FUTURE OF

PLATFORM ENGINEERING REPORT

How organizations are evolving platform engineering, what's working, what isn't, and why

2026



Octopus Deploy

Dr. Charlotte Fleming and Steve Fenton

The Future of Platform Engineering report

How organizations are evolving Platform Engineering, what's working, what isn't, and why

Dr. Charlotte Fleming and Steve Fenton

The Future of Platform Engineering report

How organizations are evolving Platform Engineering, what's working, what isn't, and why

Dr. Charlotte Fleming and Steve Fenton

Copyright © 2026 by Octopus Deploy Pty. Ltd.

All rights reserved.

Except for brief quotations in critical articles or reviews, no part of this book may be reproduced in any manner without prior written permission from the publisher, Octopus Deploy. Level 4, 199 Grey Street, South Brisbane, QLD 4141, Australia.

This publication contains opinions and ideas of the author. It is intended to provide helpful and informative material on the subjects addressed in the publication. The author and publisher make no warranty, express or implied, with respect to the material contained herein.

Read our other research publications at octopus.com/publications. For questions about this report, or to request permission to reproduce any part of it, email **research@octopus.com**.

Octopus, Octopus Deploy, and the Octopus logo, are registered trade marks of Octopus Deploy Pty Ltd.

Ed.1.2026.2

Executive summary 1

Introduction 3

Adoption insights 4

1: Motivation 7

Why build a platform? 7

Sponsor, producer & consumer priorities 9

Satisfaction & well-being 10

2: Features 15

Top features 16

Platform breadth 16

Features & goal delivery 18

Features found together 20

Platform features & AI 22

Build or buy 23

3: Obstacles 29

Common obstacles 29

Crucial obstacles 30

4: High-performance 37

Platform maturity 37

Adoption strategy 39

Developer satisfaction & success 42

Feedback frequency & channels 43

Budget confidence 46

5: Measurement 49

MONK metrics 49

6: How the findings connect 57

Theme 1

Mandatory platforms, organizational governance & goal delivery 58

Theme 2

Leadership & job satisfaction 59

Theme 3

Workload & well-being 60

Theme 4

AI impact on platforms 61

7: Predictions & Conclusions 63

Conclusion 65

Authors 66

8: Methods & firmographics 69

Quantitative survey 69

Data preparation 69

Demographics & firmographics 74

Sponsors 78

References 79

Executive summary

Platform Engineering has moved from a frontier practice to a mainstream industry-standard. Today, it's an established discipline with recognizable patterns and a maturing body of evidence.

This report draws from two sources: primary survey data from 379 technical platform practitioners worldwide, and a review of the current literature to establish where the discipline now stands.

A consistent theme runs through the findings, separating platforms that succeed from those that struggle. The decisions and practices around the technology is what determines success, like how adoption is managed, how clearly strategy is set, and how engaged leadership is.

> **Key finding:** The foundations for platform success are the people and practices around the technology.

Headline insights

AI	**AI is now part of every platform initiative**, but its value is not evenly distributed. Teams on mature, well-designed platforms get more out of AI than teams with ad hoc setups (**DORA, 2025**).
Automation	**Automation is the primary reason teams build platforms:** Organizations primarily adopt platforms for automation and efficiency purposes, and developer experience is a secondary consideration.
Strategy	**The organizational constraint is strategy, not technology:** Struggling teams most often report a **lack of a clear strategy** as an obstacle rather than the **technical complexity** of building platforms.
Adoption	**Mandatory platform/tool adoption outperforms optional adoption on goal delivery:** A platform that developers are forced to use is associated with higher goal attainment than optional platforms.
Direction	**Clear leadership direction tracks with success:** Platforms aligned with leadership direction are more likely to deliver their goals.

Introduction

The concepts behind Platform Engineering have a long history, but there has been a recent rapid increase in Platform Engineering initiatives. A few years ago, it was an emerging discipline, largely borrowed from a small group of high-output engineering teams at web hyperscalers. Today, it's a recognizable practice across most large software engineering efforts. It has dedicated conferences, vendors, certifications, and contested ideals.

Using DORA's definition as the starting point, "Platform Engineering is a sociotechnical discipline that sits at the intersection of how teams work together and the technical work of automation, self-service, and repeatability", often referred to as internal developer platforms (**DORA, 2026**). It involves building the shared infrastructure that development teams need to build, test, and deploy software safely and reliably (**DORA, 2025**).

In this report, a **platform** is a set of features and capabilities shared across multiple applications or services. An organization may run several overlapping platforms, but we refer to them collectively as "the platform". A **platform team** is a group dedicated to building and maintaining a platform, but a platform can exist without a platform team.

This report builds on the **2025 Platform Engineering Pulse report** by Octopus Deploy, which established a baseline for platform adoption. Several findings have remained consistent, but there are also changes that provide insight into how the discipline is evolving.

The stable results include:

- Platforms are adopted for efficiency more often than developer well-being.
- Platforms improve productivity when they fill genuine capability gaps.
- Successful adoption requires patience and attention to what works in practice, not strict adherence to theory.

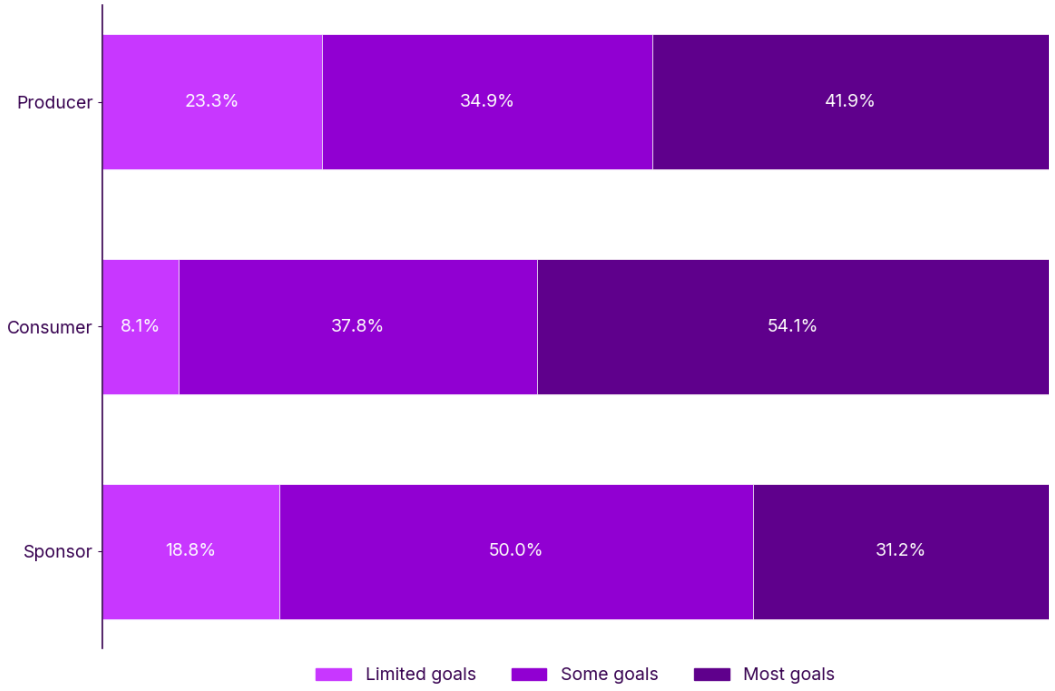
This research highlights what high-performing teams do that makes their platforms more successful. We go deeper into why organizations adopt a platform in the first place, the reasoning that shapes what they build and buy, and the obstacles they meet along the way. Continuous themes across each section are delivery performance and developer well-being.

Platform Engineering adoption insights

Throughout this report, **"high-performers"** are defined as teams/respondents that have reported to successfully achieve **most goals** they set for their platform initiative. One challenge in measuring Platform Engineering initiatives is the lack of consensus on expected outcomes. That makes it necessary to measure each initiative by a self-assessed perception of how well its specific goals have been achieved. While this has disadvantages, it is uniquely able to assess whether the organization is getting the value they intended from their investment and can highlight perceptual differences between those building platforms and those sponsoring or using them.

We categorize respondents into 3 personas:

- **Producers:** Those who build internal developer platforms.
- **Consumers:** The people who use these platforms.
- **Sponsors:** The stakeholders who fund or direct the platform initiative.



Goals delivered by platform persona. "To what extent does your organization's current internal developer platform deliver the expected benefits? (single-select)".

Sponsors are frequently technology optimists who see the potential of new tools and are insulated from the implementation problems. On the other hand, producers are more familiar with the potential drawbacks of changing systems. These personas offer a valuable perspective on the gaps in how each group perceives platforms within their organization.

The figure above shows that consumers are the most positive about goal achievement, with over half (54.1%) saying they achieve most goals, and just 8.1% reported limited goal delivery. Producers are most likely to report limited goal delivery, even with 41.9% reported to achieve most goals. Sponsors sit between the two with 50.0% reporting some goals were delivered, and 31.2% reporting most goals.

The gap here may reflect proximity or knowledge bias, where producers see the platform's unfinished work, while consumers judge it by the experience it delivers for them on a daily basis. This could suggest an encouraging signal for platform teams: the people using the platform rate its success more highly than the people building it.

SECTION 1

Why organizations build platforms

This section explores what benefits organizations hope to get out of Platform Engineering initiatives, and whether those benefits actually align with measurable outcomes like job satisfaction, well-being (i.e., reduced burnout), and work-life balance (i.e., reduced work-life impact).

Key takeaways

1. Automation is the leading reason organizations build platforms.
2. Motivation to build platforms signals the state the team is already in, not just a goal.
3. Organizations listing **developer productivity** as an adoption reason have higher well-being and better work-life balance.
4. Automation and burnout-driven motivations signal teams under pressure, while experience- and autonomy-led motivations mark healthier teams.

Why organizations build platforms

Automation is the most common reason and benefit for organizations to invest in Platform Engineering initiatives in 2026. We found that 69.6% of respondents selected automating more tasks as their motivation, ahead of **improving efficiency** (60.8%) and closely followed by **reducing complexity** (58.8%). **Standardizing processes** (55.9%) and **increasing developer productivity** (54.9%) were the other leading motivations found this year.

This order has changed over the past year. In the **2025 Platform Engineering Pulse report**, the leading motivations were improving efficiency, standardizing processes, and increasing developer productivity, with automation coming in fourth. Within 12 months, automation has moved from the middle of the pack to the most common driver of platform adoption. This change suggests that automation has been reframed from a nice-to-have to a competitive necessity for scaling software delivery.

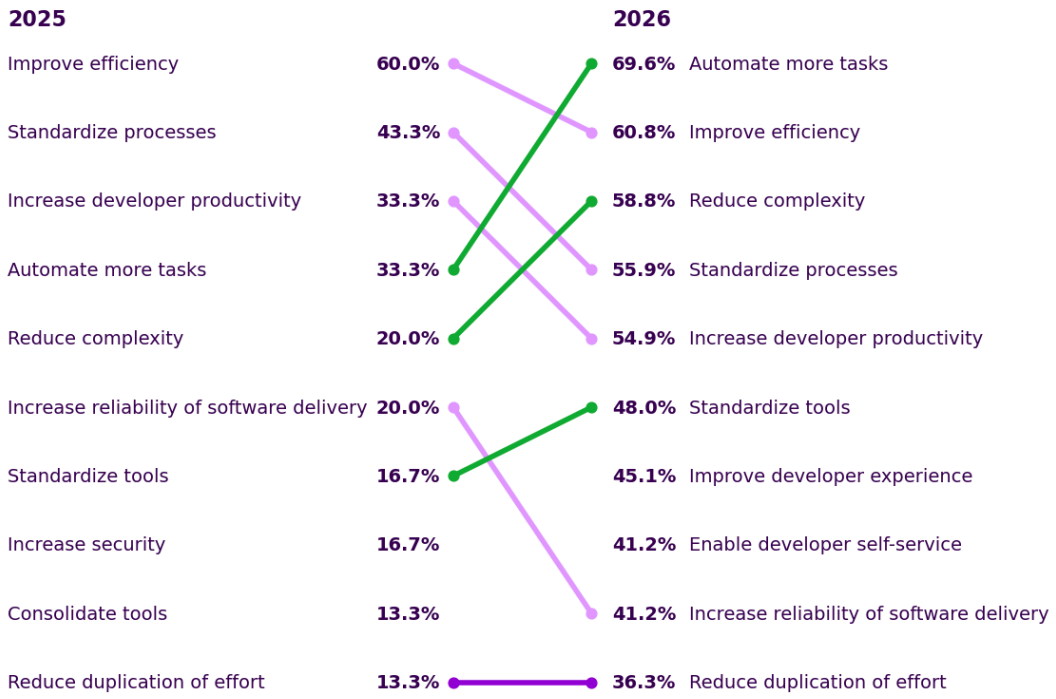


Figure 1.1: *Top 10 reasons for adopting platforms in 2025 and 2026.*

One likely reason for the shifting emphasis is AI. AI has extended automation beyond rule-based scripting by incorporating agentic workflows into systems, which enables automation to handle more complex unstructured work. Also, as AI accelerates individual tasks like code generation, it exposes downstream bottlenecks, increasing the urgency to automate the operational hurdles where work now gets stuck (**Faros AI, 2025** and **Varshitha, 2026**).

As a result, platforms are taking on a broader scope, making platform automation a more ambitious endeavor to build around. This is consistent with broader industry trends and our results, which show that 78.4% of organizations have shifted priorities to incorporate AI into their applications and services over the past year. While we cannot definitively attribute this shift in motivation to AI investment, both trends are occurring in parallel.

78.4% of organizations have shifted priorities towards AI

What sponsors, producers, and consumers each prioritize

The three platform persona groups largely agree on the reasons platforms are built, indicating that motivations for Platform Engineering initiatives are relatively consistent across organizational perspectives. There is only one difference in the data: producers selected **standardize tooling** significantly more than consumers or sponsors (**p=0.02).

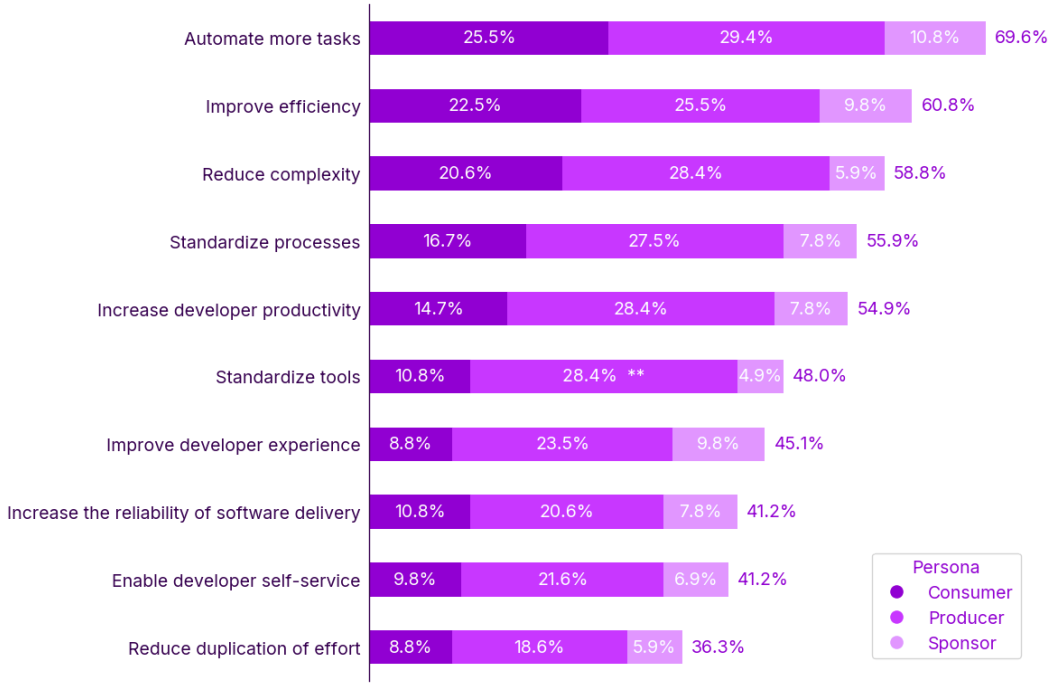


Figure 1.2: *Top 10 reasons for platform adoption by platform persona. Shown as percentage of respondents who selected the option by platform persona. Asterisks indicate significance, with Bonferroni correction applied (**p<0.05).*

This gap highlights a clear operational reality in which teams building and maintaining the platform experience the burden of tool sprawl firsthand. If platform producers do not actively and continuously simplify the tooling landscape, the maintenance overhead and support effort will fall directly on them (**Netesanyi, 2026** & **Kropov, 2026**). As one expert put it:

> With only three or four people left on the platform team, the goal is to automate more. But you can't automate until you've standardized the process into a predictable, well-documented flow.
>
> *Denis Jajcevic, Principal Platform Architect at CROZ*

However, this push for standardizing tools creates an interesting balance. While some redundant tools enter the system through organic sprawl, others are selected and intentionally adopted to address specific use cases or domain requirements. This brings Chesterton's Fence into play: abrupt tool consolidation without understanding why specialized tools were originally adopted risks resurfacing previously solved problems for application teams, reintroducing friction for platform builders in the name of standardization (**Furnell, 2024**). This likely increases friction for platform consumers who must work around gaps caused by blunt consolidation. This is a failure mode of Platform Engineering.

What motivations reveal about satisfaction and well-being

Is the Platform Engineering motivation associated with well-being outcomes? We asked a series of questions about how people feel about their work and tested whether any stated motivation was associated with three distinct measures:

- **Overall job satisfaction**: measures how positively people feel about their job and work experiences overall. Higher job satisfaction scores are a positive outcome.
- **Well-being (inverted burnout)**: measures feelings of exhaustion and cynicism related to work, resulting from a prolonged response to chronic stressors on the job. Higher well-being scores are a positive outcome (**DORA, 2025** & **American Psychological Association, n.d.**).
- **Work-life balance (inverted work-life impact)**: measures how much negative feelings about work spill over into life outside work, affecting personal time. Higher work-life balance scores are a positive outcome.

The organizational reasons behind adopting Platform Engineering may signal wider characteristics: its culture, its values, and its attitudes towards staff.

The clearest finding was respondents reporting **increased developer productivity** as a motivation for adoption were significantly more likely to report better well-being than those who did not (**p=0.0021). The same motivation also points towards better work-life balance (*p=0.042).



