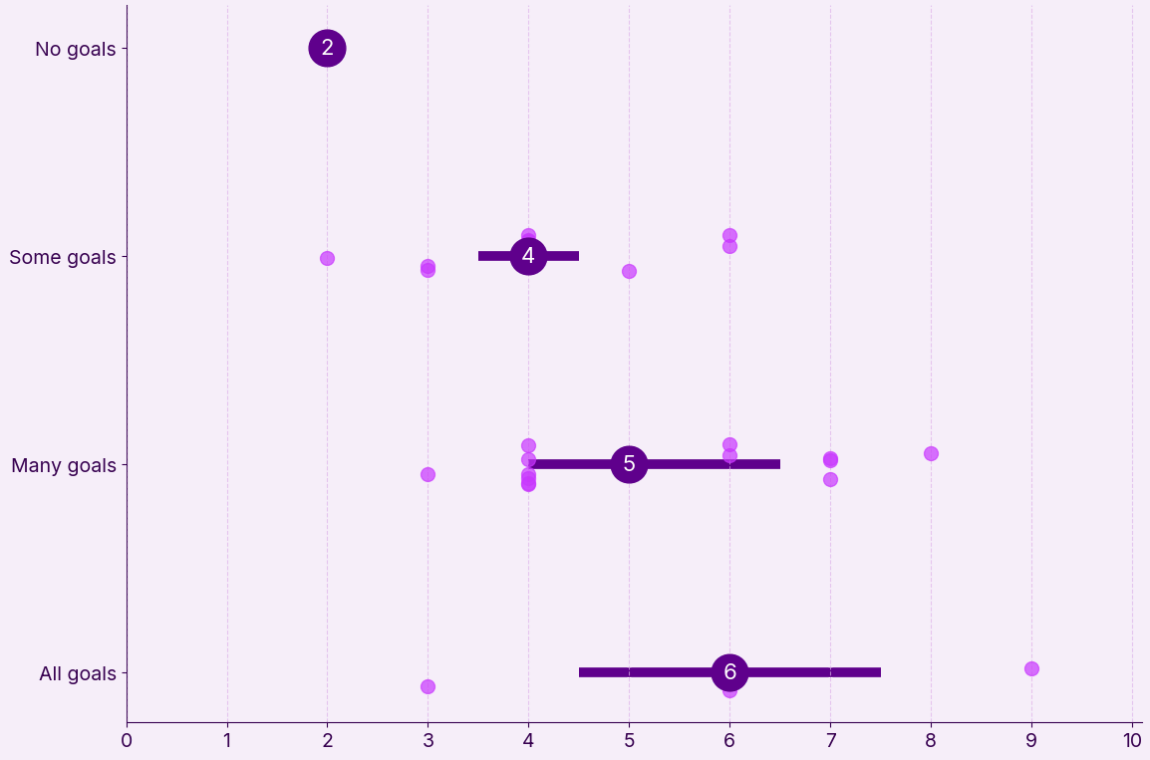
Platform breadth



Feature count by % of organizations

Most organizations have built platforms with 4 features (37%).

Platforms with fewer than 3 features were less successful, while those with a broader feature set are more likely to meet their goals.



Number of features by goal attainment

While the increased number of features correlates with platform success, an interesting point is that platforms can also be very successful with as few as 3 features. This highlights the importance of a product mindset.

Goal attainment	Average features
None	2.0
Some	4.1
Many	5.2
All	6.0

Instead of building a large platform with every possible feature, a successful team focuses on understanding the tasks developers are trying to complete and what gets in their way. By focusing on solving the problems experienced in their organization, platform teams can get a high impact with less extensive feature or service sets. This user-centric approach often results in a smaller, more successful platform than a team that operates with a "we know best" anti-pattern.

Additionally, there are patterns for successful feature combinations, which you'll see next in this report.

While the most common platform features overall are:

- Builds
- Deployment automation
- Infrastructure automation

There are differences in feature selection between high-performing teams (meeting many or all goals) and low-performing teams (meeting some or no goals).

Features used by high-performers

The most common platform features for high performers are:

- Builds
- Deployment automation
- Test automation
- Monitoring and observability
- Artifact management
- Infrastructure automation

Low-performers are less likely to be doing:

- Test automation
- Artifact management
- Monitoring and observability

High-performers are:

- 3.6x more likely to do test automation
- 3.3x more likely to do artifact management
- 2.2x more likely to do monitoring and observability
- 2.0x more likely to do one-click setup for new projects

Builds need complementary practices

Build automation is a definite starting point for high performers. However, automating builds without complementary practices results in platforms not achieving their goals. Alongside builds, you need to smooth the path for:

- Test automation
- Artifact management
- Deployment automation
- Monitoring and observability

Features to focus on

Feature	Chance of being a higher performer
Builds	5.0x
Deployment automation	1.6x
Test automation	1.6x
Monitoring and observability	1.3x
Infrastructure automation	1.3x
Artifact management	1.3x

For deployment pipelines, builds are the table stakes. Automated deployment and test automation are the highest priority items after builds.

Platform feature evolution

There is a clear three-stage evolution of internal developer platforms, from a basic deployment pipeline through to a comprehensive DevOps offering.

1. **Deployment pipeline**
2. **Secure pipeline**
3. **DevOps pipeline**

Stage 1: Deployment Pipeline

Automates the deployment pipeline and ensures production is monitored.

- Builds
- Test automation
- Artifact management
- Deployment automation
- Monitoring and observability

The Deployment Pipeline focuses on getting code to production, including code quality features like automated builds and test automation. Artifact management controls the storage and versioning of builds, while deployment automation ensures repeatable and reliable processes. Monitoring and observability are used to track the live product performance.

Stage 2: Secure Pipeline

Brings additional security measures to the deployment pipeline.

- Builds
- **+ Security scanning**
- Test automation
- Artifact management
- **+ Secrets management**
- Deployment automation
- Monitoring and observability

The Secure Pipeline integrates security directly into the deployment pipeline by adding security scanning, which automatically checks for code weaknesses and dependencies, and secrets management, which consolidates the storage and rotation of secrets. This is significant as it ensures security measures are addressed earlier, reducing the risk of incidents in production.

Stage 3: DevOps Pipeline

Adds missing DevOps capabilities to the deployment pipeline.

- **+ Documentation**
- **+ One-click setup for new projects**
- Builds
- Security scanning
- Test automation
- Artifact management
- Secrets management
- **+ Infrastructure automation**
- Deployment automation
- Monitoring and observability

The DevOps pipeline enables developers to have full-lifecycle capabilities. Maintaining high-quality internal documentation is pivotal in increasing product performance, as documentation acts as a feedback loop affecting teams and products.

The **DORA 2024 Accelerate the State of DevOps report** encourages *high-quality* documentation rather than *comprehensive* documentation. A smaller set of documentation that can be kept up-to-date and provides useful answers is better than a large volume of documentation[5]. Healthy documentation culture involves defining clear ownership and consistently updating content to manage priority changes.

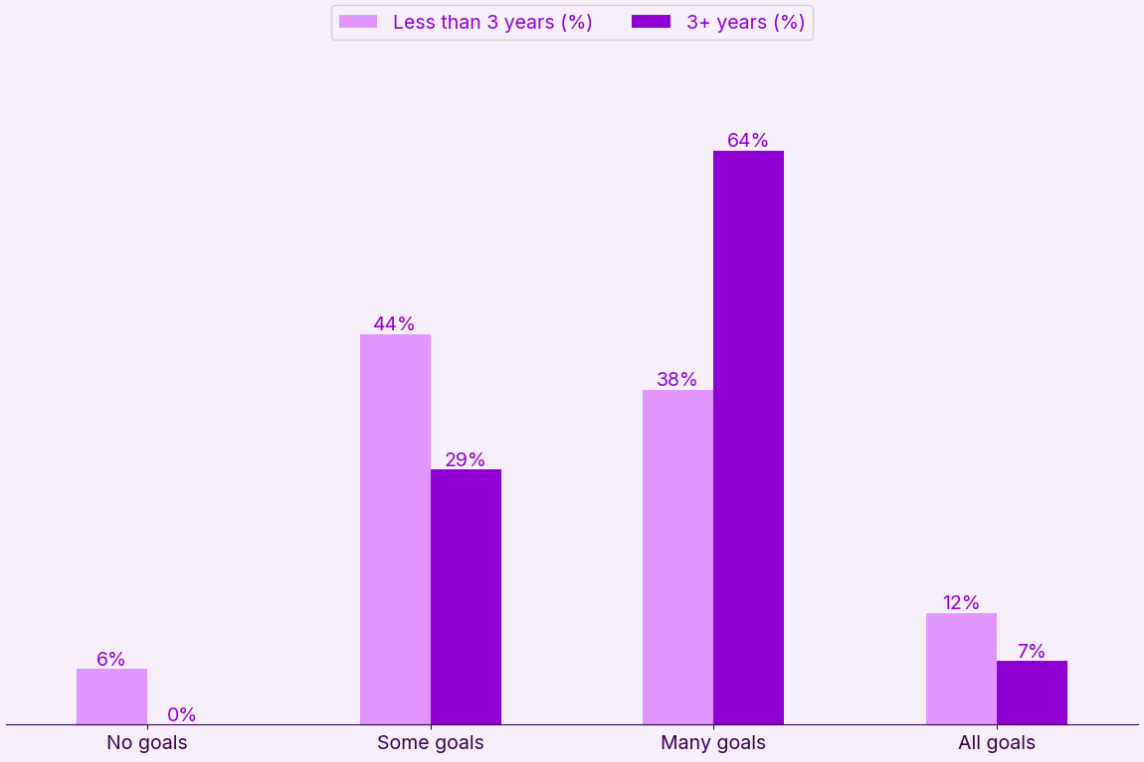
Using one-click setup and infrastructure automation streamlines the developer workflow. One-click setup automates the entire "path to production" for a project, handling the configuration of version control, boilerplate project code, and deployment processes in a single action or "click". This is underpinned by infrastructure automation, which stores configurations in version control instead of relying on manual "ClickOps." These practices verify that all environments are identical, simplifying scaling, testing, and creating ephemeral (temporary) environments.

High-performing teams consistently mature their pipelines, with top-performers incorporating the most advanced DevOps capabilities.

Success by platform age

Platform Engineering initiatives follow a predictable curve of success, a pattern identified in the **DORA 2024 Accelerate the State of DevOps report**.

The research found that teams often see an initial surge of success, from early, straightforward wins, followed by a temporary dip as they encounter more complex challenges, and then a recovery period as the platform matures (the famous "J-curve" pattern)[5]. This happens when platform teams slip into old ticket-based operations habits that slow developers down.

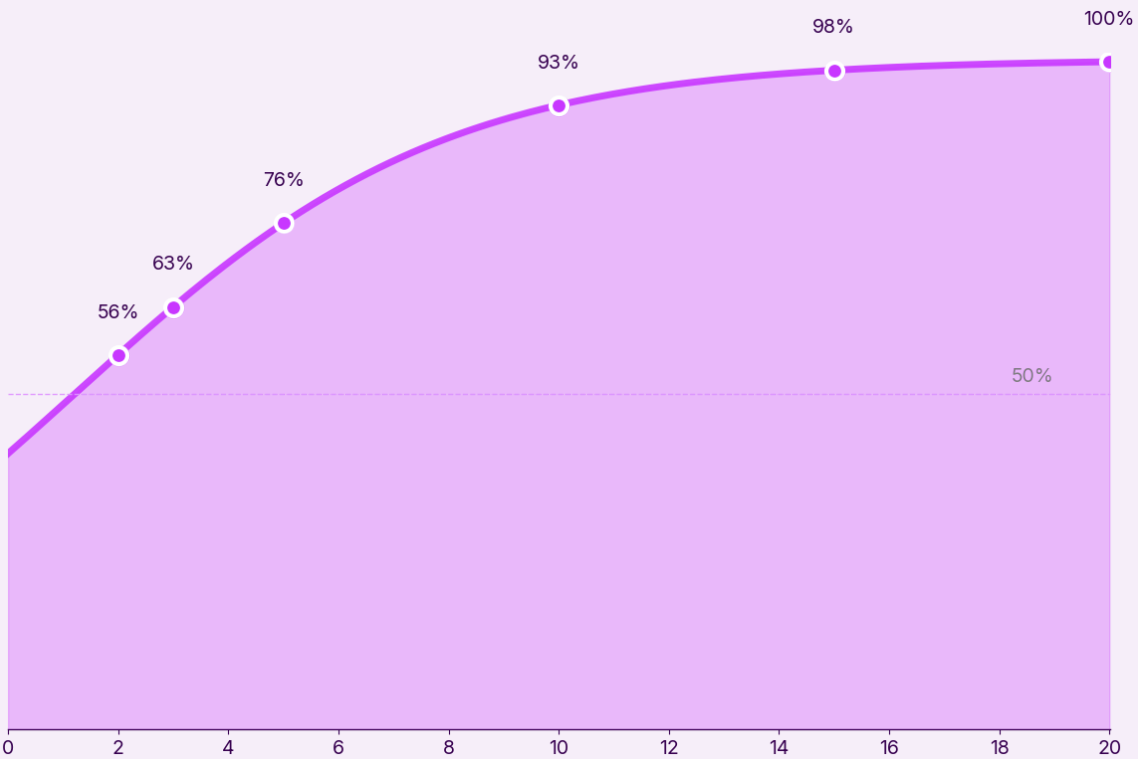


Comparison of platforms less than 3 years old, and 3 or more years old

Similarly, we found a comparable trend in our research: platform teams are typically more successful after the three-year mark, with 71% of platforms with 3+ years of experience meeting many or all of their goals. This highlights that while some early wins are common, the most significant improvements are a product of long-term investment and maturity.

The full benefits of an internal developer platform become most evident over time, with sustained goals met as the platform matures.

The figure below shows a strong positive correlation between platform maturity and success, with platforms achieving more of their goals as they become more experienced. This is consistent with the **Puppet 2023 State of DevOps report**, which found that Platform Engineering initiatives that started more than 3 years ago has a success rate in speeding up software teams of 53%, compared to 35% of platforms less than three years old[9].