Figure 1.3: *Adoption reasons by job satisfaction, well-being (inverted burnout), and work-life balance (inverted work-life impact). Rank-biserial heatmap showing the effect for respondents who selected expected benefits, compared with those who didn't, across 3 outcomes. Asterisks indicate directional significance (*p<0.05) and Bonferroni correction applied (**p<0.0028).*

A consistent pattern is seen through productivity- and efficiency-oriented motivations: **increasing security**, **enabling developer self-service**, **standardizing tools**, **reducing duplicated effort**, and **increasing delivery speed**, are all associated with better well-being. **Reducing duplicated effort** and **increasing delivery reliability** also lean towards better work-life balance.

Automating more tasks, the most common adoption motivation, has a slight association with lower well-being. This may reflect breadth rather than automation itself as these respondents selected more motivations overall, suggesting that automation is a route to several other benefits rather than a focused goal. Automation is otherwise well established as a driver of efficiency, productivity, reliability, and standardization (**DORA, 2024**).

Improving the developer experience was weakly associated with better well-being and better work-life balance. The direction is telling here: teams that build their platforms around **developer experience** also tend to report healthier working lives, which fits with the idea that this motivation marks teams already in a good place rather than those still under pressure (**Noda et al., 2023**).

For job satisfaction, the links are more subtle. **Empowering developer decision-making**, and **reducing complexity**, and **increasing developer productivity**, all point in the same direction towards higher job satisfaction, aligning with autonomy- and clarity-oriented goals. This suggests that teams motivated by giving developers more control and a cleaner, simpler environment are more satisfied in their jobs (**Deci & Ryan, 2000**).

There is a tension here with **empowering developer decision-making** associated with higher job satisfaction, however, slightly lower work-life balance. This could indicate that giving developers more control also entails more responsibility, which can affect personal time, even when it makes work more satisfying (**Storey et al., 2021**).

The reverse is also interesting: teams that chose **reducing developer burnout** as an adoption motivation reported slightly lower job satisfaction. This signal aligns with automation, where teams tend to name burnout as a goal when they are already under strain, so the motivation may be a marker of a team struggling rather than a pathway out of it.

The takeaway here is that the motivation for platform adoption itself is a signal worth watching. We theorize that struggling teams will reach for automation to reduce duplicated effort. In contrast, teams that are already productive or high-performing will focus on developer well-being and an improved developer experience.

Summary

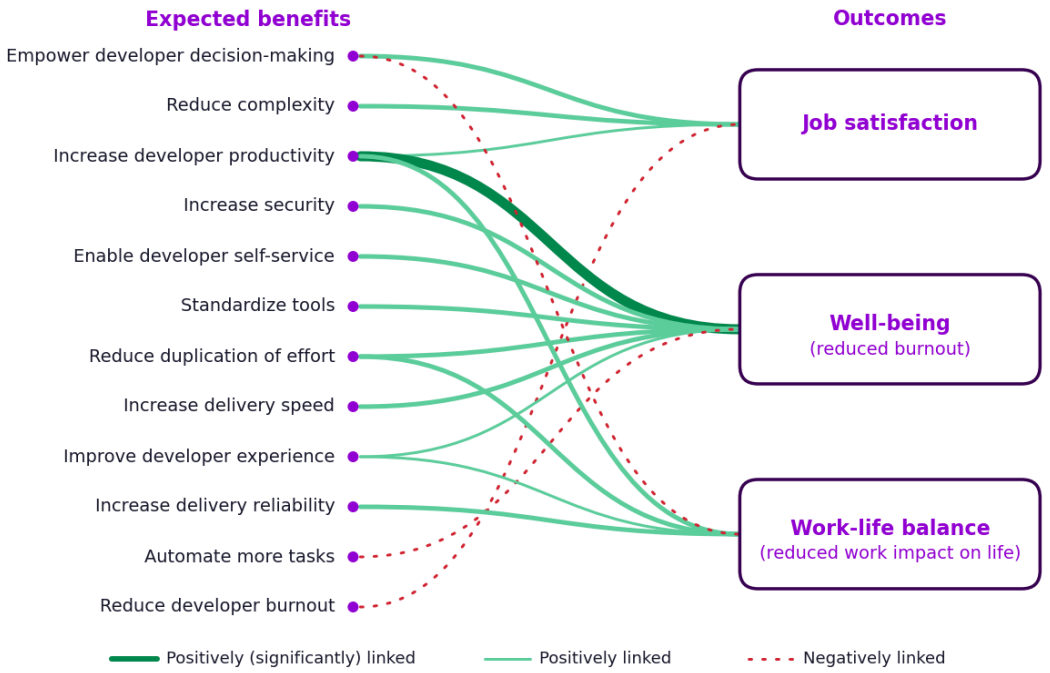


Figure 1.4: *Section 1 summary*

SECTION 2

What teams put into their platforms and the reasoning behind it

Platform teams have increasing ownership responsibilities, including platform size and expansion, as well as build-or-buy decisions. This section describes the current state of platform team ownership: the features platforms most teams are responsible for, how that scope has changed, which features track with goal delivery, how they relate to AI's impact, and the reasoning behind build-versus-buy decisions.

Key takeaways

1. Most platforms share a common set of core features: builds, version control, deployment automation, artifact management, and security/secrets.
2. Features that distinguish high-performers are local development and version control.
3. Ephemeral environments are an advanced milestone, because they require solid build and deployment pipelines in place first.
4. AI amplifies a capable platform, with teams having code coverage, cost control, or ephemeral environments as features, shifting teams from no impact to positive AI impact.
5. Teams are buying, not building. Using an underlying tool dominates and tracks with high-performance.

Top features for Platform Engineering teams

Across all respondents in our survey, we found that most platforms share a common core delivery pipeline. The most common features that platform teams are responsible for are **builds** (85.3%), followed by **version control** and **deployment automation** (both 79.4%), **artifact management** (74.5%), **security scanning** and **secret management** (both 71.6%). This describes a discipline grounded in the essential work of safely and reliably getting changes from a developer's machine to production.

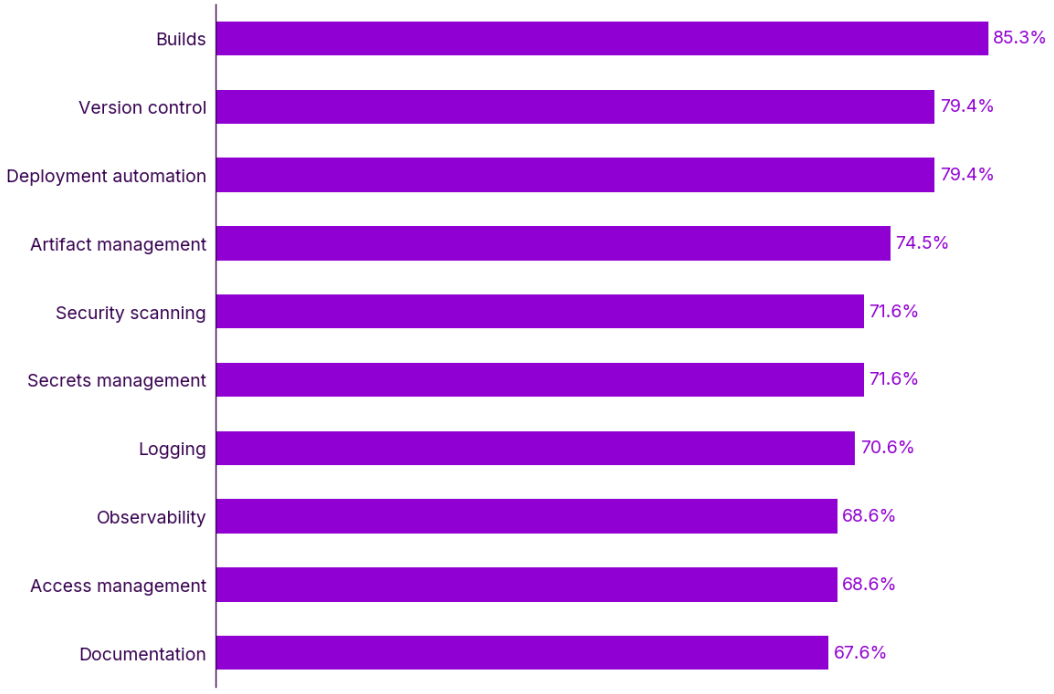


Figure 2.1: *Top 10 features platform teams are responsible for. "What software delivery features and capabilities does the platform support? (multi-select)". Shown as a percentage of respondents who selected the option.*

How platform breadth changed in a year

The shift in the number of features platform teams own is one of the most striking year-on-year findings from the survey. In 2025, the most common number of features platform teams were responsible for was 4. In 2026, the most common feature count is 23, with other peaks around 14 and 18 features. This shows that platform teams are not only more common but also accountable for a far broader set of capabilities than a year ago.

Platform Engineering has grown from a tooling function into a broad, central engineering department. This jump from 4 to 23 features suggests a shift from maintaining isolated tools to managing a unified internal product platform that spans CI/CD, observability, and security (**Tuite, 2025**). This trend aligns with broader industry observations that Platform Engineering is entering its maturity phase, moving from baseline infrastructure towards full-featured developer portals (**Tuite, 2025**).

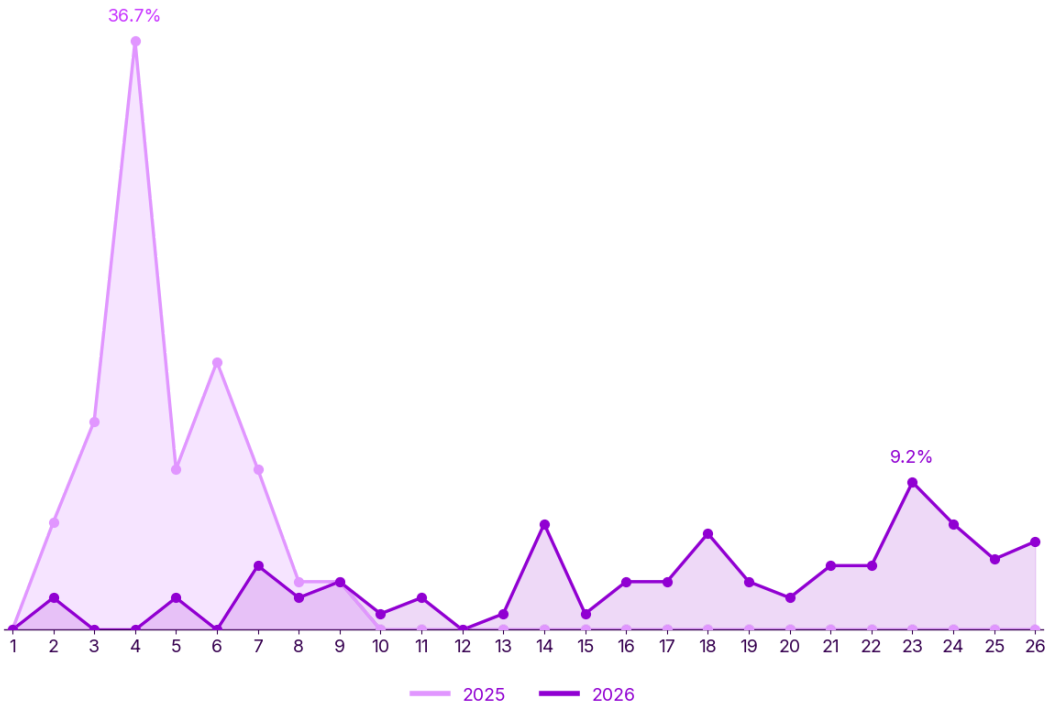


Figure 2.2: *Platform breadth (feature count) comparing 2025 and 2026 platforms. Shown as a percentage of respondents within each year selecting the number of features their platforms are responsible for.*

An important point from our analysis shows that managing more features is not associated with achieving more goals. Expanding a platform's feature list is no substitute for quality, ease of use, and developer adoption. A team managing under-adopted features may deliver far less real value than a team offering a highly effective core set of tools.

The features that track with goal delivery

Having shown that the scope for platform teams is growing, we explored whether particular features are associated with platform teams meeting more of their goals.

When looking at overall adoption, the most common features across respondents who are delivering on their goals include foundational delivery features: **builds**, **version control**, **deployment automation**, **artifact management**, **secrets management**, and **access management**. These are the features that constitute the baseline for modern platform teams (**Cloud Native Computing Foundation, n.d.** & **Tuite, 2025**). From here, we wanted to pinpoint which specific features more strongly distinguish high-performing teams from the rest of the cohort.

To do this, we tested each feature against the goals delivered. The results reveal a consistent pattern: the features associated with goal delivery relate to developer experience, like **local development**, **builds**, **test automation**, **code coverage**, and **feature flags**. **Version control** and **access management** extend this more through self-service: when developers can reach the code and permissions they need without waiting on tickets or hand-offs, the gating that slows teams is removed (**Noda et al., 2023**).

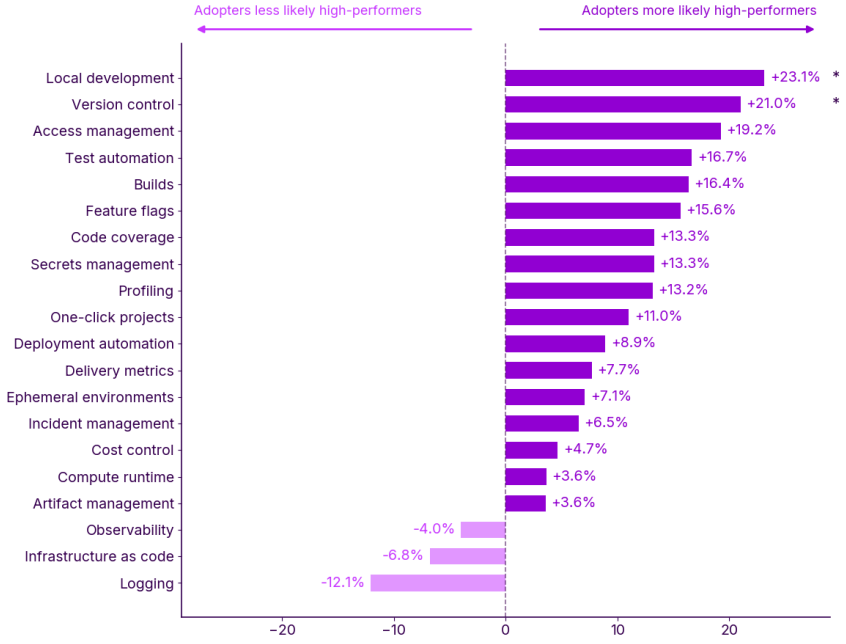


Figure 2.3: *High-performer percentage gap between feature adopters and non-adopters, showing the top 20 features (multi-select). Asterisks indicate directional significance (*p<0.05), with two-sided Mann-Whitney U test.*

The two features that showed the strongest directional association with high-performing teams are **local development** and **version control**. A plausible reason could be that these are features where developers spend most of their time. **Local development** is the innermost feedback loop: every edit-build-test cycle passes through it, with **version control** mediating every merge and collaboration, so the friction here is felt on every task rather than just occasional ones. One industry expert framed this as the organizing principle of platform work, not just a property of two features:

> The actual thing that people need is fast feedback loops... everything that you do as a platform team is built around improving that feedback loop.
>
> *Liam Mackie, Lead Cloud Engineer at Octopus Deploy*

This is consistent with the DevEx framework's finding that fast feedback loops are one of the three core dimensions of developer experience, and that slow loops raise cognitive load and disrupt developer flow (**Noda et al., 2023**). Another explanation could be that they tend to sit at the base of the delivery stack, with **Continuous Integration (CI)**, **test automation**, and **ephemeral environments** depending on fast feedback loops working well, so a platform with these features working well is likely to have the rest in order.

There is strong bi-directionality here. The advanced capabilities are out of reach until you have the foundations in place and have developers using the platform. The foundational capabilities are a marker of a maturing platform.DORA's research points in the same direction, finding that the comprehensive use of foundational capabilities, like **version control**, predicts **Continuous Delivery (CD)** and amplifies the benefits of AI adoption (**DORA, 2025**).

Feature flags carry this ratcheting effect further. A **feature flag** is a control point in a software feature that allows for a change in behavior after the software is deployed, without needling a software update. This allows for the user experience to move at the pace users are comfortable with, without forcing development teams to slow down.

When the platform matures, more sophisticated development behaviors become possible and safe (**Wahid, 2026** & **Kwaśniewski, 2024**). When sophisticated development behaviors are normalized, the platform produces a smoother, faster experience for developers. Each improvement in capabilities releases energy to be applied to the next bottleneck, problem or opportunity.

They tend to appear in advanced platforms, and this relationship likely runs both ways. **Feature flags** depend on a working deployment pipeline, which itself relies on a code delivery platform that includes robust version control, reliable behavior, strong workflow processes, and code safety features (**Wahid, 2026**). In turn, feature flags support a healthy pipeline, letting teams deploy more often without exposing in-flight features.

Feature selection should align with the platform's goals. Without clear direction, platform teams may build features that miss the crucial issues developers face, or solve real developer problems at the expense of the platform's goals, removing a slow security scan to improve throughput, say, when the platform should be improving compliance. Organizations should make the expected outcomes of the platform initiative clear, as teams need this context to make the right choices and trade-offs in their work.

Which features travel together

Next, we will look at feature co-occurrence, as this can signal the shape of a typical platform setup, including core and unique capabilities. The figure below maps the co-occurrence of features which appear on the same platform, with each line connecting a pair of features, with **builds**, **version control**, **deployment automation**, and **artifact management** all pair with one another on over 70.0% of all respondents' platforms. Feature pairs on over 60.0% of respondents' platforms are drawn, and each remaining feature's most common pair is also drawn (i.e., **ephemeral environments** most common pair is **deployment automation**).