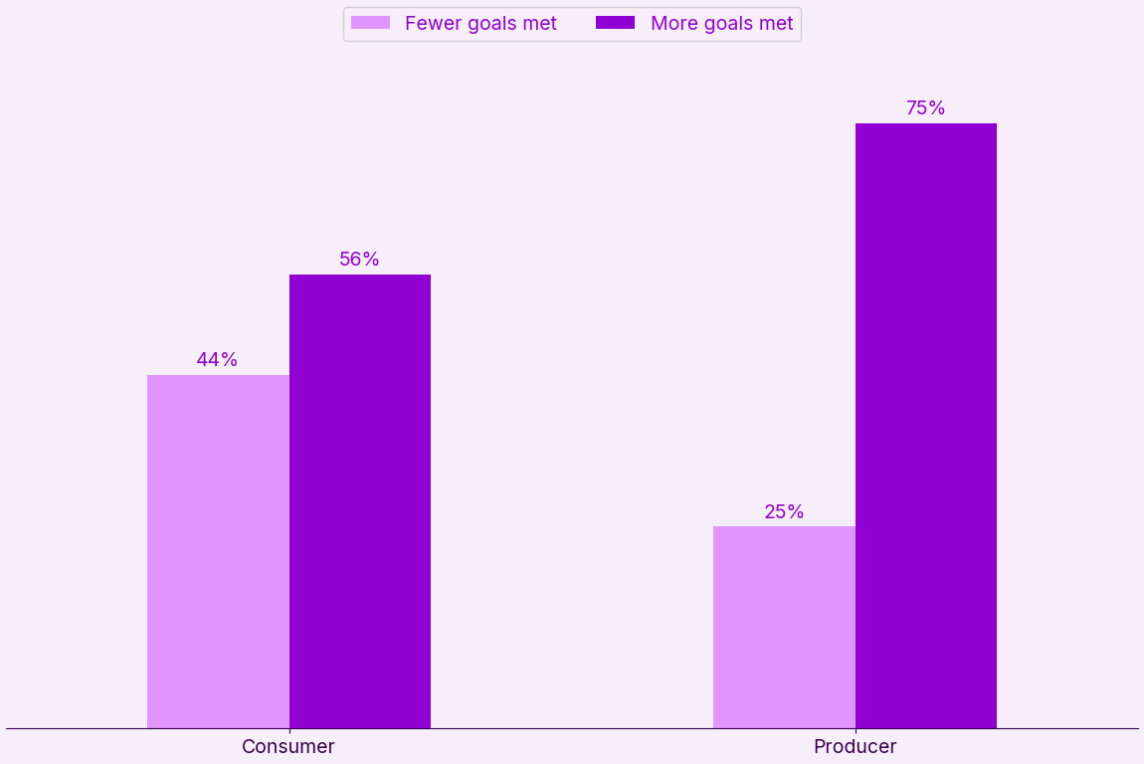


Predictive platform success over years of experience

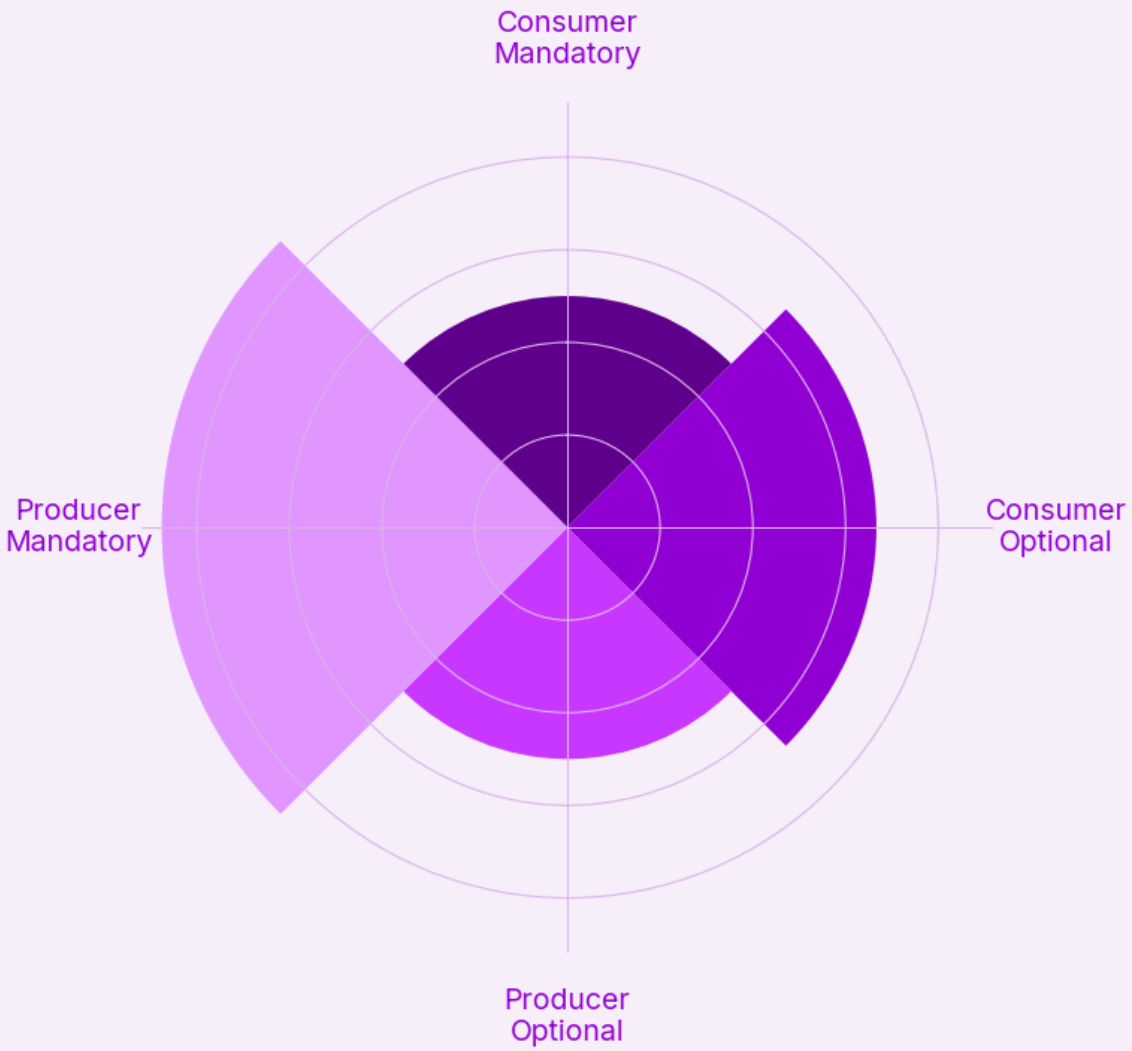
Success by perspective

We collected perspectives from producers (platform engineers), consumers (developers and testers), and sponsors (team leads, executives, department heads, and founders). Producer and consumer perspectives are presented below.

- Fewer goals met (none or some): low-performing
- More goals met (many or all): high-performing



Comparison of producer and consumer opinions of success



High performance based on perspective and adoption strategy

We found that producers tend to be more positive about goal attainment than consumers. 75% of producers reported platforms are meeting many or all goals, compared to 56% of consumers. Producers think mandatory platforms are more successful, while consumers are more likely to rate an optional platform as successful.

- 88% of producers working on mandatory platforms rate that it meets many or all goals compared to 50% of consumers.
- Producers working on optional platforms are split down the middle with 50% telling us it meets many or all goals compared to 67% of consumers.

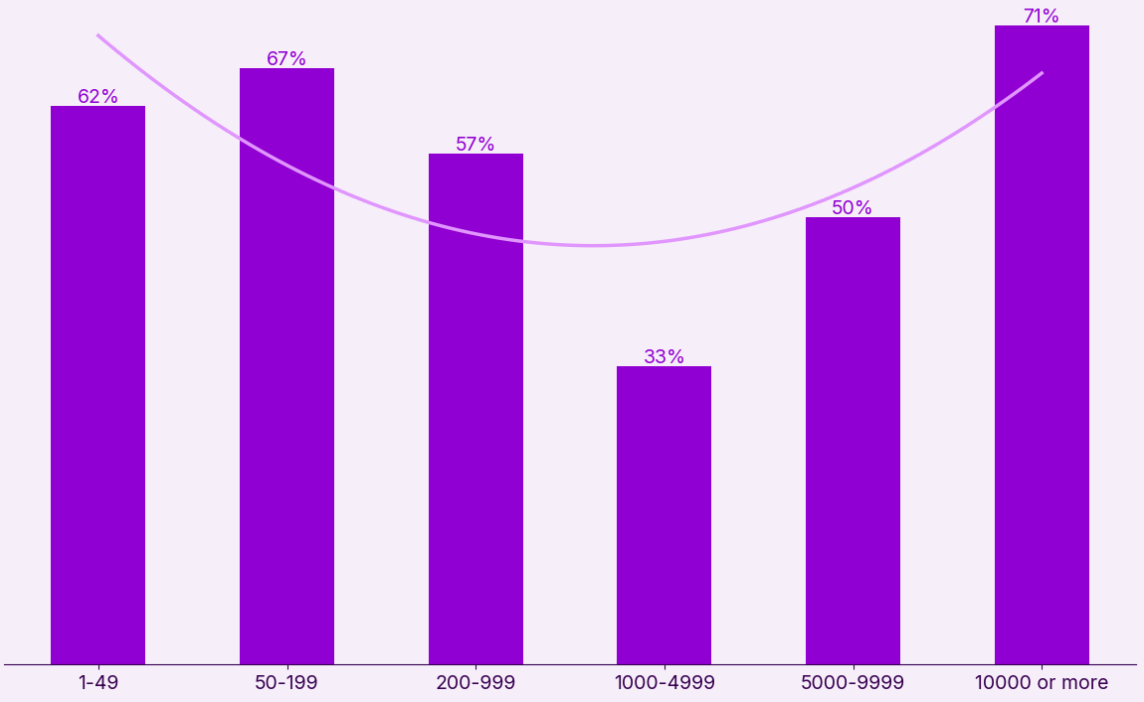
It is interesting to note the perception gap between producers and consumers. This could be due to "producer bias" where creators overestimate the value of the product. This result highlights why it's crucial to move beyond internal assumptions in order to truly build a successful product.

Recommendation: Teams building an internal developer platform need to connect to the platform's users to assess its success. The perception gap shows that platform teams may view their work more positively than the developers using it. The MONK metrics can help take a more user-centric approach to Platform Engineering.

Sponsors perspectives

Sponsors are cautious about the project's overall success. They consistently report that while progress has been made, only some goals have been achieved.

Success by organization size



High performance based on % of organization size

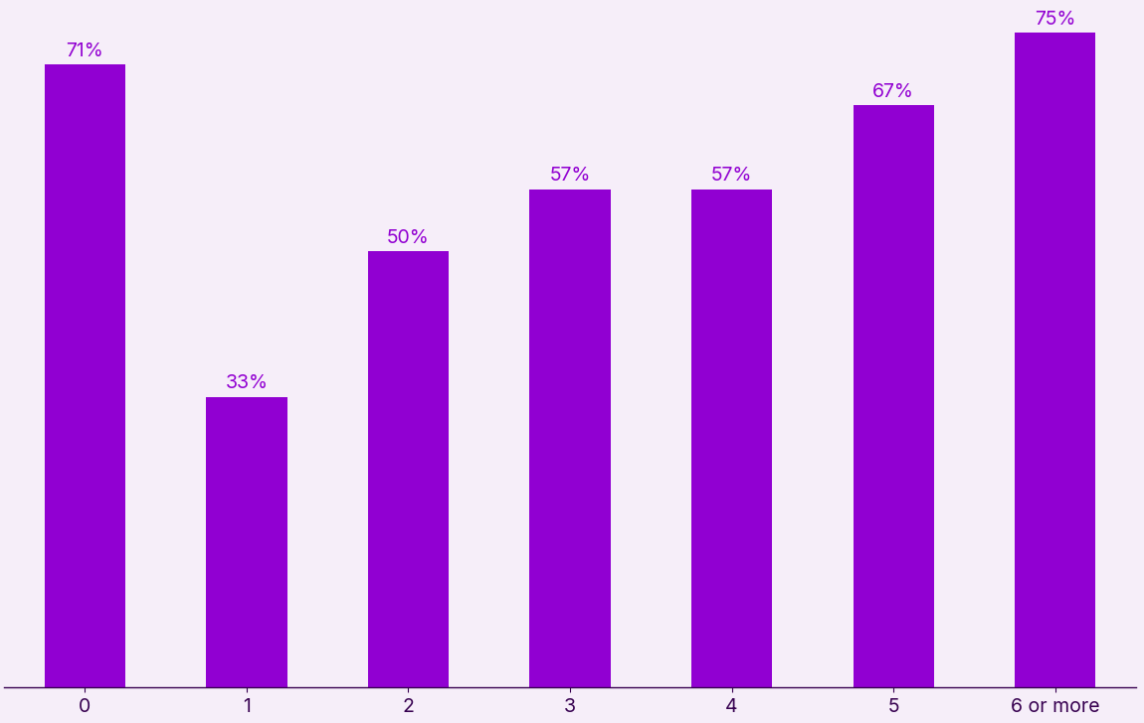
The survey had responses from organizations of all sizes, from small and medium-sized businesses to larger enterprises.

High-performing platforms generally were from a larger organization size and a larger developer base, particularly in organizations with over 100 developers. These organizations tend to have the highest success rates, exceeding 85%.

There is a notable performance dip for organizations with 1000 to 4999 employees, possibly due to the "growing pains" that accompany scaling a platform and organization. As a company transitions from a medium-sized business to an enterprise, the growth of the platform initiative may be faster than its ability to be managed. This period of growth often brings shifts in organizational and team priorities, which could contribute to the decline in performance.

Success by number of metrics tracked

When metrics aren't tracked, you will likely believe all goals are being met. However, it is questionable whether this confidence is well-placed without tracking. Not measuring any metrics creates a ***zero-metric success illusion*** where platform teams report high success rates simply because they're not collecting data that might contradict their assumptions[10]. This could occur from a lack of objectives or relying on anecdotal evidence, leading to no clear goal or motivation for improvement.

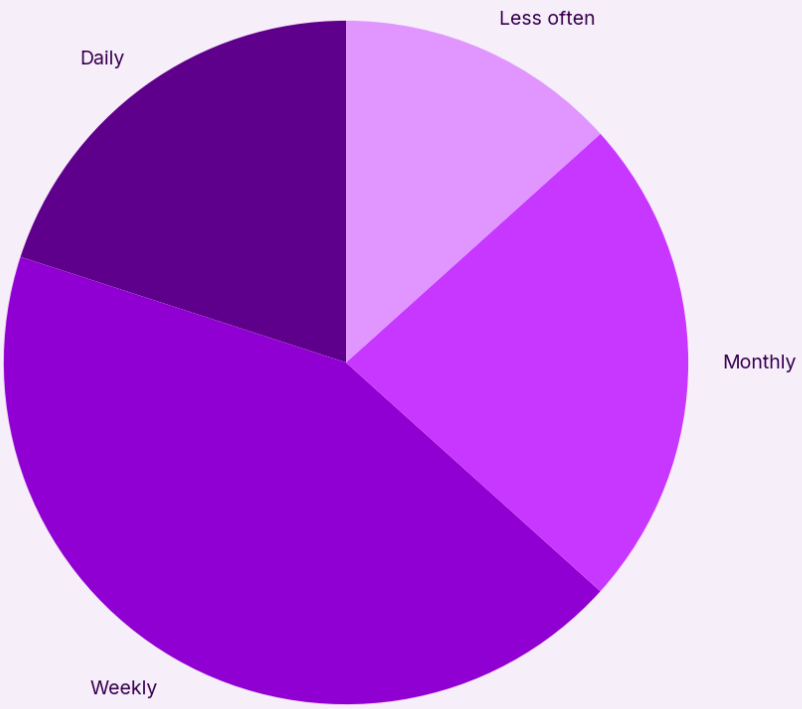


High performance based on % of metric count segment

Organizations that track more metrics are likelier to meet many or all goals. However, many high-performing organizations rely primarily on technical metrics (deployment time, frequency, build time, etc.) without measuring user satisfaction. While these metrics make platform teams feel successful, they miss out on improving developer experience.