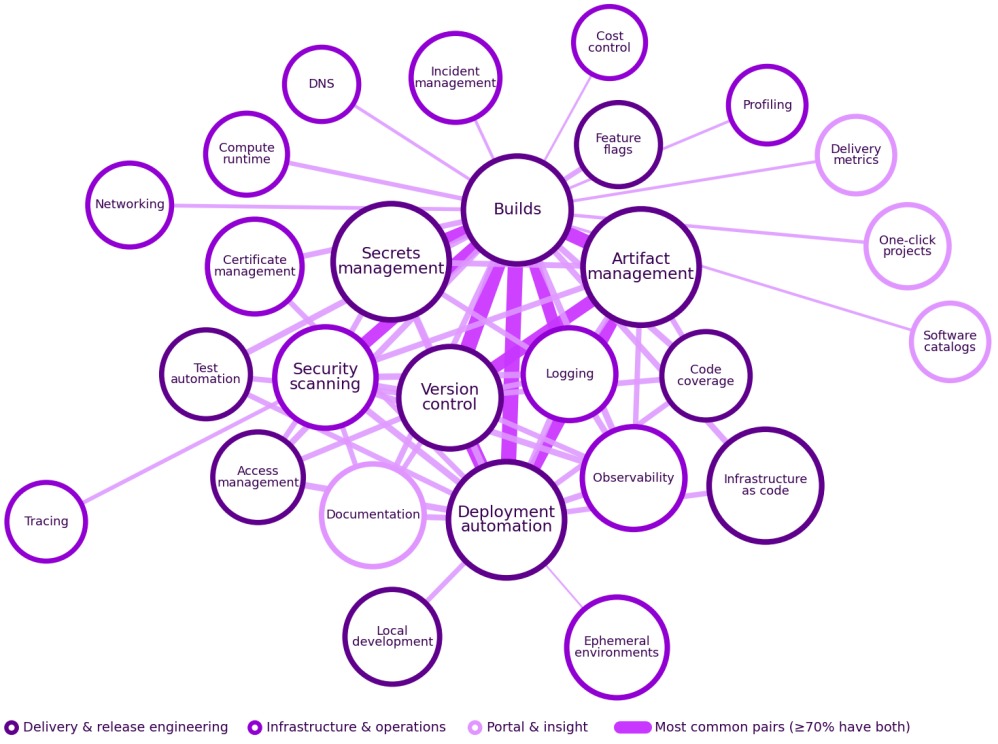


Figure 2.4: *Platform feature co-occurrence network. Lines connect features appearing on the same platform. Most common pairs (bright purple), all pair with one another on over 70.0% of platforms, pairs co-occurring on over 60.0% of platforms (multiple pairing; light purple), and each remaining feature's most common pair (single pair; light purple).*

The network centres on the fundamental software delivery pipeline: **builds**, **version control**, **deployment automation**, and **artifact management** all pair with one another with **builds** and **version control** the most common pair (79.4%). This aligns with the feature-adoption pattern, where the same pipeline features lead.

Ephemeral environments are sitting at the very bottom of the network, and were the least selected feature for standalone adoption in our survey (48.0%). These are temporary, self-service preview environments set up for pull requests and deleted after merge. The data also shows that teams with **ephemeral environments** have a much broader platform feature set (e.g., a mean of 22 features, compared to 12 features without it), and the features that co-occur are **compute runtime**, **cost control**, and **incident** and **certificate management**.

Given **ephemeral environments** clear developer-experience benefits, like reducing staging bottlenecks and catching integration bugs early, the low occurrence suggests potential rollout barriers:

- **Technical complexity:** Creating a temporary production-like environment on demand requires advanced cloud automation, container management, and complex database setup (**Piaggio, 2026**).
- **Cost:** Without strict, automated removal policies and cloud governance, they risk runaway infrastructure spend. Within the context of Continuous Delivery and with the right control mechanisms in place, **ephemeral environments** often use fewer compute hours than a single shared test environment.
- **Platform maturity:** Environments must provision and deprovision automatically and reliably, ideally using Infrastructure as Code (IaC).
- **Deployment maturity:** A fully automated, working release pipeline must be in place before adding **ephemeral environments**. Deployment processes will be used frequently with every pull request, so they must be robust. Though that frequent exercise is itself a benefit.
- **Process consistency**: You should use the same deployment process for **ephemeral environments** that you do for production, which builds confidence that your process will work when the deployment is promoted to production.

These barriers surfaced in our interviews, with one interviewee explaining:

> People aren't avoiding ephemeral environments because they don't think they're useful; they avoid them because they're really difficult to do.
>
> *Survey respondent*

This suggests that **ephemeral environments** are an **advanced milestone** in Platform Engineering teams, rather than a starting point. Organizations usually build them only after their core delivery pipeline and cloud automation are working seamlessly (**Andrews, 2025**).

Platform Features and AI impact

In the survey, we asked about AI adoption and its impact. Is there any association between a platform's features and how impact is reported? This relates to industry knowledge that AI's value depends on the system or platform on which it is being added (**DORA, 2025**). We found that 64.9% of respondents reported that AI has had a positive impact on the speed and stability of software delivery.

64.9%

reported positive AI impact on delivery speed and stability

Several features are associated with a more positive impact reported, with three standing out: **code coverage**, **ephemeral environments**, and **cost control**. Adopting these features actually shifts platforms from no AI impact to a positive AI impact on software delivery speed and stability.

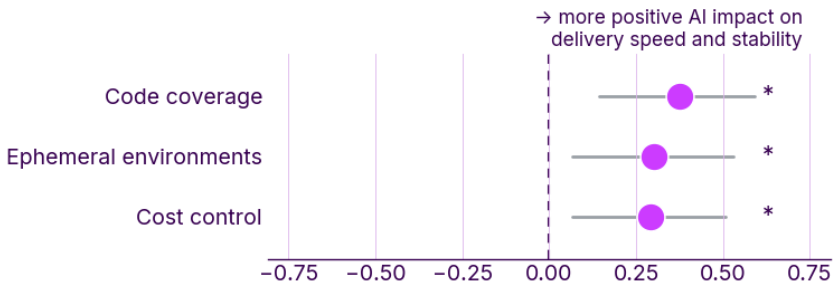


Figure 2.5: *Features ranked by AI impact on software delivery speed and stability. Rank-biserial effect of respondents who selected the feature against those who didn't, on their perceptions of AI impact. Asterisks indicate directional significance (*p<0.05), with two-sided Mann-Whitney U test.*

AI encourages developers to produce more and more quickly, but it still needs to be stable and controlled. **Code coverage**, **ephemeral environments**, and **cost control**, sit at different points in the delivery lifecycle but play similar roles in shaping the impact, stability, and speed of AI-generated code.

- ***Code coverage*** ensures that the increased volume of AI-generated code is automatically and rigorously tested, preventing potential bottlenecks in the verification pipeline.
- ***Ephemeral environments*** offer a secure, temporary space for developers to test AI-generated code, ensuring that rapid development doesn't compromise the stability of the main branch.
- ***Cost control*** provides the necessary governance to manage the surge in cloud resource consumption and build frequency that often accompanies AI-assisted development.

This pattern extends beyond software delivery, with 78.1% of respondents also reporting a positive impact of AI on organizational performance. **Code coverage** and **ephemeral environments** surface again, as well as **version control** and **certificate management**.

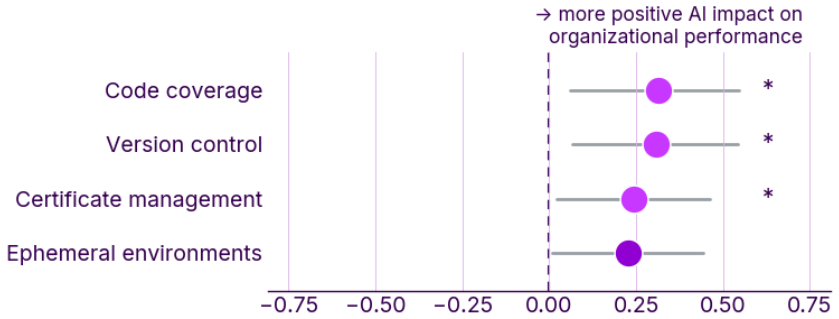


Figure 2.6: *Features ranked by AI impact on organizational performance. Rank-biserial effect of respondents who selected the feature against those who didn't, on their perceptions of AI impact. Asterisks indicate directional significance (*p<0.05), with two-sided Mann-Whitney U test.*

The two additional features read as safety measures against a growing volume of change. **Version control** keeps a history of every code change, so teams can see what AI produced and undo anything that breaks. **Certificate management** automatically renews the digital certificates that let parts of a system connect securely, rather than relying on someone remembering to. DORA names strong version control among the capabilities that amplify AI's benefits, a critical safety net as AI increases the volume and velocity of change (**DORA, 2025**).

This may indicate AI is being added to more complete, well-built systems, resonating with DORA's central finding that AI amplifies rather than solves: it magnifies strengths while exposing weaknesses. The report also stresses that quality internal platforms enable AI's positive impact, the better built the system, the more AI has to work with (**DORA, 2025**).

Build or buy: how teams decide

To understand how platform choices impact strategic outcomes, we asked respondents how the platform team supports each feature, whether it's **built from scratch**, **built on existing templates**, or **uses an underlying tool**. Across most features, **using an underlying tool** is the primary implementation approach, with 51.2% of respondents reporting this method, compared to **building from scratch** (31.2%) or **building using an existing template** (17.5%).

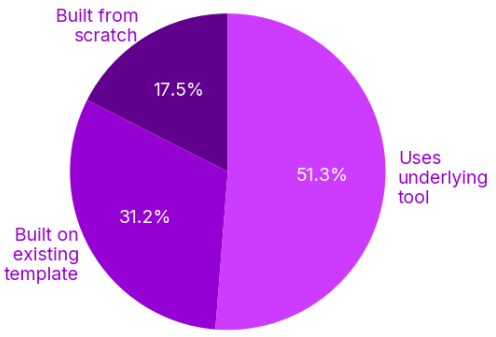


Figure 2.7: *Build-versus-buy feature implementation approach.*

High-performers are more likely to assemble platforms using **underlying tools** (58.8%) rather than **build on existing templates** (47.8%) or **from scratch** (33.3%). These point to a broader industry shift: as core architectural patterns settle and technology choices consolidate, platform teams are standardizing on underlying tools rather than building their own. **Tuite (2025)** describes this as the "build-versus-buy" trap in platform maturity, where teams maintaining custom interfaces spend 6-18 months upgrades and operational overhead rather than delivering value.

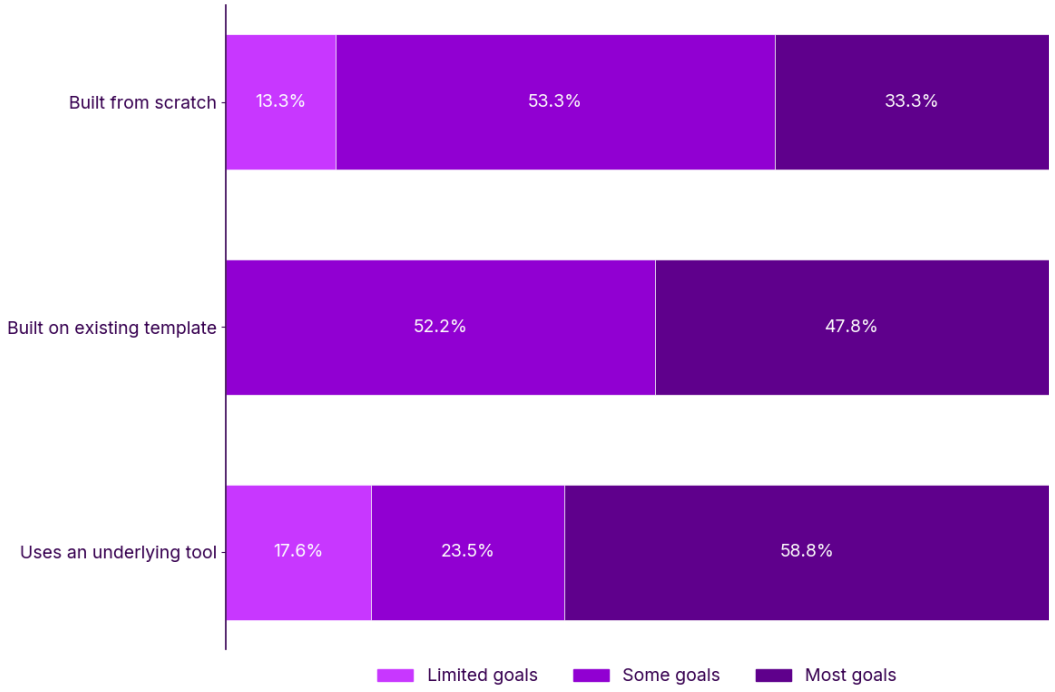


Figure 2.8: *Build-versus-buy feature implementation approach by goals delivered. Shown as percentages, respondents' dominant build method by goal delivery outcome across the cohort.*

The likely explanation is where each approach directs a team's effort. **Building from scratch** redirects engineering time to problems that have already been solved by an external tool. That reduces the time spent on the work that is unique to the organization, which only the platform team can do. By selecting appropriate tools, platform teams accelerate the delivery of value, provide a higher quality offering, and reduce the maintenance burden. They can then spend more time tailoring and tuning the tool chain to the organization's specific needs.

The exceptions: where teams don't reach for a tool

A few features break this pattern. **Local development**, **one-click projects**, and **ephemeral environments** were found to be most often built on existing templates, reflecting how tightly these features are bound to an organization's specific operating model: **one-click projects** encode an organization's "golden paths", and **local development** and **ephemeral environments** have to match its specific tech stack.

Given that both **local development** and **ephemeral environments** were previously associated with advanced or high-performing users, it is interesting that they are the features more commonly built on existing templates rather than on a tool. This pattern may reflect technical necessity rather than strategic avoidance of standard tooling.

Supporting these features well depends on being able to define reusable templates, reference them across projects, and manage template versions as they evolve, which few off-the-shelf tools currently provide. Where tooling cannot satisfy an organization's proprietary build systems, legacy dependencies, and bespoke delivery pipelines, teams assemble template-based solutions themselves. Consequently, teams seem to prioritize custom-built or template-based solutions not to avoid "buy" options, but to address specific functional requirements.

Why teams build, why teams buy

We asked for the reasoning behind build-or-buy decisions, as it could reveal how teams are operating beyond the choice itself. We found that the most common categories of responses here were **cost-benefit analysis** (29.6%) and **prefer open-source** (28.6%). Other common reasons were **fit/requirement assessment**, **prefer/commercial/SaaS** (both 22.4%), and **built for core/differentiation** (16.3%).

The open-text responses here describe these decisions as exercises in resource allocation, and they reveal teams balancing multiple considerations simultaneously. The first and most common is the economic efficiency of building, which is generally a straightforward weighing of development and maintenance effort, though this is subject to Hofstadter's law (it always takes longer than you expect, even when you take into account Hofstadter's law) (**Milanović, 2026**).

> Build what makes us unique, buy or adopt what's standard, and always consider long-term cost and lock-in.
>
> *Survey respondent*

The second is whether the feature or capability aligns with the ecosystem, not whether to buy, but what kind of dependency to take on. And the third turns decisions about cost into a strategy, in that the team views it as either a commodity that can be bought or a differentiator that warrants custom engineering.

> I start by assessing whether the feature is a core differentiator for our platform. If it provides a competitive advantage, we consider building it; otherwise, we prefer open-source or commercial tools.
>
> *Survey respondent*

Ultimately, these considerations show that teams aren't following a rigid policy, but rather on whether the feature or capability is a standard commodity to be bought or a unique differentiator to be built. The most considered responses here combine all three considerations: buying what is standard, controlling costs and lock-in, and reserving custom engineering for what is genuinely unique to the business.

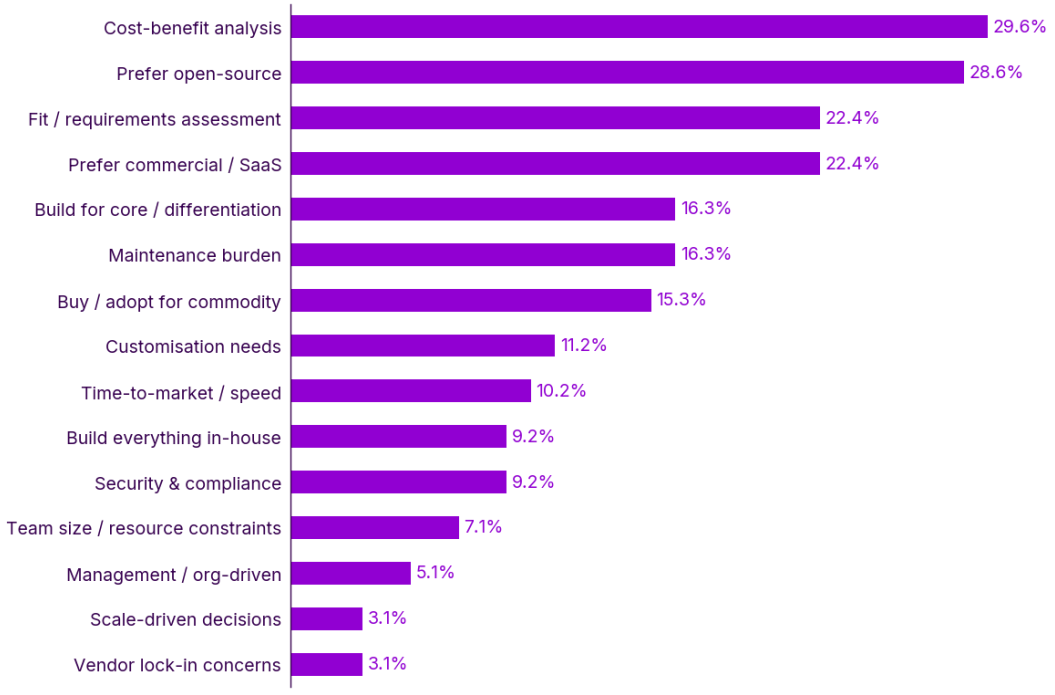


Figure 2.9: *Themes categorized in response to "Explain your approach to deciding what to build versus when to use an open-source or commercial tool to offer platform features." (open-text question). Shown as a percentage of respondents who reported each category.*

Summary

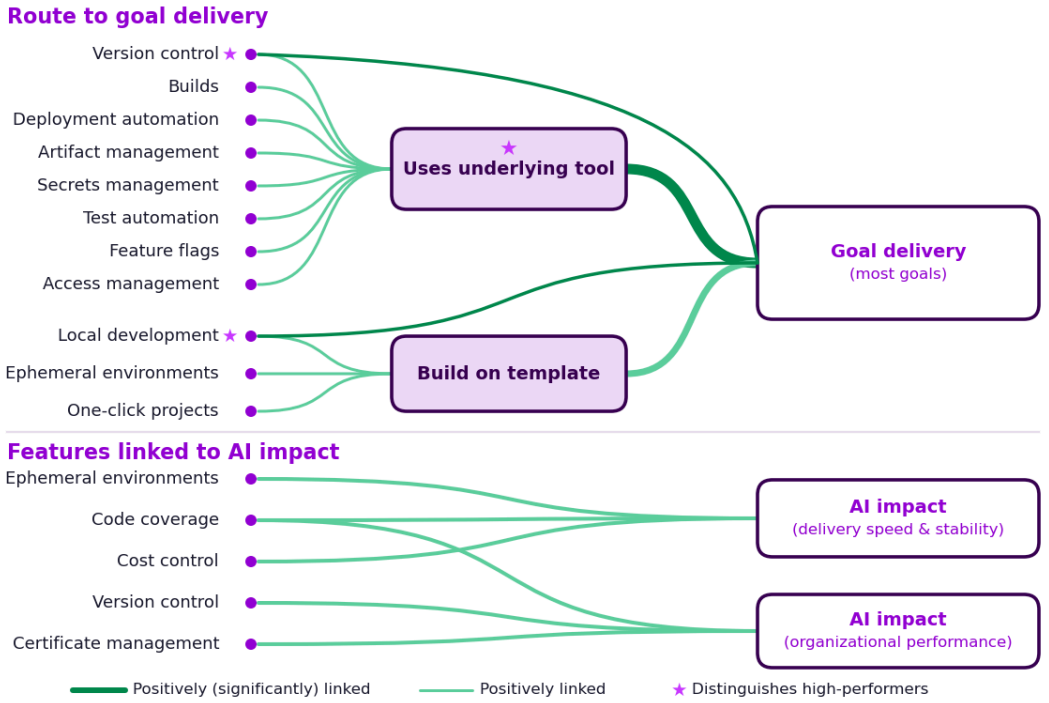


Figure 2.10: *Section 2 Summary.*

SECTION 3

Obstacles preventing platform success

What stands in the way of platform success? This section explores the barriers that prevent platform teams from achieving success and analyze how these obstacles link to other measurable outcomes. One pattern is clear: strategic and organizational obstacles are more strongly associated with reduced goal delivery than technical obstacles.

Key takeaways

1. Obstacles are organizational, not technical. Competing priorities and budget constrain teams most, but how often an obstacle is reported isn't how much it holds a team back.
2. A lack of a clear strategy is the most damaging obstacle, linked to low leadership perception and low job satisfaction.
3. Building from scratch is linked with a lack of a clear strategy.
4. Burnout tracks with tooling sprawl, resistance to change, and a lack of skills.

The obstacles that every platform team faces

We begin by examining the most common barriers Platform Engineering teams face and find that **competing priorities** are the universal obstacle (54.9%). It is followed by a **lack of budget** (39.2%), **technical complexity** (33.3%), **resistance to change** (27.5%), **lack of clear strategy** (21.6%), **lack of skills**, and **tooling sprawl** (both 19.6%).

Based on frequency alone, respondents report being most constrained by competing priorities and resource limitations across the priorities here, rather than by technical hurdles. We also tested a hypothesis in our study: that the number of obstacles a team reported would be associated with goal delivery; however, we found no association in our survey data. It is not the count of obstacles that matters, but which ones.

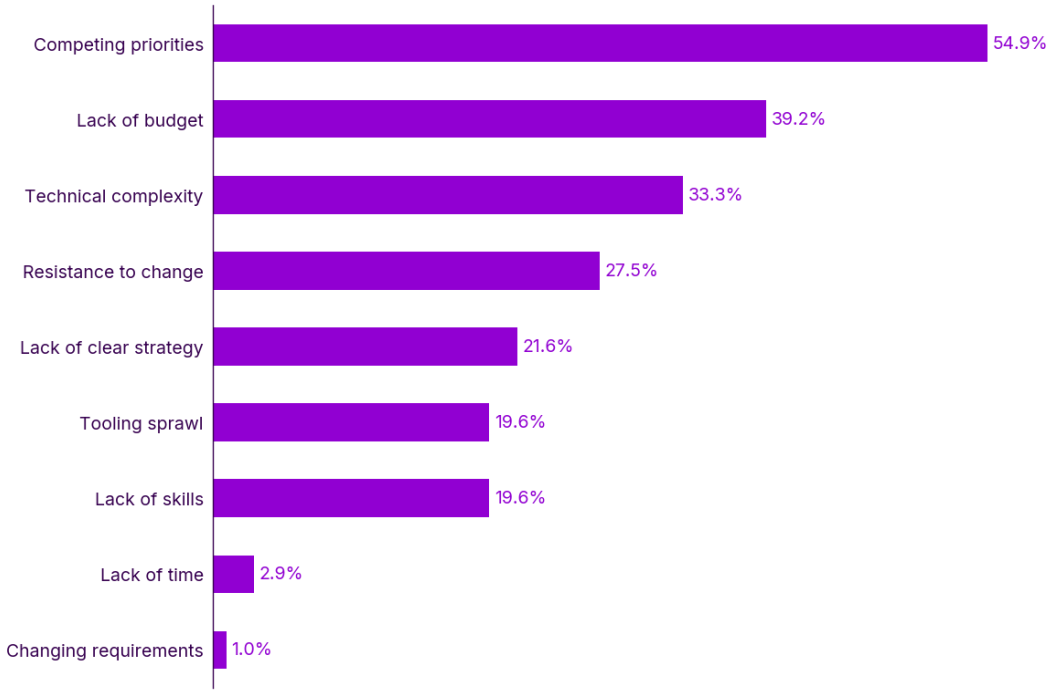


Figure 3.1: *The most common obstacles. "What are the most significant obstacles preventing your organization from fully realizing the benefits of the Platform Engineering initiative? (multi-select)."*

The obstacles that matter most

A more valuable perspective on obstacles is their impact on the platform's success. This is particularly crucial in Platform Engineering, as some obstacles are the reason for a platform team's existence, while others get in the way of them achieving their goals. An obstacle that stands between a platform team and its goals should be addressed as a high priority.

The **DORA 2025 State of AI-assisted Software Development report** and **"Thinking in Platforms"** both reinforce that developer friction, strategic clarity, and platform adoption are systemic properties rather than individual technical ones. Therefore, capturing a team's perceptions of these barriers is an essential way to uncover the underlying operational strategies that ultimately define platform performance.

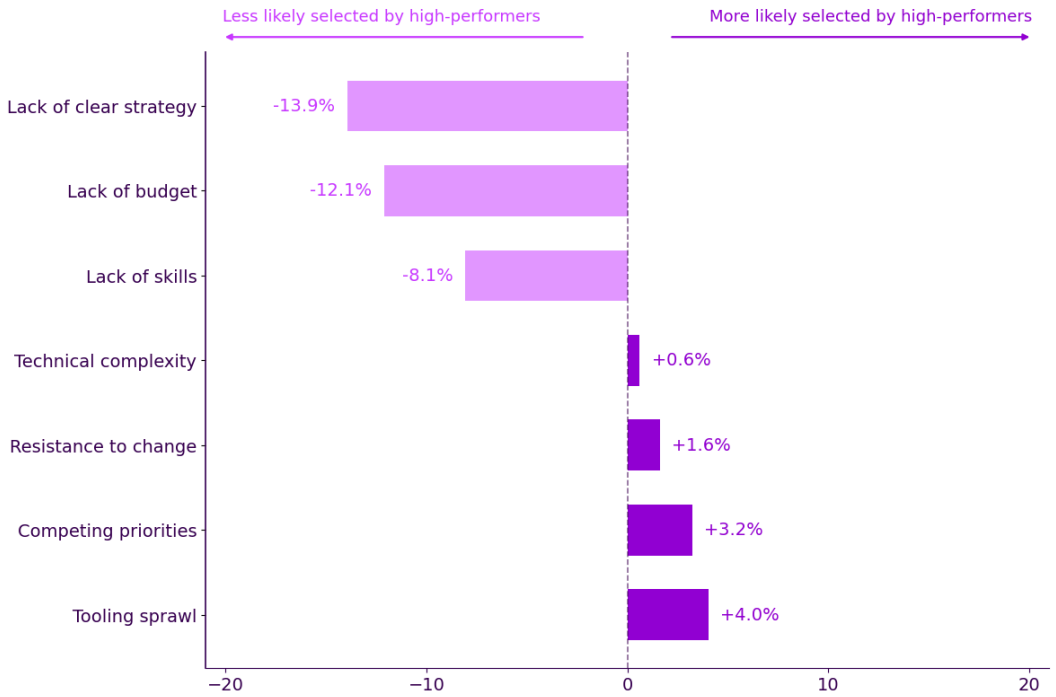


Figure 3.2: *Percentage gap between high-performers and the rest of cohort on reported obstacles. Tested using Fisher's exact test (positive percentages indicate obstacles selected more often by high-performers).*

We analyzed the obstacles in relation to respondents' reports of achieving more goals. We found that a **lack of a clear strategy** has the weakest association with high-performers (-13.9%), followed by a **lack of budget** (-12.1%) and a **lack of skills** (-8.1%). This is consistent with broader findings that identify budget constraints and skill gaps as challenges for platforms, as well as workflow integration and security risks even for advanced platform teams (**Red Hat, 2024**).

All other obstacles sit relatively close to neutral, indicating a low and consistent level of impact among respondents. Notably, **tooling sprawl**, often reported as an industry-wide barrier that wears teams down (**DiGirolamo, 2026**), shows no such pattern here. High-performers report it, similarly to all other respondents, indicating it is less decisive for goal delivery than strategy, budget, or skills. The data suggest that the barriers that most distinguish lower-performing teams are strategic and resource-related, not tooling-related.

Lack of clear strategy: the obstacle that separates teams

Of all the obstacles, a **lack of clear strategy** is associated with the least favorable outcomes. Respondents who selected it rate their leadership lower on having a clear sense of direction (i.e., **leadership direction**; "my organization's leadership understands where the organization is going and where we want to be"), on challenging them to rethink assumptions (i.e., **leadership thinking**; "my organization's leadership challenges team members to think about problems in new ways and to rethink some of their basic assumptions about their work"), and report lower job satisfaction. The consistency in scoring low on leadership, **job satisfaction**, and goal delivery points more to a direction setting problem rather than day-to-day delivery.

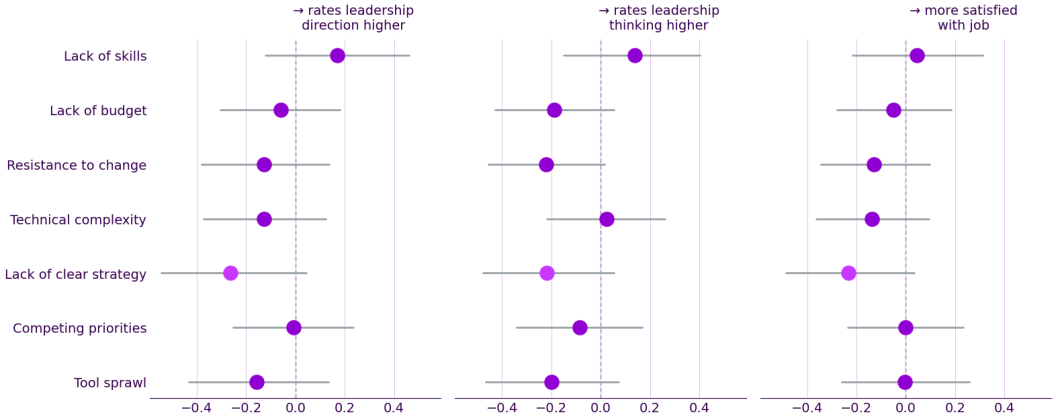


Figure 3.3: *Obstacles reported and ranked by perceptions of leadership direction, leadership thinking, and job satisfaction. Rank-biserial effect comparing percentages of respondents who selected obstacles against those who didn't, by leadership direction, leadership thinking, and job satisfaction.*

Without a strategy, teams may work without a clear picture of their goals, which can make leadership seem disengaged or ineffective, even if that's not the reality (**Santos Paulo, 2026**). **It also shifts delivery from building something meaningful to simply completing tasks, which wears down morale over time.** Lower job satisfaction in these environments follows from that missing long-term goal: without a clear "why" behind the platform, developers can't connect their work to organizational success (**Association of Chamber of Commerce Executives, 2021** & **Maslach, 2024**).

This pattern is sharpest among the people building the platform. Platform Engineers report a **lack of a clear strategy** more than any other job role, at 44.4%. Two mechanisms may be at work here:

The first being directional: platform engineers need a clear direction to work towards more than most roles do, so they feel its absence most, while the executives who set the direction feel it least.

The second is an implementation or change cost: strategy and tooling changes are decided by one group but implemented by another, and its Platform Engineers who absorb the rework each shift demands.

Only 21.6% of respondents chose this as an obstacle, yet the gap between its frequency and its strong association with negative outcomes is substantial and carries more weight. Industry research reinforces these findings, treating a strategy-and-measurement gap as a key barrier to platform success (**Haigh, 2026**). This isn't a technical execution problem, but can be viewed as a signal that organizations should have a closer look at strategic alignment.

Built from scratch approach and the strategy gap

Revisiting the build-versus-buy analysis from **Section 2**, against obstacles surfaces a notably strong association. Looking at the heatmap below, respondents who predominantly build features from scratch are significantly more likely to report a **lack of clear strategy** as an obstacle (57.9%; **p=0.0006), close to 4 times the rate of respondents who report **building on existing templates** (15.6%) or **using underlying tools** for features or capabilities (16.7%).

Wardley's (2015) strategic frameworks highlight that teams with low strategic clarity often waste effort on custom-building solutions to problems already solved by off-the-shelf tooling because they lack a clear view of the technical landscape. This matches what one expert told us:

> Engineers often build a thing because by building it, they'll work out the problem. It's a problem-solving strategy.
>
> *Tim Nicholas, Lead Automation Architect and Product Owner at Octopus Deploy*

Building from scratch, then, may be less a consequence of absent strategy than an unacknowledged substitute for it, which may explain why the two are reported together often.

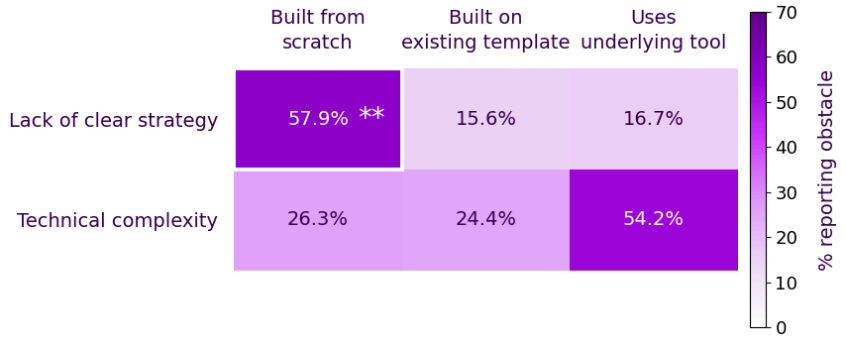


Figure 3.4: *Build-versus-buy feature implementation approach by lack of clear strategy and technical complexity. Heatmap showing the percentage of respondents reporting each obstacle, split by dominant build method. Asterisks indicate significance, with Bonferroni correction applied (**p<0.0071).*

Interestingly, **technical complexity** points in the opposite direction. It is the most commonly cited obstacle by respondents who **use underlying tools** (54.2%), compared with 26.3% of those **building from scratch** and 24.4% of those **building on existing templates**. This is less of a contradiction than it seems. **An external tool doesn't remove complexity so much as move it (i.e., from the work of building to the work of integrating, configuring, and maintaining it)**. Where building from scratch trades complexity for effort and time, using a tool trades it for integration overhead. This may mean that neither approach escapes the cost but instead shifts it elsewhere (**DORA, 2022**).

This may indicate that technical complexity is the reason for a platform's existence. While unnecessary complexity should be avoided, a platform must tackle the inherent complexity in software delivery to provide a meaningful offering to developers.

The obstacles that track with lower well-being

Three obstacles stand out for their association with reduced well-being (i.e., more burnout): **tooling sprawl**, **resistance to change**, and **lack of skills**. Notably, they are the only 3 obstacles associated with reduced well-being, with the remaining obstacles showing flat or slightly positive associations.

It is unsurprising that **tooling sprawl** tracks with lower well-being, with industry evidence pointing in the same direction: developers lose several hours per week to tool switching, and reorienting to other tooling increases cognitive fatigue and drives burnout (**Wolfe, 2025**, **Millan, 2025** & **Bedard et al., 2026**).

There is a tension here worth naming: **tooling sprawl** is not a simple problem to remove. Standardizing with a single tool can be worse than dealing with the necessary sprawl of choosing the best tools for the job; a generalized all-in-one tool may reduce the platform team's own effort and complexity while increasing the friction experienced by developers using the platform.



Figure 3.5: *Obstacles reported and ranked by well-being (i.e., inverted burnout). Rank-biserial effect comparing respondents who selected obstacles against those who didn't, on their self-reported well-being.*

The aim of tool consolidation is not to have the fewest tools, but to have a shared set that minimizes onboarding communication friction that teams may face when working across different tools that do the same thing. In this view, the association between lower well-being (i.e., burnout) and **tooling sprawl** could also reflect deliberate tool selection rather than aggressive consolidation (**DORA, 2024**).

Resistance to change is the other broad-impact obstacle, associated with lower well-being and a lower perception of leadership thinking (figure 3.3). Both point to resistance being as much a cultural and organizational signal as a technical one. This reinforces the need to address cultural and strategic blockers directly, rather than thinking technology will compensate for them.

Who reports **resistance to change** as an obstacle itself is informative. Developers report resistance to change at the highest rate of any other role (53.8%). This makes sense, as developers are platform end users, so adoption decisions are made for them rather than by them. Broader-industry literature explains why this matters. When platforms are adopted from the top down, developer satisfaction is lower than when teams adopt them because the platforms are genuinely useful. A high adoption rate can mask problems, since it may simply reflect the mandate rather than real value (**Tekkesinoglu et al., 2026**). Developers sit on exactly that side of the gap. We explore mandatory adoption in the next section.

Lack of skills points in a similar direction and could be explained by skill gaps increasing the effort required for otherwise routine work, so tasks that should be straightforward become draining.

Platform teams are held back less by the number of obstacles they face than by which ones: a missing strategy, constrained budgets, and skill gaps separate lower-performing teams, while tooling sprawl, resistance to change, and a lack of skills wear down the people doing the work.