Recommendation: Use a variety of metrics, including technical performance (e.g., DORA metrics) and user satisfaction in some form (e.g., MONK metrics) to gain broad perspectives of success. This creates data-driven feedback loops to identify issues and inform decisions affecting users and the product.

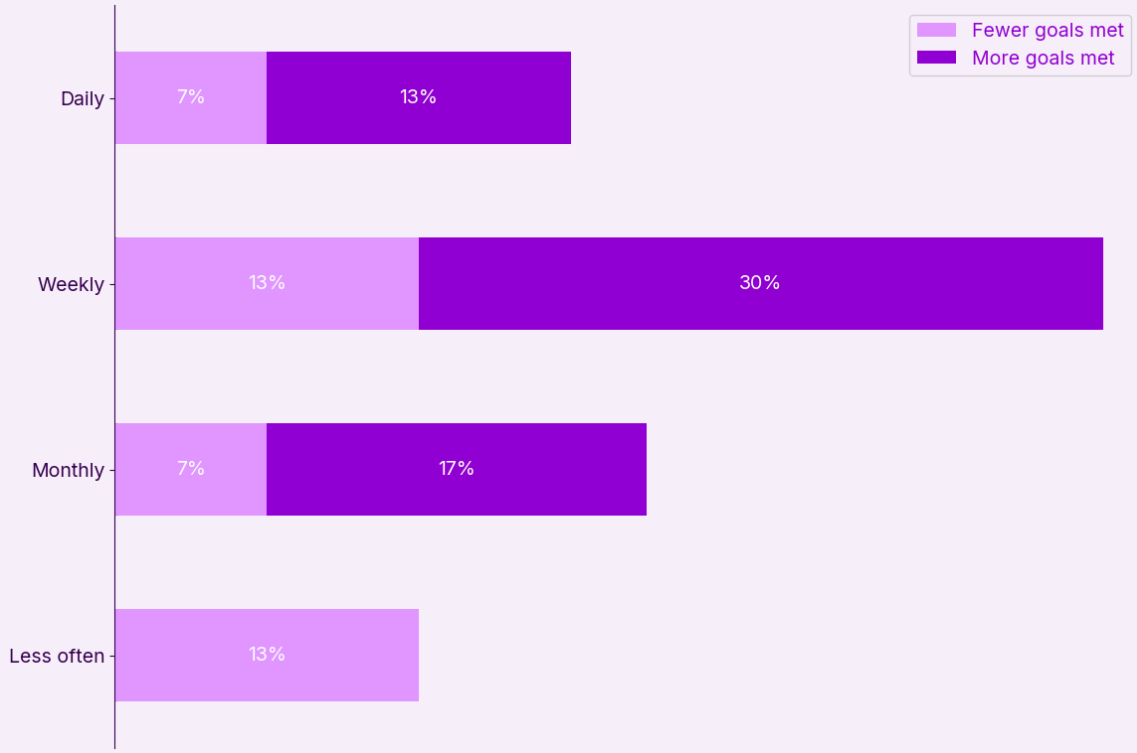
Feedback frequency



Feedback frequency by % of organizations

When looking at the feedback frequency between producers and consumers, most platform teams (43%) are talking to developers weekly, followed by monthly (24%), then daily (20%) with the "less often" (e.g., quarterly or never) feedback frequency being used the least (13%).

Success by feedback frequency

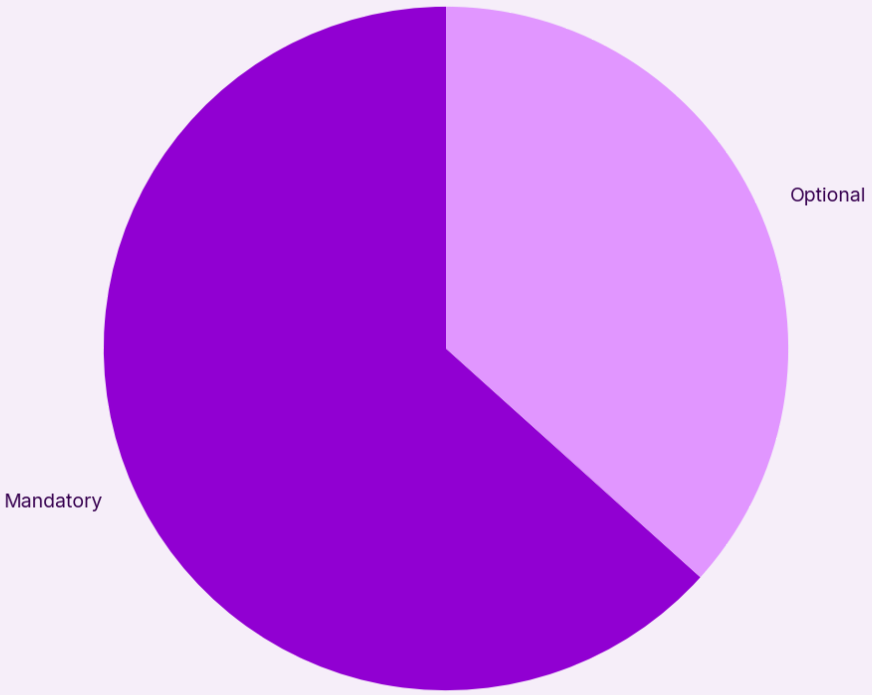


Feedback frequency and performance

While there are only slight differences in performance for teams receiving daily, weekly or monthly feedback, what stands out is getting feedback less often guarantees a low-performing platform. This suggests that receiving feedback, regardless of frequency, is crucial and that the type of feedback may be more important than its frequency.

Regular feedback is a key factor in high performance, as metrics should be used to visualize your efforts to continuously improve **DORA 2024 Accelerate the State of DevOps report**[5].