Summary

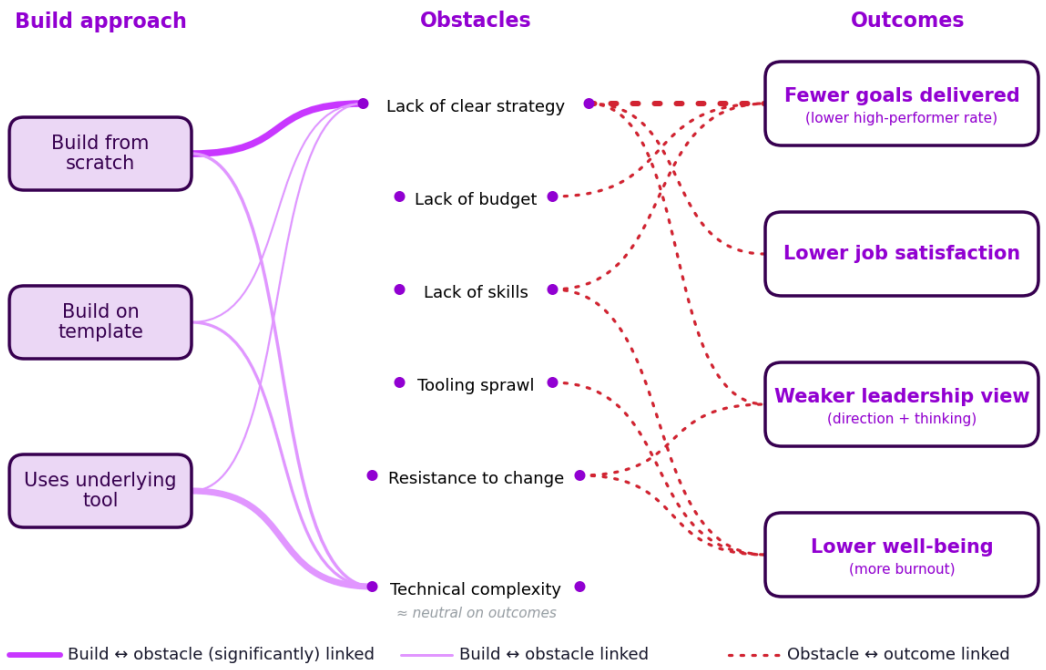


Figure 3.6: *Section 3 Summary.*

SECTION 4

What high-performing teams do differently

This section looks at what separates high-performing platform initiatives (respondents who reported achieving **most goals**) from the rest of the cohort. Comparing this cohort against teams reporting to achieve limited goals, we can see frameworks, adoption strategies, and feedback loops associated with successful outcomes.

This section also tests a set of hypotheses about what separates high-performers, drawn from industry research, and our expectations. Some were supported, and some were overturned. Each is flagged against the relevant findings below, with all five summarized at the end of the section.

Key takeaways

1. Platforms running for 3+ years are much more likely to meet their goals.
2. Mandatory adoption is significantly associated with high-performers compared to optional adoption.
3. The producer-consumer gap has effectively closed under mandatory adoption, suggesting a maturing discipline.
4. Measuring developer satisfaction is more closely associated with high-performing teams.
5. Weekly feedback, gathered in-workflow (code reviews and PRs) is the sweet spot for effective feedback cadence and channels.

Platform maturity follows an S-curve

Platform Engineering initiatives rarely deliver immediate, linear results. Building developer platforms and standard ways to deploy software takes upfront commitment and requires teams to change how they work.

Our findings show a positive correlation between operational platform age and the likelihood of achieving strategic organizational goals.

- **Mature platforms (+ 3 years)**: 82.0% of platforms that have operated for more than 3 years are high-performers.
- **Early stage platforms (less than 3 years):** Only 33.3% of platforms operating for less than 3 years are high-performers.

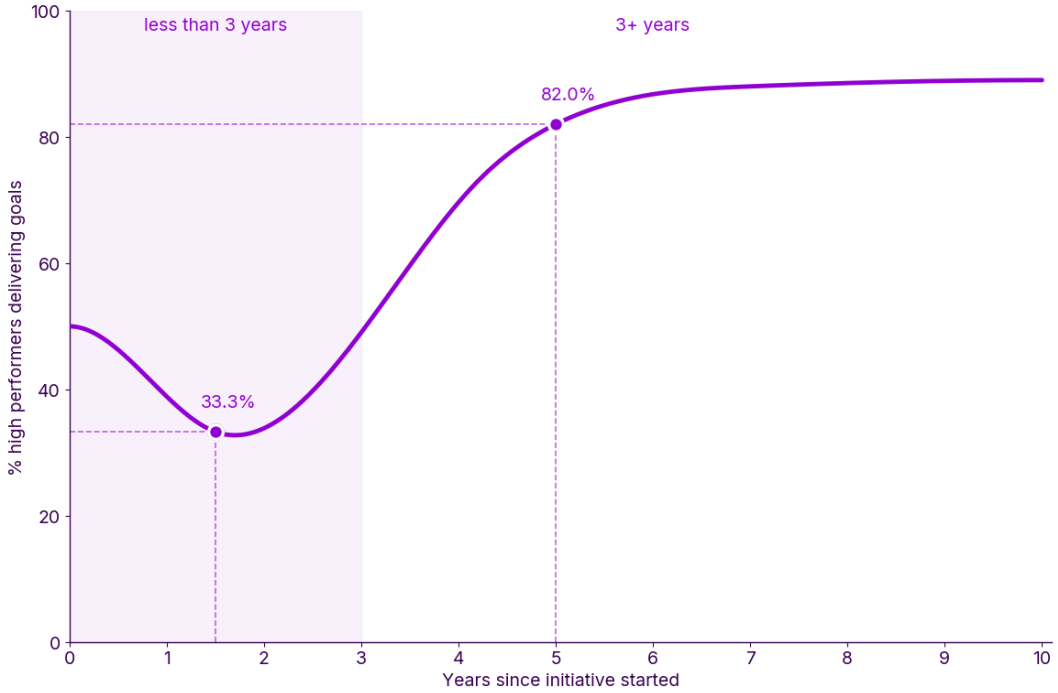


Figure 4.1: *Platform Engineering percentage of goal delivery for high-performers following a S-curve over time from early-stage operation (less than 3 years) to later-stage operation (3+ years).*

Mature platforms are 2.4 times more likely to be high-performers than newer platform initiatives. This aligns with our previous report findings (**Octopus Deploy, 2025**) and the broader industry literature, which show that in the initial 12 to 24 months of platform initiatives, there is setup friction, legacy migrations, and developer adjustment curves (**DORA, 2024**).

This highlights that once organizations get past the initial adoption curve, the value of Platform Engineering compounds over time, and that sustained effort is one of the most significant signals of success.

Adoption strategy: mandatory adoption tracks with goal delivery

One of the most persistent and controversial debates in Platform Engineering is whether platform adoption should be voluntary (optional) or enforced (mandatory). Leading industry frameworks and advice, like the **CNCF Platform Engineering maturity model** and **Team Topologies**, say that internal developer platforms should be treated like products, winning over developers organically rather than forcing them to use them (**Skelton & Pais, n.d.,** & **Cloud Native Computing Foundation, 2025**). We expected our data to align with this, with optional adoption more strongly associated with high-performers.

In the **2025 Platform Engineering Pulse report** by Octopus Deploy, we found that 63% of platforms were mandated and just 37% were optional. This year, the balance has tipped the other way, with optional platforms now in the majority (54.1%), compared to mandatory platforms (45.9%). This suggests that optional platforms are on the rise and aligns with the direction recommended by industry frameworks, which is to treat the platform as a product developers choose to adopt rather than one they're required to use (**Bottcher, 2018**, **Skelton & Pais, n.d.,** & **Cloud Native Computing Foundation, 2025**).

There is a direction of travel towards the optional; however, the performance-based findings strongly contradict our assumption. We found that 62.2% of respondents reporting mandatory platform adoption are high-performers, compared to 27.5% of teams reporting optional platform use. This is a significant finding, showing that mandatory adoption is associated with a substantially higher rate of high-performers than optional adoption (*p=0.0009).

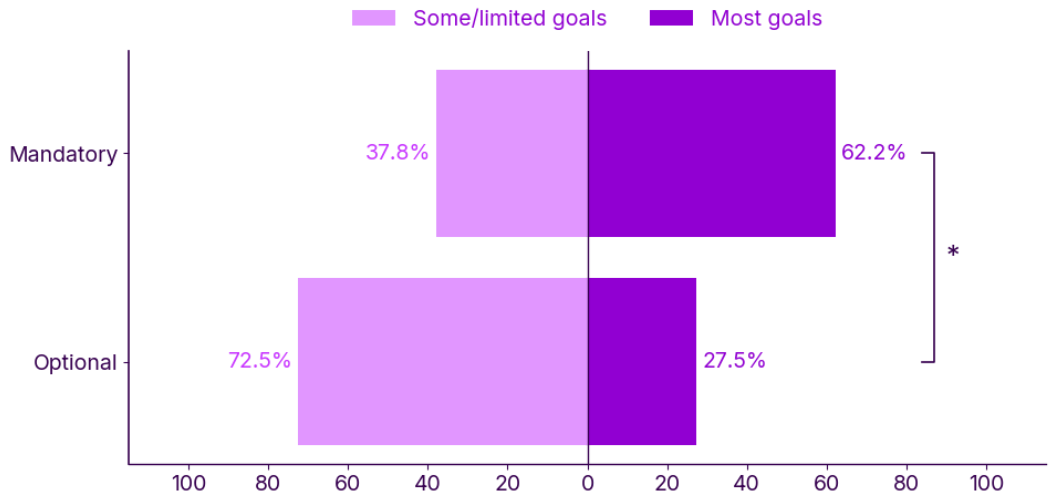


Figure 4.2: *Goals delivered by adoption strategy. Bars are normalized within each adoption strategy, and are split by goal outcome. Asterisks indicate significance, Fisher's exact test (*p<0.0009).*

This difference between mandatory and optional platforms stems from real-world friction points that product-led platform adoption may overlook:

1. **Decision fatigue and developer cognitive load:** Optional platforms encourage autonomy, but allowing developers to choose their own tools could also lead to decision fatigue and tool sprawl. Mandatory platforms remove this cognitive overhead by setting non-negotiable defaults that standardize maintenance and allow engineering teams to focus entirely on delivering value (**Skelton & Pais, n.d., & DORA, n.d.**).
2. **Getting value from your platform investment:** Building developer tools takes significant commitment, and if adoption is optional, user numbers may not indicate a clear return on investment (ROI). Requiring everyone to use the platform creates immediate scale and simplifies security updates, compliance, and cost control across the board.
3. **Enabling the platform:** Mandatory adoption can actually help developers by handling complex security checks, compliance, and governance standards, and by automating server setup, making the easiest path for developers the right one for the company.
4. **AI-assisted software development**: Developers are spending more time learning and applying techniques to embed coding assistants and agents into their workflows, so they have less time and attention to dedicate to the platform adoption decision.

This isn't really a case of mandatory "winning" and optional "losing", the two are answering different questions. Mandatory adoption is the strongest predictor of **goal delivery** in our data, which is the performance question. Optional adoption speaks to whether a platform earns its place because developers choose it, not because they're told to. A platform can score well on either, and the strongest are moving toward both, mandated enough to guarantee scale and consistency, and good enough that developers would opt in anyway.

A potential third way, which is that the organization's policies should be mandatory, and platform adoption should be optional. A platform that provides the simplest path to meeting policies will be popular with development teams and provide meaningful benefits to platform sponsors. Where a platform is mandatory, measuring and managing developer satisfaction is even more crucial as adoption no longer signals platform success.

Closing the producer-consumer perception gap

To investigate whether adoption creates friction for end users, we assessed the adoption strategy of high-performers against platform personas. For this analysis, we specifically examined the perspectives of platform Producers and Consumers.

In our previous report (**Octopus Deploy, 2025**), we found a significant perception gap: platform producers rated mandatory platforms far higher than platform consumers (developers), as end users frequently felt constrained by rigid, top-down tooling. In this year's data, this gap has effectively closed:

- **Producers, reporting mandatory adoption:** 65.0% report being high-performers (compared to 19.1% under optional adoption for producers).
- **Consumers reporting mandatory adoption:** 62.5% report being high-performers (compared to 40.0% under optional adoption).

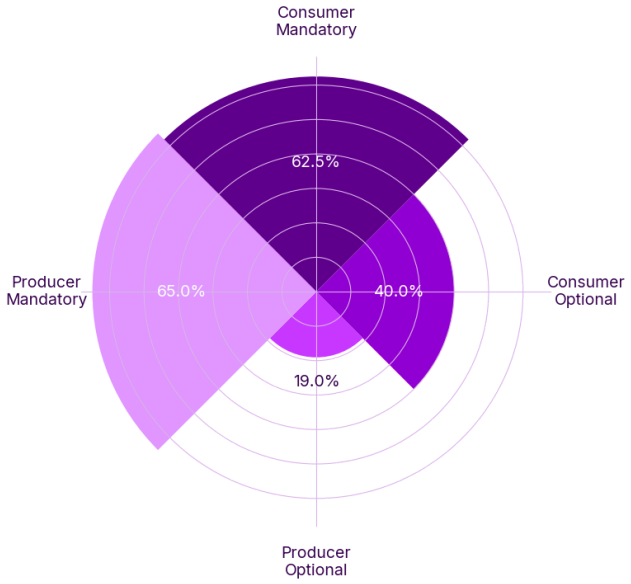


Figure 4.3: *High-performing teams by adoption strategy and platform persona. Each segment shows the proportion of high-performers within each group (normalized). A larger segment means a higher proportion of that group met their goals.*

This is almost complete alignment for mandatory adoption across both producers and consumers, demonstrating that Platform Engineering as a discipline is maturing. When mandatory platforms are well designed, developers stop perceiving them as administrative burdens and instead see them as seamless tools that simplify security, streamline CI/CD workflows, and handle infrastructure deployment.

Under optional adoption, consumers are more than twice as likely as producers to be high-performers (40.0% vs. 19.1%, respectively), which states that the teams using the platform derive more value from optional adoption than the teams building it. This is consistent with the pattern we found in our previous report, where more consumers rated optional platforms more successful than producers did. This is an important finding, and suggests that where developers retain choice, end users report the strongest outcomes, which is a sustainability signal, not a contradiction of the performance findings.

Measuring developer experience

Across the industry, measuring developer satisfaction has become an established marker of a successful platform. It's one of the most consistently corroborated findings in developer-productivity research, from the **SPACE framework** to **DevEx** to **DORA** (**Forsgren et al., 2021**, **Noda et al., 2023** & **DORA, n.d.**). An assumption we made, was that teams treating the platform as a product, and measuring its users' satisfaction, would align with more goal delivery, and our findings confirmed this. Among teams that measure developer satisfaction, 50.7% are high-performers, compared with 39.1% among those that don't measure it (figure 4.4). There is little difference in the middle, however widens at the lower end, with 13.4% of respondents measuring developer satisfaction falling into a limited goals cohort, compared with 26.1% of non-measuring teams, which is about half the incidence of under-performing teams.

Two caveats come with these results. The first is that they don't tell us how often satisfaction was measured, or what was captured. The second is that 39.1% of teams not measuring developer satisfaction still reported delivering most goals. This suggests they may have other ways of staying close to their developers, or performance may simply look healthier from the inside without that signal. Measuring developer satisfaction may signal the kind of feedback-oriented culture that tends to produce it. A platform can meet all its technical targets and still under-serve people who use it.

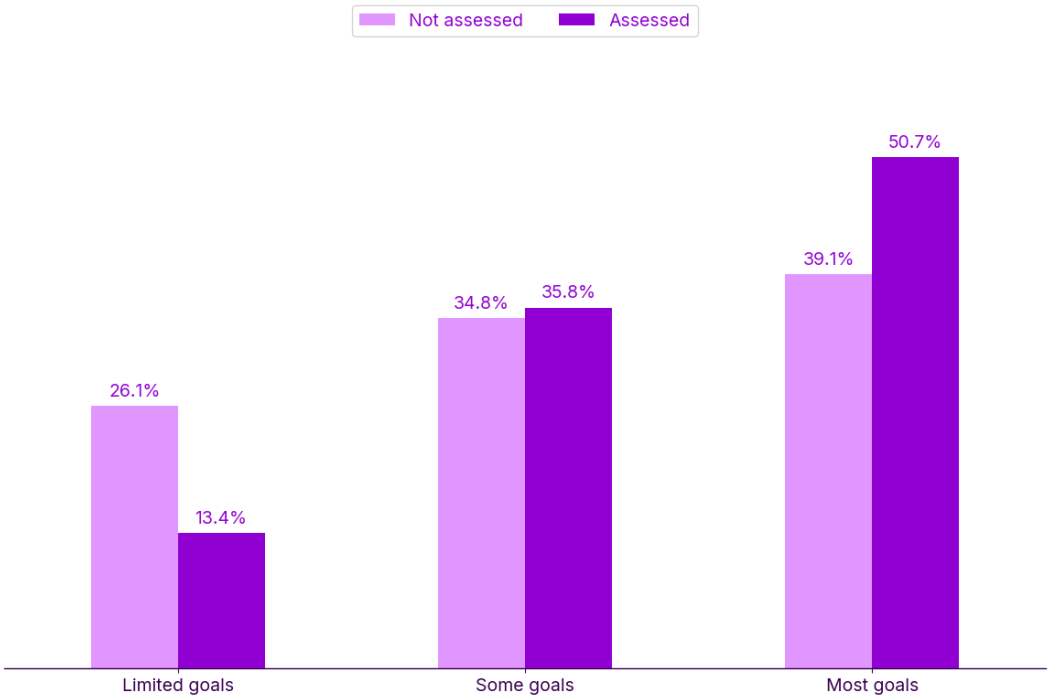


Figure 4.4: *Goals delivered by whether developer satisfaction is assessed. Bars are normalized to each group (assesses developer satisfaction or not), so each adds to 100% across goal outcomes.*

The feedback loop: how often and where

How you give and receive feedback determines whether you drive meaningful change or create unnecessary friction, and more frequent feedback means less time blocked and quicker resolutions, so we expected more frequent feedback to be more aligned with high-performing teams. We found that both the frequency and the channel of feedback are associated with goal delivery, and that some frequencies work better than others.

Feedback cadence

Comparing how often teams provide and receive feedback, measured against goal delivery, revealed a clear operational "sweet spot". **Weekly** feedback frequency is the most effective cadence, with 53.3% of respondents reporting being high-performers. This is consistent with agile sprints, which give platform engineers sufficient time to review feedback, build fixes, and release updates without constantly interrupting application developers (**Beck et al., 2001**).