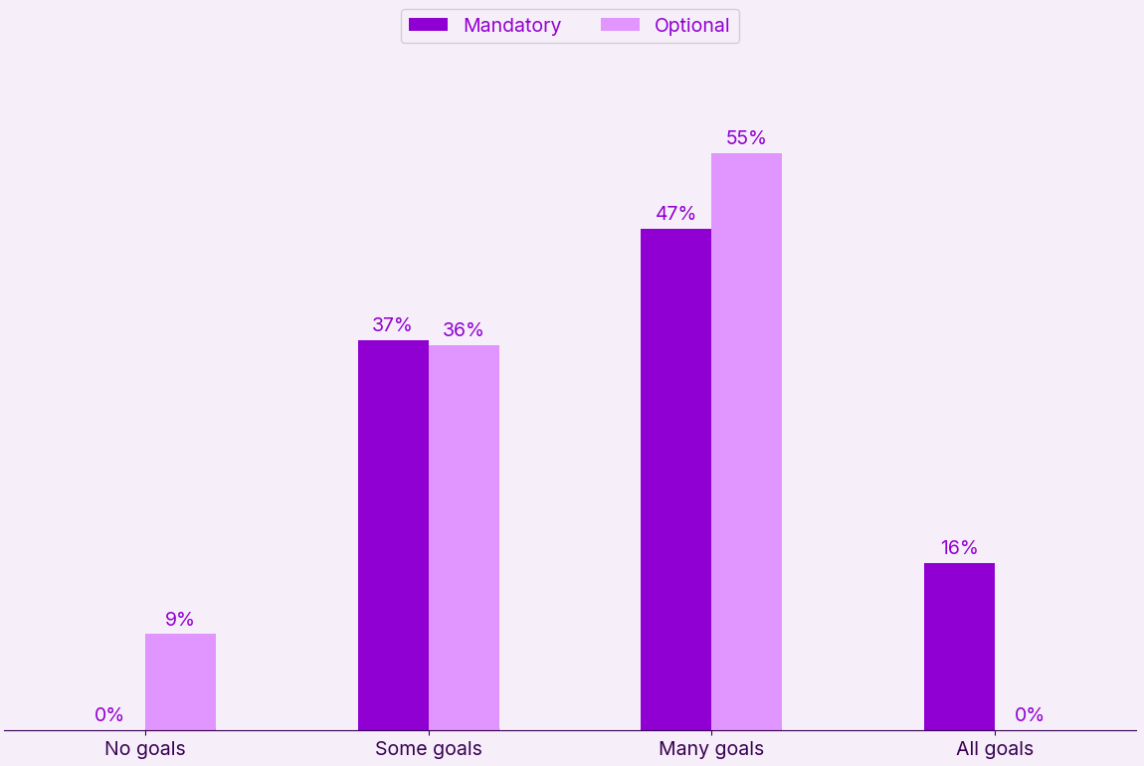
Adoption strategy



Adoption strategy by % of organizations

It is interesting to note the contradiction between Platform Engineering theory and practice. While reading material promotes the idea that a **platform should be composable**, and does not force an inflexible way of working upon a delivery team[11], this data suggests a different reality: with 63% of platforms are mandated compared to 37% of platforms being optional.

Success by adoption strategy



Goal attainment comparison by adoption strategy

A closer look at performance regarding adoption strategy found that mandatory platforms can boost performance, with most compulsory initiatives meeting many (47%), if not all (16%), of their goals. This could be a direct result of its non-negotiable status, or a side-effect of producer preference for mandatory platforms.

A key factor in implementing platforms is that they should be secure and compliant, based on rules and standards defined by their organization[12]. The **Puppet 2024 State of DevOps report** reported that 43% of platforms have dedicated security and compliance teams, highlighting the growing importance of proactive security management[12]. Therefore, making the platforms mandatory could address these critical compliance and security obligations needed within organizations.

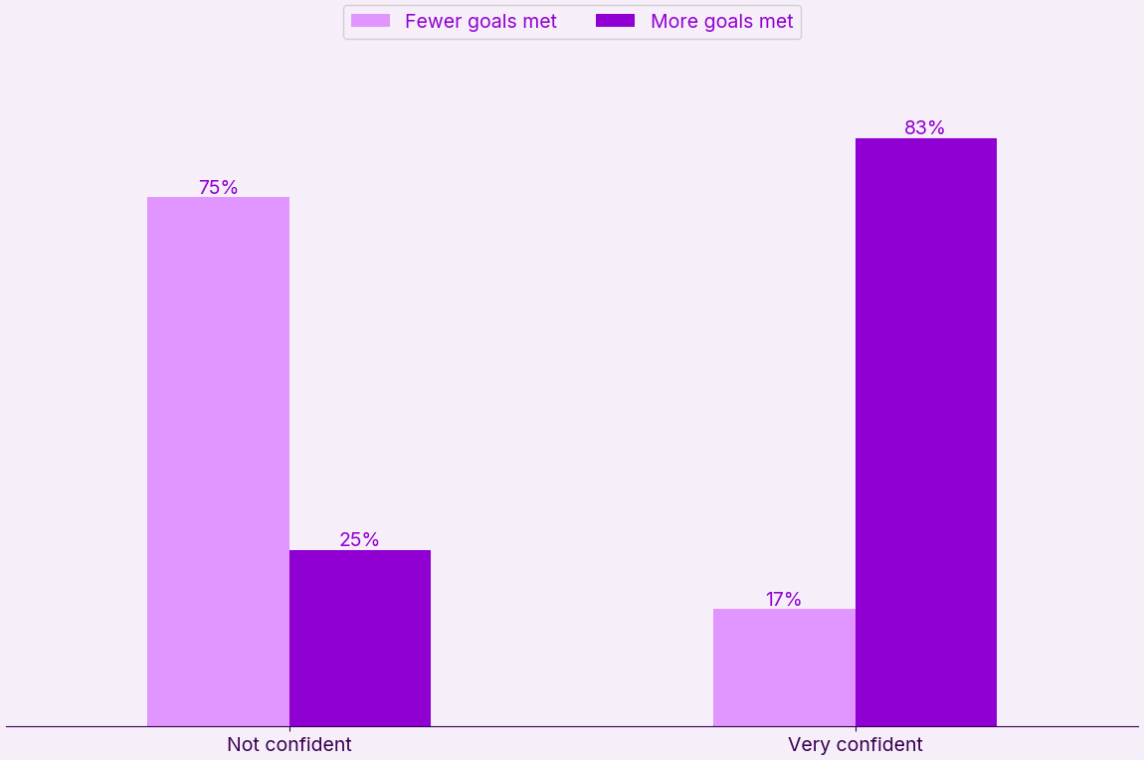
Furthermore, when developers are required to use a platform, they're compelled to spend time and effort making it work for them, which leads to necessary optimizations and improvements. In contrast, an optional platform that's too complex or difficult to use might get ignored, leading to low adoption and a feeling that the initiative has failed.

> You should make policy rather than platforms mandatory. When a team forging their own path is held to the organization's standards for security and compliance they are highly motivated to adopt a platform that helps them achieve these goals.
>
> *Steve Fenton*

Budget safety

When it comes to platform budgets, almost two-thirds (60%) of organizations feel *somewhat confident* that their funding is secure over the next five years. For the remaining third, the platform's performance is a strong indicator of future budget confidence.

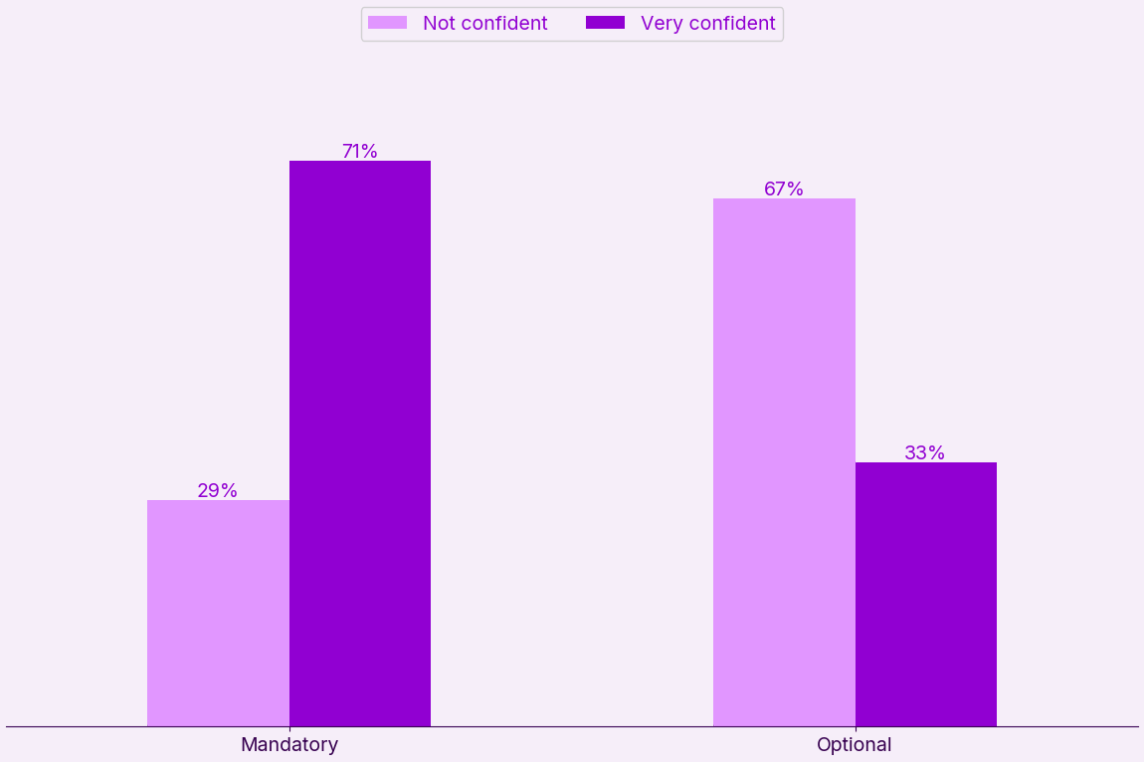
Successful platforms retain funding



Budget confidence based on goal attainment

As expected, the more a platform meets its goals, the safer its budget is predicted to be. The majority of platforms (83%) that were confident that their budget would be safe over the next 5 years, were meeting many or all goals.

Adoption strategy and budget



Budget confidence based on adoption strategy

When considering the adoption strategy regarding budget safety, we found mandatory platforms were almost 2.5x more likely to have budget safety. There is a disconnect between producers and consumers regarding confidence in budget safety and platform success. This suggests that mandatory platform adoption creates heightened confidence among producers, where compliance is interpreted as success and translates to confidence in budget safety.

However, this confidence may be well-founded. Producers' success perceptions could reflect alignment with organizational goals, coinciding with the identified adoption drivers (i.e., efficiency and operational improvements). From this perspective, mandatory adoption serves the organization's objectives, so may justify the confidence in budget safety.

On the other hand, consumers may hold different views about platform value and performance that aren't reflected in these measures. While mandatory adoption may achieve organizational goals, it does not fully address end-user experience and satisfaction.

Platform initiatives should recognize that the adoption strategy influences budget confidence and perception of success. The disparity between mandatory and optional platform confidence levels indicates that forced adoption might mask genuine user satisfaction issues, even when the platform successfully meets business objectives. This warrants further investigation.

Recommendation: Organizations should make sure platform goals and success metrics are uniformly understood through transparent communication, and feedback loops that capture genuine user perspectives. Platform success requires ongoing dialogue between organizational objectives and user experience needs.

The platform maturity model

The CNCF has published a model that can help you further assess your platform initiatives. It uses five aspects of platforms along with a scale of implementation that can help you find where you are and plan future improvements.

Aspect	Provisional	Operational	Scalable	Optimizing
Investment	Voluntary/temporary	Dedicated team	As a product	Enables ecosystem
Adoption	Erratic	Extrinsic push	Intrinsic pull	Participatory
Interfaces	Custom processes	Standard tooling	Self-service solutions	Integrated services
Operations	By request	Centrally tracked	Centrally enabled	Managed services
Measurement	Ad hoc	Consistent collection	Insights	Quantitative and qualitative

This model is accompanied by a **white paper** that goes well beyond the checkbox exercise of typical maturity models[1]. You can use the **Platform Engineering Maturity Model Assessment** to help find your current location in the model and possible areas of improvement[13].

Conclusion

Platform Engineering can significantly improve developer productivity when platforms fill real capability gaps and address genuine developer needs. While platforms show strong success potential, they require patience, comprehensive success metrics for all stakeholders, and an understanding of what works in practice.

Key success factors

1. Time is the most significant variable, with platforms achieving higher success rates as they mature. Meaningful transformation requires sustained investment over 3+ years.
2. Organizations tracking multiple and a diverse range of metrics consistently outperform those without data-driven feedback loops, demonstrating the critical importance of measurement-driven approaches.

Differences in practice

While literature favors optional adoption, 63% of successful platforms are mandated, with mandatory platforms appearing to achieve higher success rates and budget security.

These results may reflect different success criteria rather than misleading results, as producers' positive ratings could reflect the organizational operational objectives (i.e., efficiency improvements and standardizing processes) rather than developer experience benefits.

The findings suggest that mandatory adoption may address organizational goals while creating gaps in user satisfaction, indicating that success metrics vary depending on stakeholder perspective.

Navigate the J-curve

The **DORA 2024 Accelerate the State of DevOps report**shows platforms typically cause a 14% decrease in stability and 8% decrease in throughput during years 2-5 (i.e., a normal "J-curve" pattern)[5]. Organizations must anticipate and manage these growing pains rather than panic, understanding this dip is part of the maturation process.

Success ultimately depends on allowing time for platform initiatives to mature, establishing holistic measurement approaches, ensuring clear communication with all stakeholders, and grounding decisions in real-world evidence rather than academic ideals. Understanding these real-world patterns helps organizations navigate the proven value potential more effectively and achieve meaningful long-term platform initiatives that serve organizational objectives and developer needs.

Contributors

Steve Fenton

Principal DevEx Researcher

A long-time software delivery practitioner and research lead for The Platform Engineering Pulse report.

Charlotte Fleming

Researcher

A researcher and neuroscientist with extensive experience as a research analyst and contributor to The Platform Engineering Pulse Report.

Saim Safder

DevOps and Cloud Engineer

A CNCF Ambassador and Cloud Native Podcast host with extensive experience in banking and financial services, passionate about making open-source technology more accessible.

Ships Mahindra

Senior Product Manager

A passionate a user-centered product manager who specializes in problem-first approaches, strategy, and delivery.

Matt Allford

Developer Advocate

A technologist and content creator who combines 15+ years of deep technical experience with a passion for community education.

Jed Lehmann

Senior Design Manager

An end-to-end product designer with 18 years experience who solves high-impact problems in complex products and domains.

Method

We conducted interviews and ran a questionnaire asking open questions. We obtained responses from platform sponsors, producers, and consumers of internal developer platforms. We performed text analysis on the questions to find common themes in their responses, such as what motivated the implementation of Platform Engineering, the features and services offered by platforms, and what new features they were considering.

We used this data to help generate questions for the survey, which included a range of open-text, multiple-choice, and Likert-type questions, adding an "other" option for people to add entries we didn't discover during interviews. The survey was distributed through a market research company to obtain responses outside our close contacts. This helped us get a more randomized sample of people involved with platforms, rather than people closely tied to our workplaces and communities we are involved with, such as the Cloud-Native Computing Foundation (CNCF) and the CD Foundation (CDF).

Responses were analyzed to view overall trends and find patterns between adoption characteristics and platform success. Responses were cleaned, for example, to de-duplicate user-entered data like "deployment time" and "time to deploy" to analyze the common customer-centric measures and the number of performance metrics tracked. We developed a scoring mechanism to convert responses into normalized scales for platform performance and success. This made it easier to compare different performance relationships.