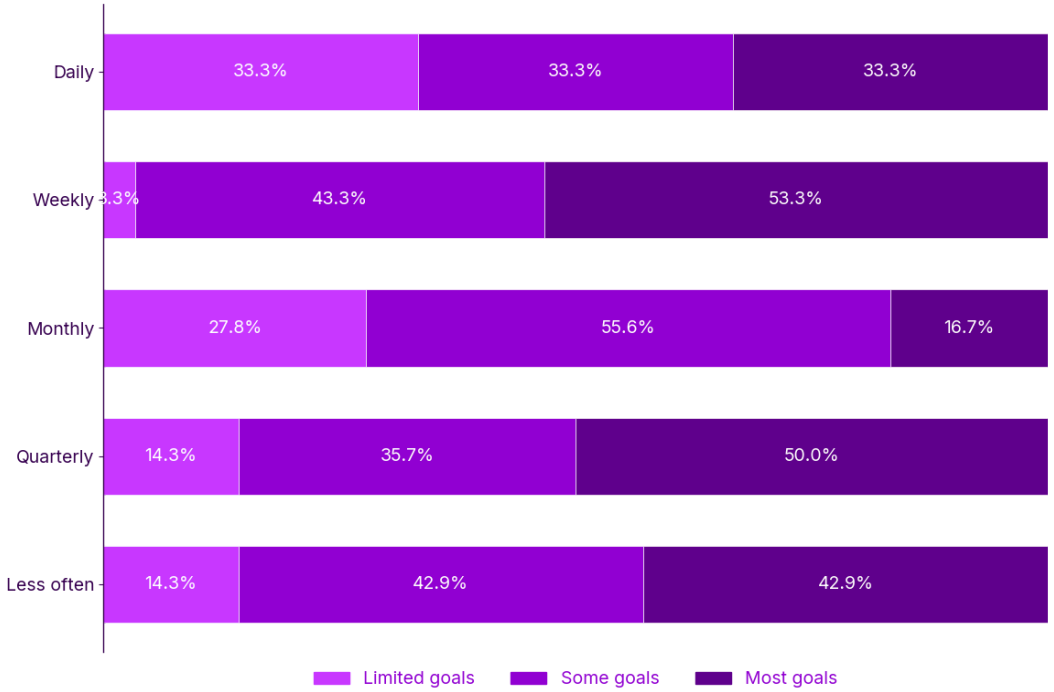


Figure 4.5: *Feedback frequency by goals delivered. Shown as a stacked bar chart for feedback frequency, with each frequency shown as a percentage of respondents goals outcomes.*

Increasing the frequency to **daily** reduces the share of high-performers to 33.3% here and could indicate that constant communication can devolve into micro-requests and quick fixes, turning platforms into help desks rather than strategic teams. In comparison, feedback that is too far apart has a negative impact, with **monthly** feedback yielding the lowest success rate: only 16.7% of high-performers report this cadence. This could be caused by platform updates being delayed beyond current developer needs, and encourages the use of workarounds.

Communication channels: where you ask matters

Alongside the timing of feedback, the way it is given and received significantly impacts its effectiveness. Gathering feedback directly within existing developer workflows, specifically through **code reviews and pull requests**, is the feedback type most strongly associated with high-performers. By collecting input directly within developers' repositories, platform engineers receive accurate, trackable notes without forcing developers to switch tasks.

Surveys, feedback forms, or **tracking systems** rank second among the most effective feedback methods. These structured tools capture developer needs into tasks, even if they are not immediately reviewed, like the real-time context of code reviews. All other feedback types show no obvious difference, suggesting a consistent level of impact across the entire cohort.

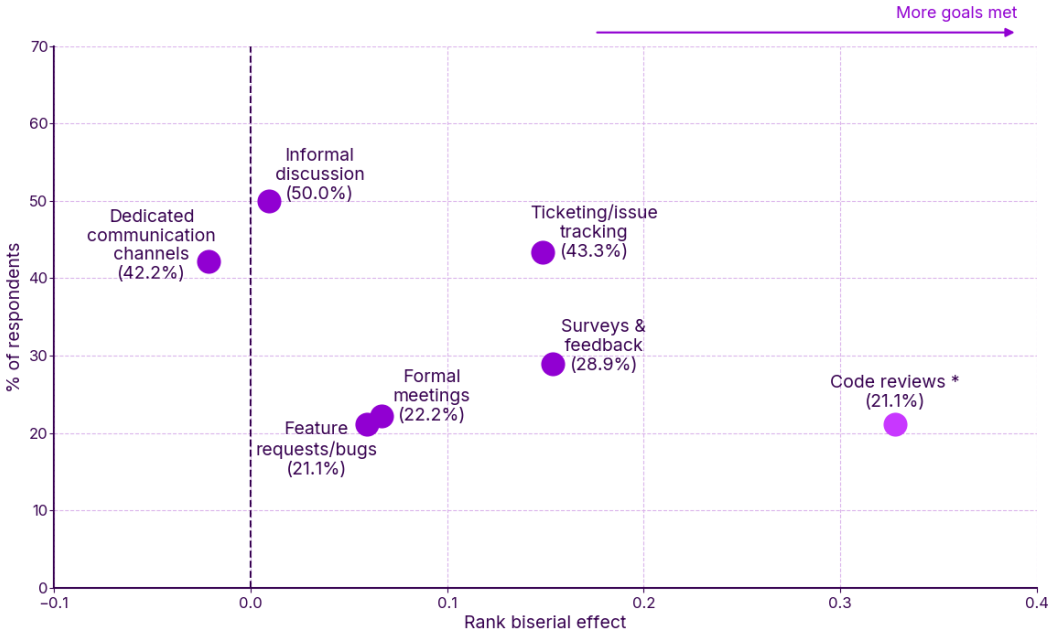


Figure 4.6: *Feedback type reported and ranked by goal delivery. Rank-biserial correlation between respondents who selected a given feedback method and those that didn't, against goals delivered. Asterisks indicates directional significance (*p<0.05), with two-sided Mann-Whitney U test.*

The most revealing findings are what sit at the bottom. Two of the channels teams use most are not the ones that work. **Dedicated communication channels (i.e., Slack or Teams)** are the least effective method, yet 42.2% of respondents rely on them, making them one of the most popular choices. **Informal discussion** is the most common (50.0%), and the same disconnect: common, but with almost no association with goal delivery.

The pattern across every channel points in the same direction; feedback only drives change when it's captured somewhere it can be tracked and used. Anything that stays in a chat thread or a conversation tends to fade before it's actioned.

Budget confidence

Demonstrating clear and measurable value is the best way to protect Platform Engineering budgets. It's no surprise that goal completion directly aligns with long-term confidence in platform budgets: 86.8% of high-performers report confidence that their platform budgets will renew over the coming years. In comparison, 54.5% of respondents who report having achieved some/limited goals report a lack of confidence in future platform budget renewals. Beyond baseline performance, the chosen platform adoption strategy significantly shapes funding security and support.

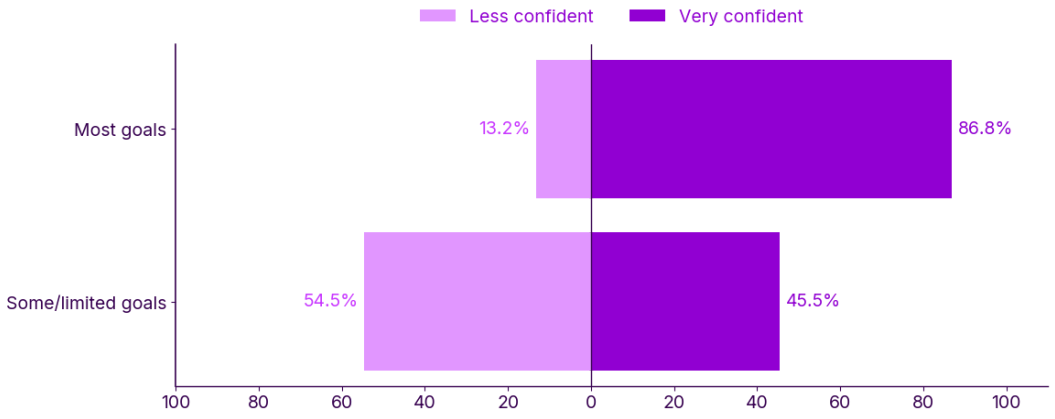


Figure 4.7: *Budget confidence by goals delivered. Bars are normalized within goal outcome groupings.*

Mandatory adoption and budget confidence

Platform initiatives operating under mandatory adoption models continue to report the most confidence in their budget, with 78.9% of mandatory platforms reporting high confidence that their budget will be renewed over the next 5 years, compared to 21.1% who report low confidence. This is consistent with trends from our previous report (**Octopus Deploy, 2025**), where confidence is up, and budget concern is down. A plausible explanation could be that these platforms guarantee company-wide adoption standards, and leadership views them as permanent infrastructure that must be maintained (**Kim et al., 2016** & **Forsgren et al., 2018**).

Alternatively, optional platforms may face great financial uncertainty as they rely on voluntary developer onboarding and adoption. However, we found a notable shift in budget confidence for optional platforms, with 50.0% of respondents reporting high confidence in budget renewal and 50.0% reporting low confidence. While this trend differs from mandated models, there is a significant increase in confidence in optional platforms here compared to our previous report, which found that only 33.0% of respondents were very confident in budget renewal.

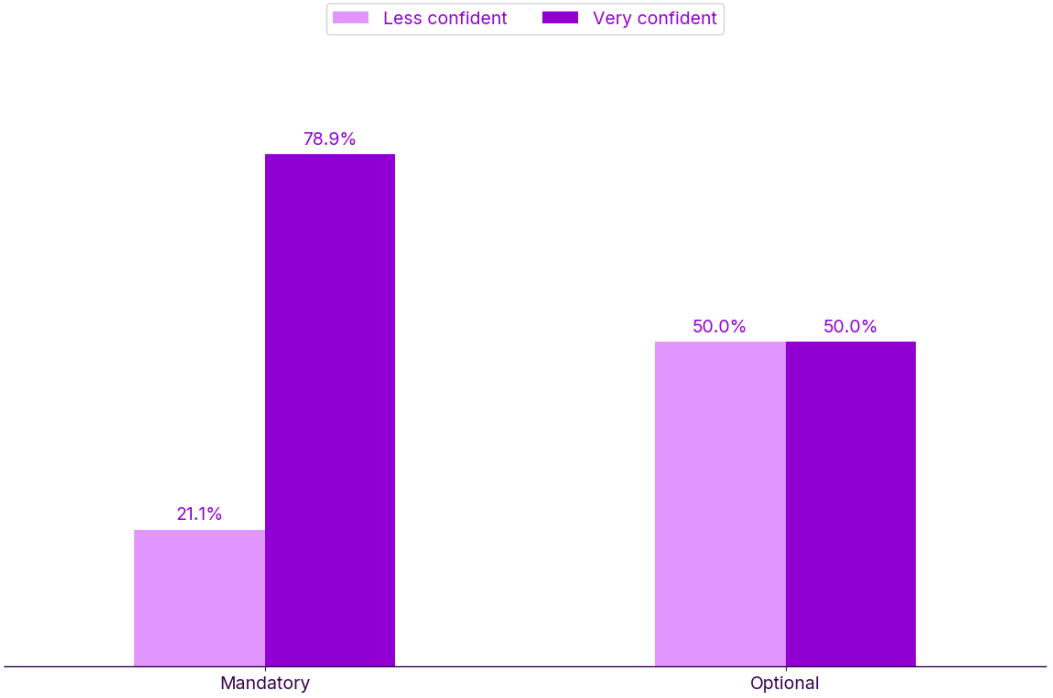


Figure 4.8: *Budget confidence by adoption strategy. Bars are normalized within each group, with mandatory and optional responses across budget confidence levels.*

This shift further suggests that platform teams may be successfully applying the "Platform as a product" approach and achieving positive developer experience (**Skelton & Pais, n.d.** & **Wilsenach, 2015**). Rather than relying on top-down push, optional platforms are focusing on developer experience (DevEx) to drive voluntary options (**Nygard, 2018**). If optional platforms can capture clear performance metrics (e.g., faster onboarding, reduced cognitive load, improved software delivery performance, etc.), they may be better positioned to demonstrate clear ROI to leadership (**Greiler et al., 2022**). While mandatory platforms offer a more direct route to initial budget security, these findings suggest the possibility of creating platforms that developers actively use, thereby becoming an option for securing long-term funding.

Alongside the adoption strategy performance data, this points to the same conclusion: mandatory adoption buys performance and budget security now, while a well-built optional platform earns its funding by being one that developers actively choose. The goal is a platform that would survive being made optional.

We tested three hypotheses in this section, and none fit neatly into a simple yes-or-no. **Mandatory adoption**, not optional, is more closely associated with high-performers, though this answers the performance question rather than whether developers would choose the platform. When developers have to use a platform, high-adoption doesn't indicate how good it is. This makes the second finding more important: teams measuring **developer satisfaction** are more likely to be high-performers, yet a third still don't measure it at all. On feedback, **weekly** is the sweet spot, with **daily** ranking below it, and where feedback is gathered matters as much as how often. **Code reviews and pull requests** track with high-performers, while the most common channels, **informal discussion** and **dedicated communication channels**, are far less effective.

Summary

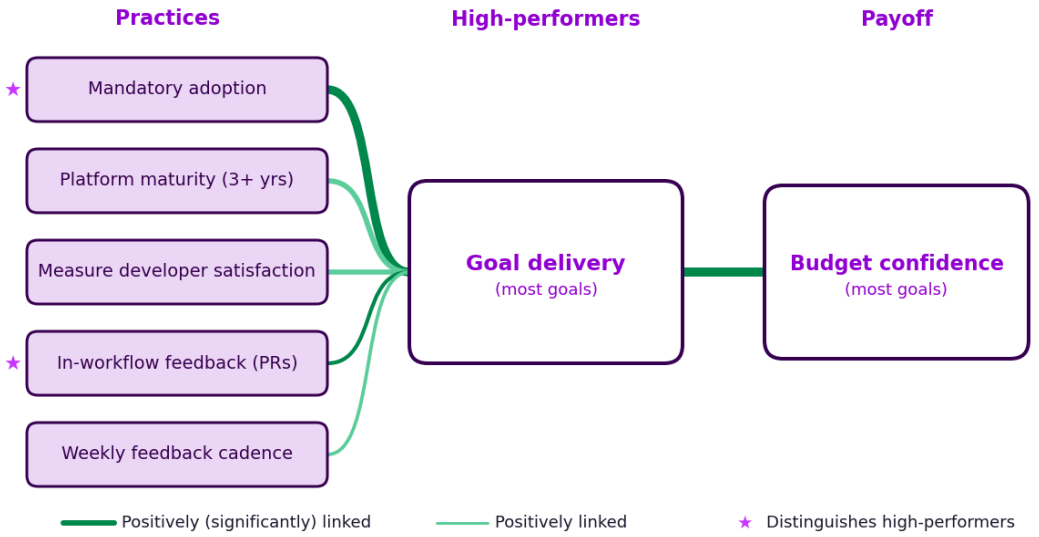


Figure 4.9: *Section 4 Summary.*

SECTION 5

Measuring platform success

How do platform teams know their platform is working? This section looks at the metrics and frameworks teams use to measure success. Developer experience is increasingly measured alongside traditional operation metrics, however, a large portion of respondents reported they don't measure platform metrics at all.

Key takeaways

1. While developer satisfaction is more commonly measured, there is still a gap, with 33.3% of teams not measuring it, and most relying on informal manual assessment.
2. Faster onboarding shows no link to goal delivery: onboarding time is really a proxy for documentation and automation quality, not a target in itself.
3. Teams measure delivery first and experience second.

MONK metrics

The **MONK metrics** were created to specifically measure Platform Engineering. Where established frameworks like DORA focus on software delivery, MONK metrics measure the platform's reach and impact. The balance between the two shows whether teams are measuring the developers they serve and the code they ship (**Fenton, 2025**).

Use of MONK metrics increased this year. Measuring developer satisfaction with the **net promoter score (NPS)** or **customer satisfaction score (CSAT)** is now used by almost a quarter of respondents, up from 5.2% in 2025. **Key customer metrics** are the most widely tracked, with 48.1%, while **market share** and **onboarding time** both sit at 5.9%. (almost all respondents provided market share and onboarding time, but only those measuring them for performance purposes are counted here.)

MONK metrics	% of respondents
Market share	5.9
Onboarding	5.9
NPS or CSAT	25.0 (NPS; 12.5, CSAT; 12.5)
Key customer metrics	48.1

Percentage of respondents who measure each MONK metric for performance purposes, based on categorized responses.

The rise in developer-satisfaction measurement is consistent with the **2025 Platform Engineering Pulse report's** recommendation to use a range of metrics spanning technical performance and user satisfaction. This is also consistent with a wider industry shift towards **user-centric metrics** (e.g., adoption rates, time to deploy, and satisfaction scores) rather than platform availability (**Kanani, 2026**).

Market share

Market share captures the number of developers adopting the internal developer platform relative to the number who could adopt it. This highlights the size of the internal market for the platform and how far the platform has reached into that pool of eligible developers. Teams can focus on tech stack support or migration to increase the size of the internal market, and on smooth onboarding and overall usefulness to increase the share of that market.

While only 5.9% of respondents reported measuring market share, our survey let us calculate it independently. We include it because it measures platform reach, rather than just adoption counts; a platform used by 100 developers means something different to an organization with 100 developers compared to one with 1,000 developers. Framing platform adoption relative to the eligible user base separates those who have saturated a small audience at their organization from those still growing into a larger one.

Most platforms in our dataset have already reached the majority of eligible users. Among respondents who reported both the number of developers who could use the platform and those who do use it, the median market share is 100% (i.e., the bubbles sitting on the diagonal line). Those below 100% are typically reaching more than half their potential audience, and only a portion of respondents reported a market share below 50% (i.e., the bubbles below the diagonal line), indicating that eligible users are not yet using the platform.

This distribution is heavily weighted toward the top; adoption is not the remaining lever, but perhaps expanding what the platform initiative covers (i.e., increasing platform breadth or helping users update from an unsupported tech stack to a supported one) so that the eligible user base widens.

Platforms that have not yet reached their full market share can increase their footprint by identifying gaps in tech stack support. By expanding the platform to support unsupported stacks or migrating users from legacy systems, teams can broaden the eligible user base and increase the platform's market penetration.