We analyzed direct relationships to ensure we didn't miss essential connections between practices and outcomes. Findings were cross-validated between tools to reduce errors. The classification of goal attainment (none, some, many, and all) into "fewer goals met" or low performers (none, some) and "more goals met" or high performers (many, all) have been used to simplify comparisons. The assumption is that a platform meeting many or all goals performs well.

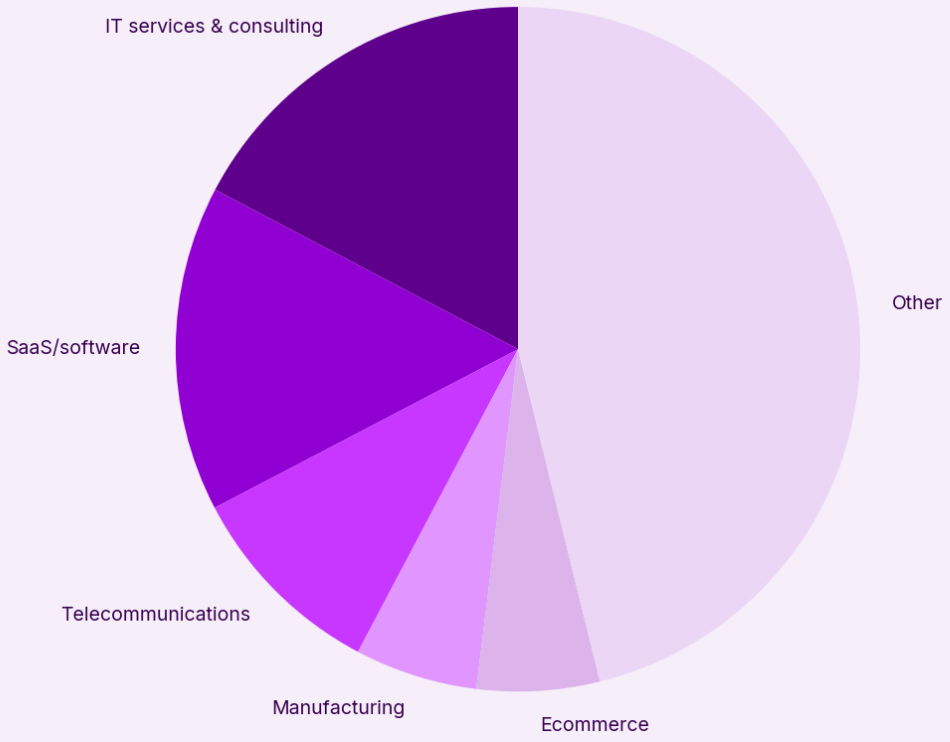
Analysis

We used Microsoft Excel and Python for data analysis and digital notebooks to organize and annotate the different threads of investigation. Software delivery is complex and requires a large number of practices for success. That means the strength of individual practices is often low, while combinations amplify each other to create a larger effect.

We are treating the findings as a baseline for further exploration and plan to explore further through larger samples to validate grouped and filtered views of data.

Firmographics

Industry



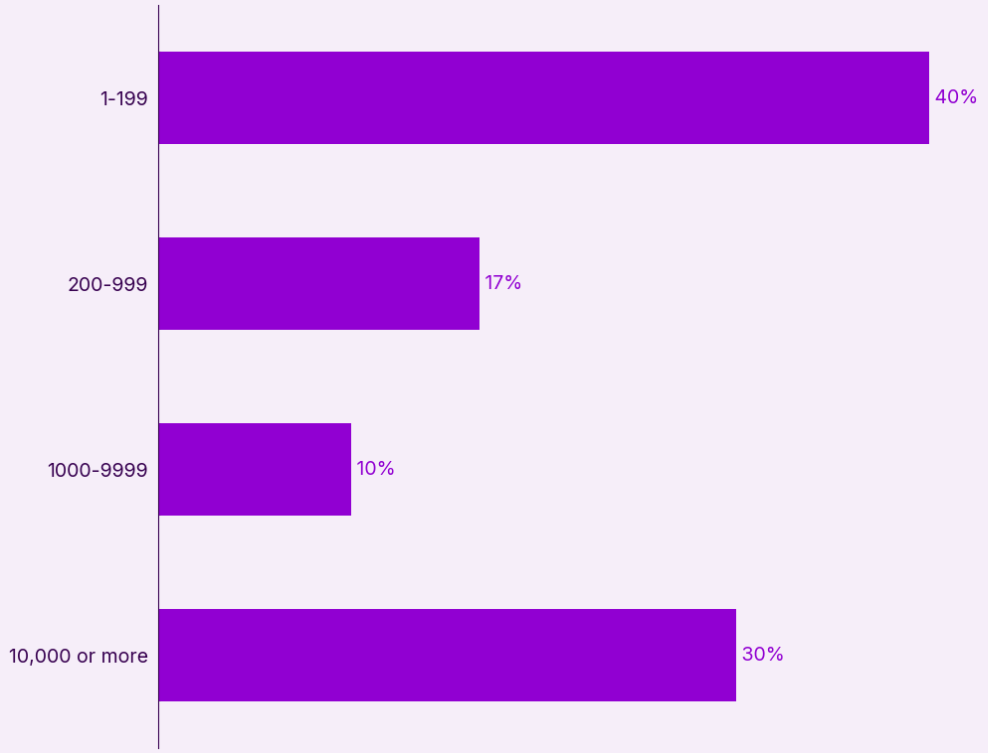
Responses by industry

Overall, more than 20 industries were represented in the responses. The top 5 industries were IT services & consulting (17%), SaaS/software (15%), telecommunications (10%), manufacturing (6%), and ecommerce (6%).

Significant industries not in the top list include:

- Automotive
- Aviation and aerospace
- E-learning
- Fintech
- Government administration
- Hospitals and healthcare
- Investment banking

Organization size



Responses by organization size

The survey had responses from organizations of all sizes, from small and medium-sized businesses to large enterprises.

Sponsors

Octopus makes it easy to deliver software to Kubernetes, multi-cloud, on-prem, and anywhere else at scale, in one platform.

Visit **octopus.com** to find out more.

References

1. https://tag-app-delivery.cncf.io/whitepapers/platforms/
2. https://www.linkedin.com/pulse/platform-vs-devex-teams-whats-difference-abi-noda-kmypc/
3. https://dora.dev/guides/dora-metrics-four-keys/
4. https://octopus.com/devops/metrics/monk-metrics/
5. https://dora.dev/research/2024/dora-report/
6. https://www.atlassian.com/devops/frameworks/team-topologies
7. https://www.qualtrics.com/en-au/experience-management/customer/net-promoter-score/
8. https://www.qualtrics.com/en-au/experience-management/customer/what-is-csat/
9. https://www.puppet.com/resources/state-of-devops-report
10. https://octopus.com/blog/how-organizations-measure-platform-engineering#breaking-the-success-illusion
11. https://martinfowler.com/articles/talk-about-platforms.html
12. https://www.puppet.com/blog/state-devops-report-2024
13. https://cloud-native-platform-engineering.github.io/pemm-assessment/

Further reading

1. https://teamtopologies.com/key-concepts-content/what-is-a-thinnest-viable-platform-tvp
2. https://ieeexplore.ieee.org/document/8851296
3. https://martinfowler.com/articles/talk-about-platforms.html
4. https://queue.acm.org/detail.cfm?id=3454124

PLATFORM ENGINEERING

Octopus Deploy

Level 4, 199 Grey St
South Brisbane, QLD 4101,
Australia

sales@octopus.com
+1 512-823-0256
octopus.com