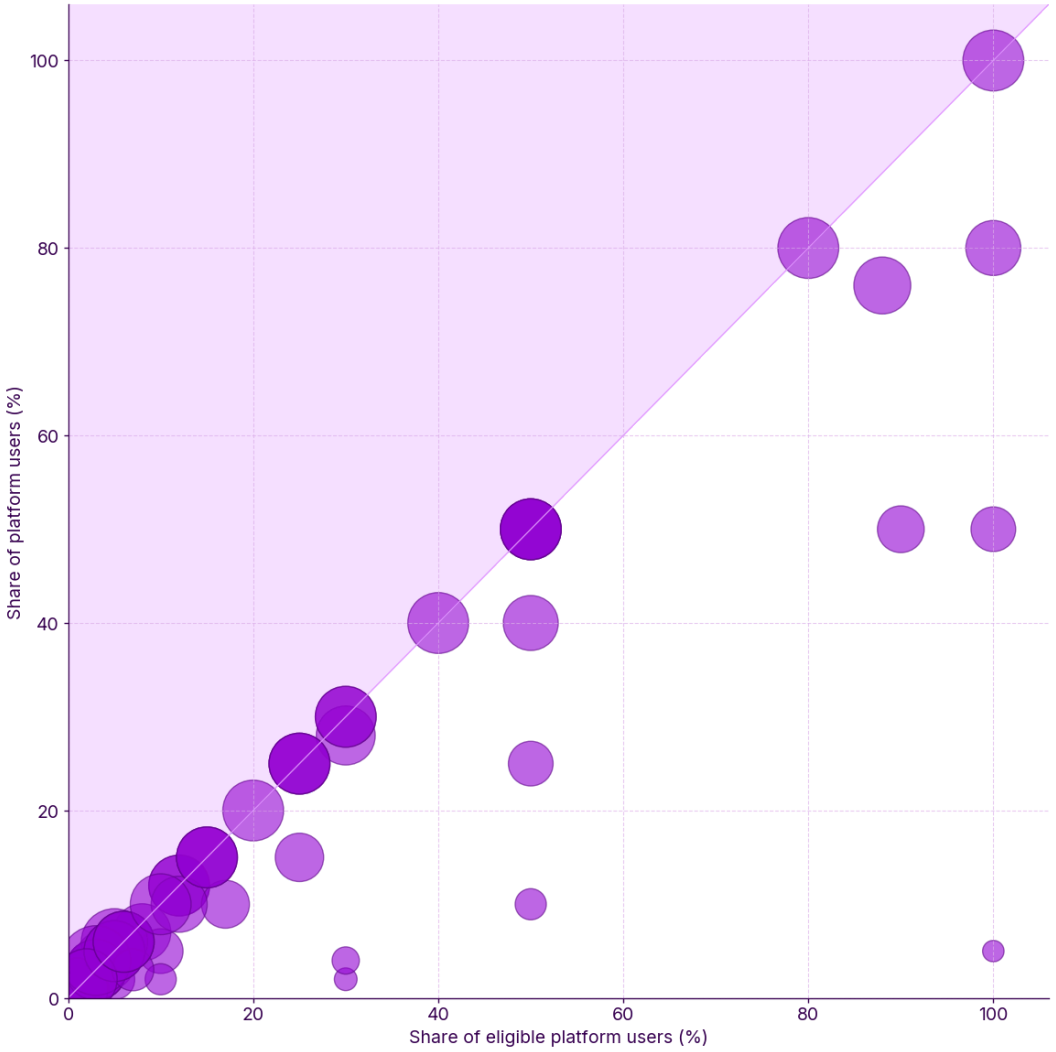


Figure 5.1: *Market Share. Share of platform users relative to the share of eligible platform users, expressed as percentages.*

Onboarding

When we think of **"onboarding"** for Platform Engineering, it can refer to several things:

- **People:** A developer switching onto a team, getting their first change to production.
- **Teams:** An existing team adopting a platform for the first time.
- **Projects:** A new project going from blank page to walking skeleton.

For this section, we're looking at onboarding a new developer and asking respondents how long it takes for new developers or those changing teams to get their first change to production. A similar set of onboarding tasks is needed when you move between teams or start working on an application that you haven't worked on before.

Onboarding time can be a clear indicator of whether a platform actually lowers the barrier to shipping, and there is a recurring claim that good platforms make new developers "productive" quickly (**Cycloid, 2023**).

The most common response was "less than a week" (24.7%), followed by two weeks (22.7%), and one month (19.6%).

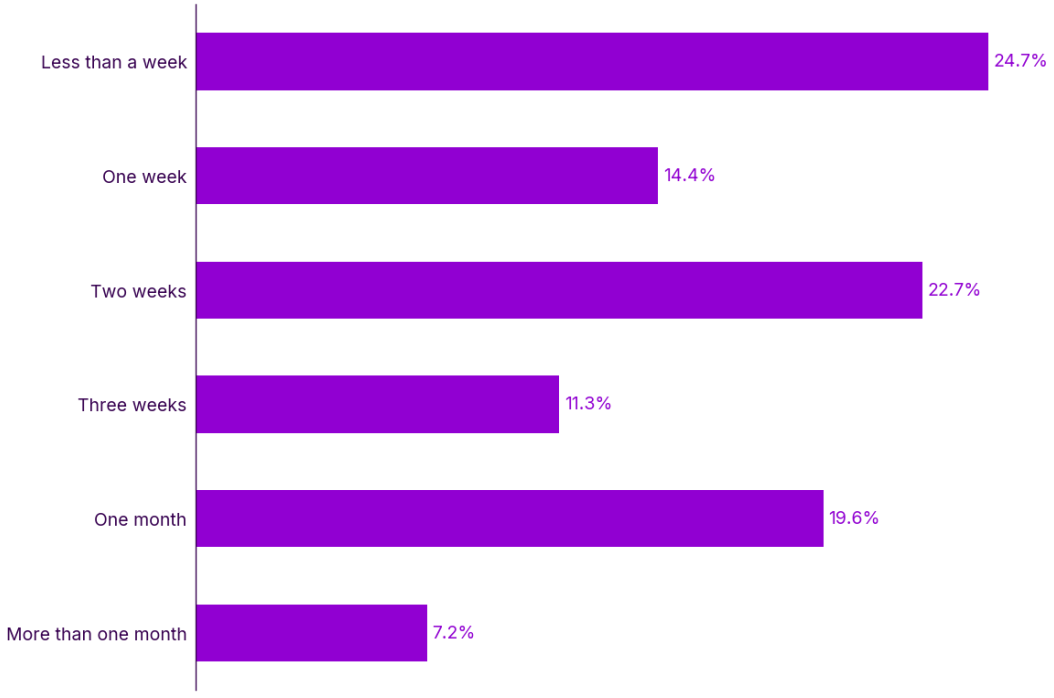


Figure 5.2: *Onboarding time. "How long does it take to onboard a new developer? The time it takes a developer to get their first change deployed to production, whether they have joined the organization or changed teams, (single select)."*

Onboarding speed and goals delivered

We expected faster onboarding to correlate with success; however, we found that onboarding speed does not correlate with more goals delivered. The fastest onboarding time, **less than a week**, had 47.8% of respondents being high-performers, while the slowest (**more than a month**) had 60.0%, and **two weeks** had 31.8%.

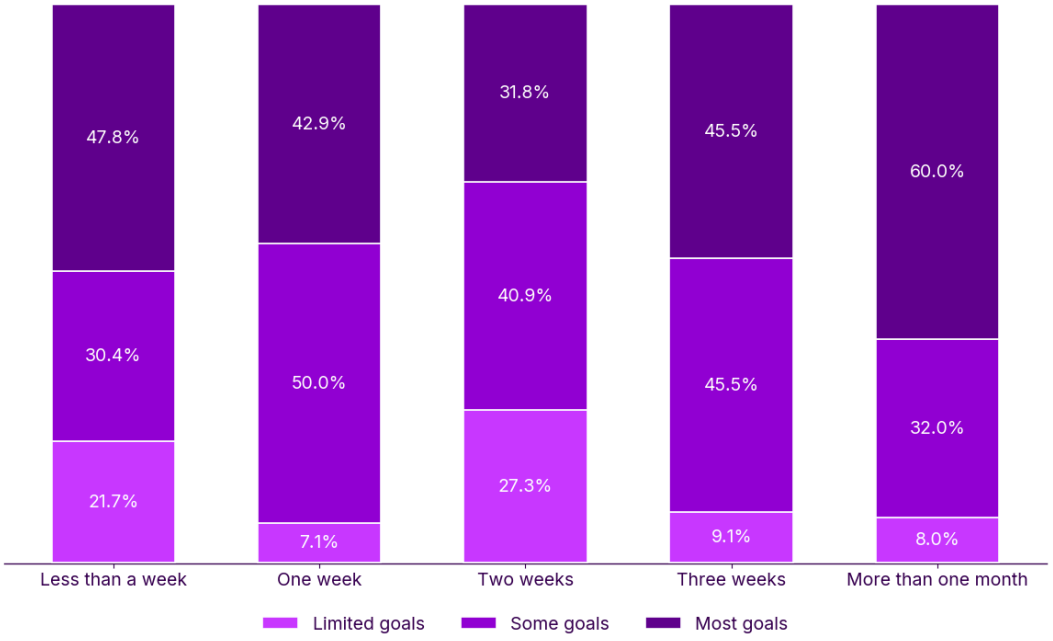


Figure 5.3: *Onboarding time by goals delivered.*

This could indicate that platforms have a broader scope and present more to onboard, so a longer, more thorough onboarding process may be associated with high-performing teams. At the same time, extended onboarding times does not always reflect platform complexity, but are often driven by organizational processes, like mandatory orientation programs, security compliance, or risk mitigation strategies that deliberately gate a developer's first production change until specific training milestones are met.

Onboarding as a metric isn't just to speed up that first change; it also reflects the quality of your documentation and the level of automation in your deployment pipeline. This framing aligns with a survey respondents perspective:

> Successful platform adoption is not about faster onboarding, but more consistent.
>
> *Liam Mackie, Lead Cloud Engineer at Octopus Deploy*

We found that the teams onboarding new users fastest also tend to rate their documentation more favorably; however, it does point in a direction where teams designing for that experience from the outset, treating smooth onboarding, dependable documentation, and reliable support as built-in concerns rather than afterthoughts (**Bridgwater, 2025**).

How teams measure developer satisfaction

Given that measuring developer satisfaction has increased over the past year, we looked at how teams are actually capturing this, which is a signal of how systematically a team listens to its developers. The most common method is manual assessment (40.3%), ahead of CSAT, and NPS, both at 12.5% with a total of 25.0%. One-third of respondents (33.3%) reported not tracking developer satisfaction at all.

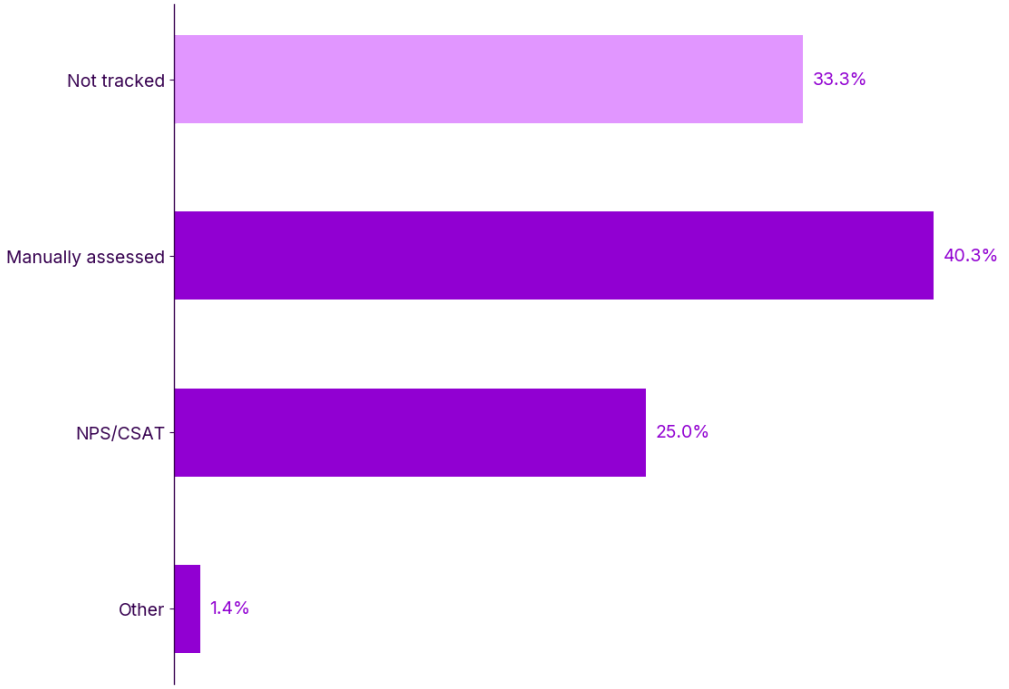


Figure 5.4: *Distribution of responses to "How is developer satisfaction assessed? (i.e., manual, NPS, CSAT, etc.)" represented as % of respondents (single-select).*

We checked whether teams that relied on manual assessments listened to their developers less systematically than other respondents reporting other methods of measurement. We found that these respondents were just as likely to run formal feedback channels, and their feedback types and frequency looked much the same as those of respondents using more formal developer satisfaction assessments. In other words, gathering feedback manually doesn't mean a team listens less often, or even less reliably, only that it records the signal differently.

Because Platform Engineering treats developers as customers of a product, NPS, which captures satisfaction, loyalty, and the likelihood of recommending the product or service, transfers naturally to developer experience. For detailed information on how NPS and CSAT scores are calculated, please refer to **Measuring platform satisfaction: The 3 most helpful techniques**.

Which metrics get tracked together

To understand how teams and organizations think about measurement, we looked at which metrics are commonly measured together, as this can be a good indicator of how teams and organizations think about measurement (i.e., whether the delivery signal travels to delivery, or if user-facing metrics are involved). Only respondents who actually track something were included in this measurement.

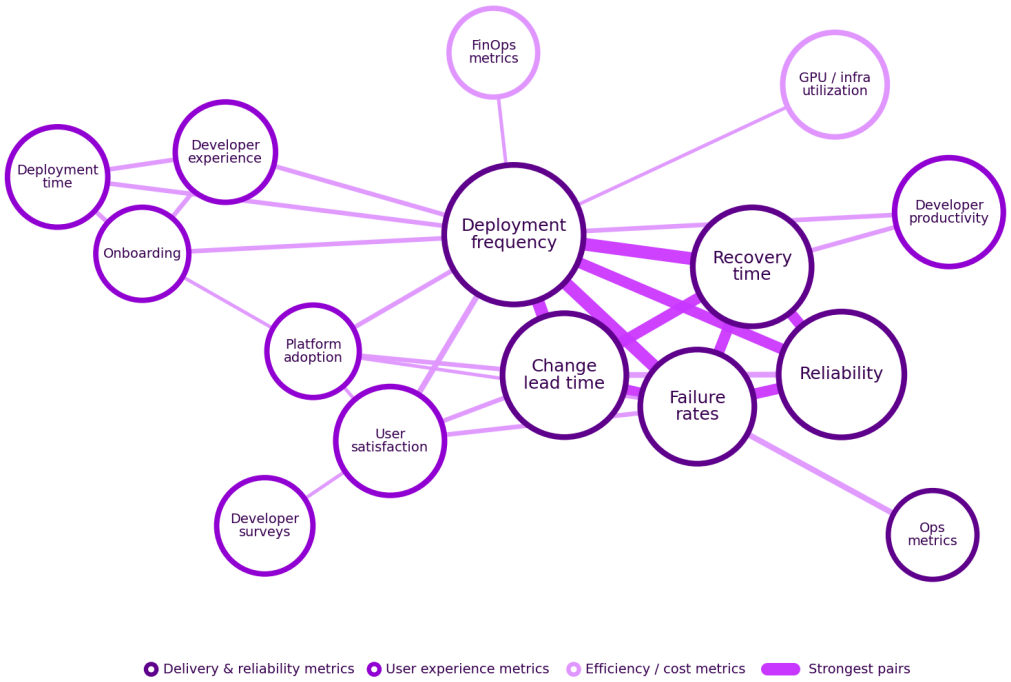


Figure 5.5: *Co-occurrence of the most common performance metrics that platforms use to measure success.*

The strongest pairings are all within the delivery and reliability area, with deployment frequency as the most commonly tracked metric alongside recovery time and failure rates. Reliability was commonly paired with failure rates, change lead time with failure rates, recovery time, and deployment frequency with reliability. This indicates that teams treat delivery speed, failure, and recovery as a connected set of metrics.

Software delivery metrics were found to pair with user-facing metrics (e.g., deployment frequency with user satisfaction and platform adoption); however, they are tied to delivery health rather than tracked in isolation. Teams are measuring delivery and reliability first, with user-facing metrics as a second-tier, lighter layer, which aligns with the industry as a whole and the push towards developer-centered measures alongside delivery metrics (**Kanani, 2026**).

SECTION 6

How the findings connect

Previous sections analyzed each finding in isolation. This section examines how they co-occur among respondents, revealing four mutually reinforcing clusters. Each pair of ranked measures was tested using Spearman's rank correlation, and pairs that show strong associations and significant associations are shown in the network themes below, with line weight reflecting the strength of each association. A rank correlation assumes a relationship that moves in one direction, so a measure whose effect peaks in the mid-range (e.g., feedback cadence) appears more weakly connected.

The cross-sectional nature of our data prevents us from determining causality; therefore, the clusters themselves are the main findings. These are not independent variables, and progress in one area is likely to correlate with movement in others.

Theme 1: Mandatory platforms, organizational governance, and goal delivery

The first cluster ties goal delivery to organizational governance rather than to any single practice. Respondents achieving most goals were also more likely to report mandatory adoption, positive perception of leadership direction, higher job satisfaction, and greater confidence in budget safety, which all also correlate with one another.

At the center, goal delivery, mandatory adoption, and budget safety each move together, so all three reinforce one another. Tracking developer satisfaction and feedback frequency connects to the central cluster through budget safety rather than mandatory adoption. Both of these create a record of how the platform performs, and suggests that measuring developer satisfaction, could be useful evidence in a renewal conversation. This aligns DORA's research where platforms deliver greater returns when they are run as internal products centered on developer experience (**DORA, 2025**). That could suggest that there are two routes to funding: enforce mandatory adoption of the platform so it becomes part of the operating model, or measure developer satisfaction of platform users to justify budget renewal. With optional platforms reporting greater confidence in budget renewal this year, and NPS or CSAT use rising, the second route is becoming very realistic.

There is one association that is inverted. Where adoption is mandatory, respondents rate documentation less favorably, which could also mean that reliable documentation is more closely associated with optional adoption. An explanation could be that a platform that developers are required to use is under less pressure to document well than an optional platform.

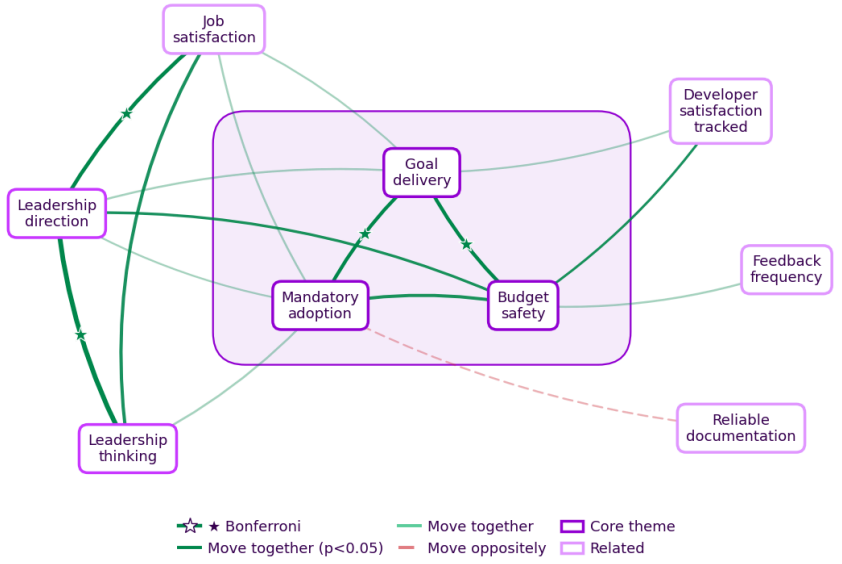


Figure 6.1: *Theme 1.*

Theme 2: Engaged leadership and job satisfaction

The second cluster centers on leadership clarity and how people feel about their work. Leadership direction is the central measure here, significantly associated with leadership thinking and job satisfaction as the core theme. Reliable documentation, onboarding speed, and goal delivery attach from the outside. Clear strategic direction and a well-documented, straightforward platform to join all reduce ambiguity and the mental load of daily work, and let people do that work and report it (**Forsgren et al., 2018**).

Documentation and onboarding speed are connected to job satisfaction rather than goal delivery, and to each other as well, with the fastest-onboarding teams rating their documentation most favorably. Together, they form a secondary cluster around the experience of using the platform. Onboarding time, therefore, could be a measure what joining is like rather than what the platform produces, and dependable documentation is much of what makes it smooth.

Goal delivery reaches the core only through leadership direction and job satisfaction, which suggests that leadership clarity affects how people feel about their work more directly than what they produce.

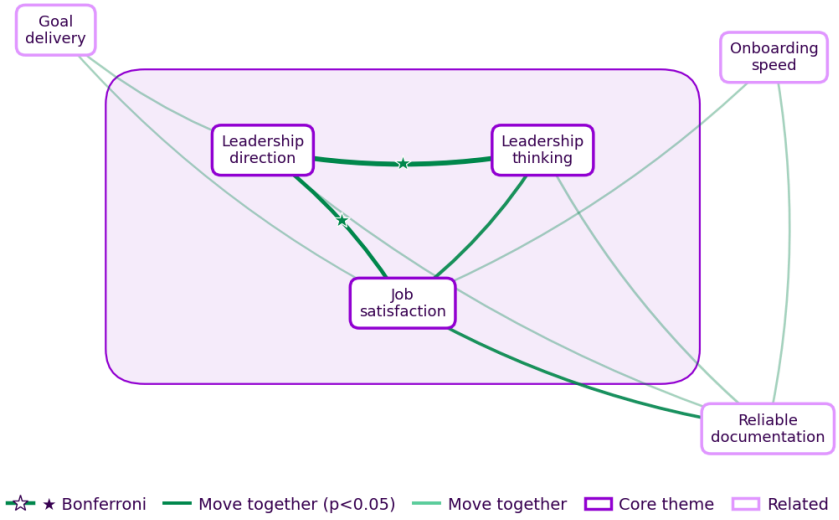


Figure 6.2: *Theme 2.*

Theme 3: Workload and well-being

The third cluster centers on well-being and work-life impact, which move together: respondents reporting better well-being also report better work-life balance. Industry research demonstrates that practices aligned with delivery performance also support healthier teams (**Forsgren et al., 2018** & **DORA, n.d.**). Our data complicates that picture.

This is the only theme built mainly from inverse relationships, with almost everything attached to the core measures but moving in opposite directions. Feedback frequency and reliable documentation are significantly associated with both core measures. Leadership thinking is tied to well-being alone, as is goal delivery (although weakly).

Rather than documentation or feedback frequency directly driving burnout, both may be markers of how close a respondent sits to the platform's daily operations. The people who know the documentation is reliable are those deeply involved in the platform and who gather feedback more frequently; this could actually signal a struggling platform or add another demand on those responding.

Whether developer satisfaction is tracked is the exception, with work-life balance moving with it. This suggests that formalizing how developer experience is captured may differ from informal measures, though these organizations also report slower onboarding, perhaps because they measure it rather than estimate it.

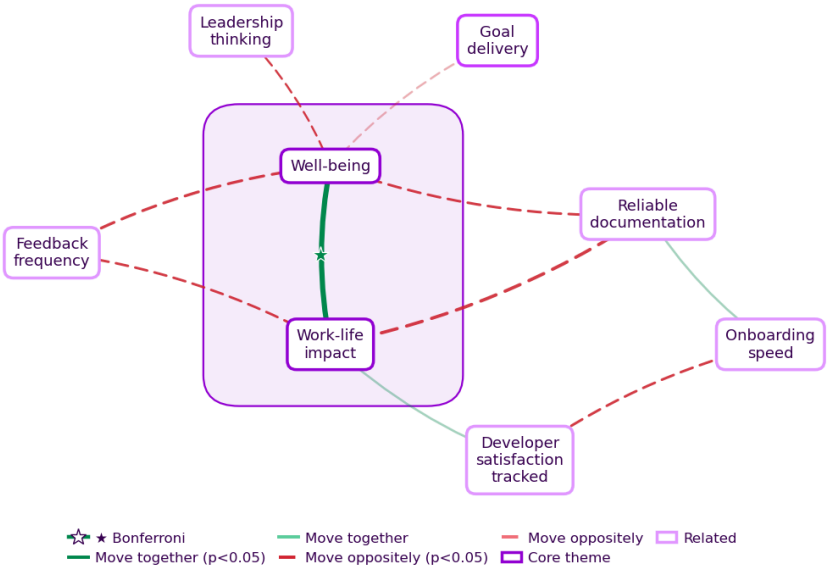


Figure 6.3: *Theme 3.*

Theme 4: AI impact across delivery, performance, and legacy modernization

The fourth cluster centers on the impact of AI. Four measures move together as the central theme: AI impact on personal performance, software delivery speed and stability, legacy system modernization, and organizational performance. These are all significantly associated with one another, and are self-reported perceptions so the clustering in this case is better read as respondents holding one overall view of AI's impact. The organization's shift in priorities towards AI sits just outside of the core cluster, with a range of measures from themes 1 and 2 attaching from the outskirts.

The strongest links are to budget safety and goal delivery (from AI impact on organizational performance), with other associations to leadership direction, leadership thinking, and feedback frequency (through organizations' shift in priorities towards AI). Taken together with the feature-level findings in section 2, this is consistent with AI acting as an amplifier rather than a fix (**Forsgren et al., 2018** & **DORA, 2025**).

Some relationships run the other way, and one clear finding actually separates two measures that are easily mistaken for each other. This shows that goal delivery was lower for organizations that shifted priorities heavily towards AI. Shifting towards AI and benefiting from it are distinct, and the change in direction seems to affect other goals. AI impact on personal and organizational performance both seem to run oppositely to reliable documentation (weak association), stating that teams reporting the greatest benefits from AI are also the ones reporting that documentation is least reliable.

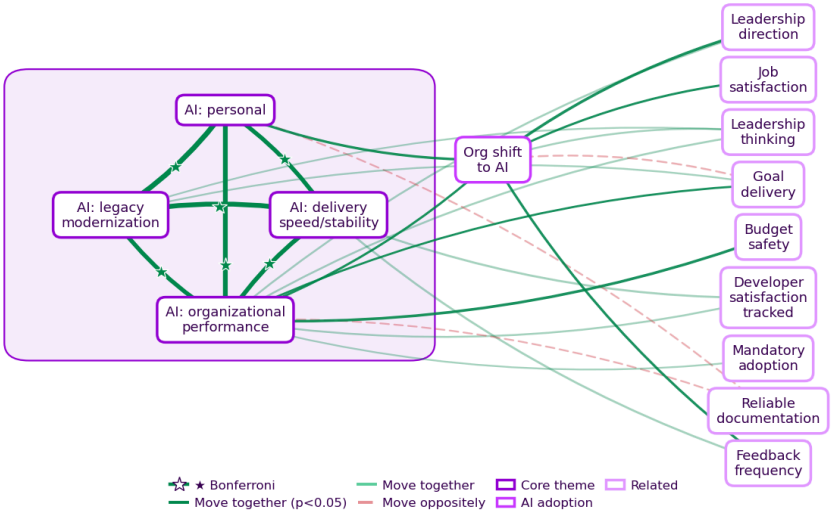


Figure 6.4: *Theme 4.*

SECTION 7

Future predictions and direction

To understand the future of Platform Engineering, a brief look at some history is in order. One example in James C. Scott's **Seeing Like a State** (**Scott, 1998**) concerns industrial forestry. In an effort to make woodland yields more predictable, foresters replaced the tangled mess of diverse woodland with single-species, evenly spaced plantations. The administrators of standardized woodland could easily count, plan, and predict the volume of wood they could obtain from these "forest platforms". The first generation of these forests provided exactly the uniform, high-yield growth they had hoped for.

It was only after the second planting that the problems became clear. Without the tangle of undergrowth, deadwood, and snags, the soil biology that had sustained the forest's productivity collapsed. The lack of diversity left the woodland at the mercy of disease, which could spread quickly through the single-species plantation. The result was forest death, or as the Germans named it, "Waldsterben".

Two crucial lessons from this example inform the future of Platform Engineering. In **Seeing Like a State** (**Scott, 1998**), there are many more, which is why some platform engineering pioneers describe it as essential reading for platform engineers.

Authoritarian	**Authoritarian standardization** impedes the practical knowledge, informal processes, and improvisation that organizations need during periods of unpredictability.
Catastrophic	Early results from **catastrophic standardization** appear promising but precede a terrible collapse.

New tool selection criteria

The teams that don't tighten their selection criteria will achieve reasonable early success but face eventual **Waldsterben**.

In Section 2, high-performers' preference to buy tools rather than build them from scratch was a clear signal. This makes sense, as platform teams should spend their time on problems that are shared by many teams but unique to the organization. The only drawback to this approach is that many tools are not built with platform engineers in mind, so they miss features that could ease the platform team's burden. Tools need built-in support for the kinds of re-use platform engineers are looking for, whether that's template creation, versioning, and updates, or robust customization routes for unanticipated use cases. Platform teams will increasingly filter available tool options based on the presence of platform-easing features.

Policies are mandatory, platforms are optional

Organizations implementing the mantra "policies are mandatory, platforms are optional" will have better platforms than those disguising failing platforms with mandatory adoption.

The ultimate test of a platform is whether developers would prefer to use it rather than roll their own solutions. The strongest signal of this quality is letting them choose whether to adopt the platform. To ensure a level playing field, organizations will make policies and evidence of compliance mandatory, preventing teams from dodging both the platform and the policy. When teams must implement policies, they will gladly use a good platform. If they resist by complying with the policies without using the platform, it's a sign that the platform is troubled.

Summary

The future holds an increasing number of distractions for organizations applying Platform Engineering. We've seen how AI has stolen most of the oxygen in the industry, and platforms have survived unscathed where they have been able to accommodate AI-related platform features, like providing developer tools or controlling their cost.

The long-term survival of Platform Engineering will depend on its ability to handle crucial edge cases, delight the platform's users and stakeholders, and offload further non-novel tasks to tools that understand what developers and platform engineers need.

Conclusion

Platform Engineering success is a strategic and organizational achievement before it is a technical one. The obstacles most associated with lower performance are a **lack of clear strategy**, **constrained budgets**, and **skill gaps**, not **tooling** or **technical complexity**, which teams report at similar rates regardless of how well they deliver. **Goal delivery**, **mandatory adoption**, **leadership direction**, **job satisfaction**, and **budget confidence** cluster together rather than acting as independent measures, suggesting that platforms succeed when they are run as part of a governed operating model rather than as standalone initiatives.

The most notable assumption we tested challenged widespread industry advice, in which we expected **optional adoption** to more closely track high-performers. Instead, **mandatory adoption** was significantly more aligned with high-performers. The number of **obstacles** a platform faced was less impactful than particular obstacles, and there was no association between **onboarding speed** and platform success. Measuring **developer satisfaction** tracked with performance, as did **weekly feedback** cadence, though the method for gathering feedback proved a stronger indicator than how often.

Two other findings stand out. The value of platforms compounds over time, with platforms that run for **3+ years** more likely to be high performers. And AI's impact on software delivery and organizational performance were most positive from teams with more established platforms beneath them, though shifting priorities towards AI did not itself track with more goals being delivered.

Together, these findings describe a maturing discipline. The producer-consumer perception gap around mandatory adoption has effectively closed, and adoption has saturated the eligible user base for most platforms. Platform teams are responsible for a considerably broader scope than a year ago, reflecting both the growth of the discipline and how much more organizations now ask of it.

The emphasis has shifted from whether to build a platform to how to operate one well. Establishing a clear strategy, deliberately selecting tools, and gathering feedback through channels closest to developers' work are key to its success. However, these same practices are felt most by the people closest to the platform's daily operations. What stands between platforms and their goals is organizational rather than technical, and that is the encouraging part, as these are choices an organization can make.

Authors



Dr. Charlotte Fleming, PhD

Researcher, Developer Relations

Charlotte Fleming is a Researcher in Developer Relations at Octopus Deploy and lead author of the Future of Platform Engineering report. She holds a PhD in Neuroscience, and previously authored Octopus Deploy's Platform Engineering Pulse and the AI Pulse report.



Steve Fenton

Principal DevEx Researcher

Steve Fenton is a Principal DevEx Researcher at Octopus Deploy, a DORA Community Guide, and an 8-time Microsoft MVP with more than two decades of experience in software delivery. He has written books on TypeScript (Apress, InfoQ), Octopus Deploy, and Web Operations.



Heidi Waterhouse

Technical Community Advocate

Heidi Waterhouse is a Technical Community Advocate at Octopus Deploy, where she works to connect deployment practitioners with best practices across the industry. She is a nerd about feature flags, industrial psychology, and disaster stories.

Contributors, advisors, and industry experts



Matt Allford

Developer Advocate

Matt Allford is a Developer Advocate at Octopus Deploy with over 15 years of experience across infrastructure, development, and operations. With a passion for education and content creation, he has authored courses at Pluralsight and presented at conferences across Australia and America.



John Bristowe

Principal Developer Advocate

John Bristowe is a Principal Developer Advocate at Octopus Deploy, where he creates practitioner-focused content on Continuous Delivery, deployment governance, and Platform Engineering. He previously worked at Microsoft and Progress/Telerik.



Liam Mackie

Lead Cloud Engineer

Liam Mackie is Lead Cloud Engineer at Octopus Deploy, working in the team that is building the next generation of Kubernetes deployment tools. He brings over a decade of experience with Linux, virtualization, observability and cloud-native technology.



Tim Nicholas

Lead Automation Architect and Product Owner at Octopus Deploy

Tim Nicholas is a Site Reliability Engineer on Octopus Deploy's Developer Platform team, with 25 years across Engineering, Architecture, and Product roles. He focuses on resilience and operational outcomes in complex, dynamic environments.

Denis Jajcevic

Principal Platform Architect at CROZ

Denis Jajčević is a Platform and Solution Architect who helps organizations design scalable internal developer platforms and modernize software delivery, with a particular interest in platform-as-a-product thinking and developer experience.





Tod Thomson

Senior Software Engineering Manager at Octopus Deploy

Tod Thomson is a Senior Software Engineering Manager at Octopus Deploy. He works at the intersection of engineering leadership, delivery, strategy, and cloud, focusing on deployment pipelines and in-product AI capabilities, with a background spanning .NET and DevSecOps/SRE.

SECTION 8

Methods & firmographics

This report uses a mixed-methods approach. Primarily based on quantitative survey data, which established the broad patterns in how internal developer platforms are built, adopted, and measured, and a follow-up set of qualitative interviews exploring the findings that emerged from it. The survey shows what platform teams report; the interviews help explain why. Throughout the report, we distinguish carefully between findings that are statistically significant and directional only, and we make no causal inferences from cross-sectional data.

Quantitative survey

The survey included a range of open-text, multiple-choice, and Likert style questions, with an "other" option allowing respondents to add entries not included in the options provided. The survey was distributed through **typeform.com** and shared via email, social media, and a research panel used in previous studies over a period of 4 months. We received 379 responses. All figures in this report are based on that survey sample.

Data preparation

Several questions offered an "I don't know" option. Many of these were excluded as they carry little information about the quantity being measured. Unless otherwise stated, a percentage reflects the proportion of respondents who chose a given option for that question.

The question format determined how the percentages are displayed:

- **Single-select questions** (only one option could be selected) produce percentages that sum to approximately 100% across the available options.
- **Multi-select questions** (any number of options could be selected) produce percentages that can and usually add to more than 100%, because each respondent may contribute to several options. For these questions, the percentage should be read as the share of respondents who selected that option.

Categorizing open-text responses

Several questions allowed open-text responses, either as an "other" option or as an open prompt (e.g., the reasoning behind build-versus-buy decision-making). These were analyzed with thematic coding where responses were read in full, and similar responses were grouped, into a small set of recurring themes (**Braun & Clarke, 2006**). Each response was assigned to the category it best fit. Categories were refined and closely related answers were merged, and the frequency of each category is reported. One-off responses are noted but not over-interpreted, and very small categories (n < 3) are suppressed in line with the rest of the analysis.

Co-occurrence analysis

This looks at which options respondents tended to pick together on the multi-select questions. For each pair of options, the number of respondents who chose both was counted. These counts were then adjusted for each option's popularity.

Statistical analysis

All data preparation, statistical testing and graphs were produced in Python, using `pandas` and `numpy` for data handling, `scipy.stats` for statistical analysis and hypothesis testing, and `matplotlib` and `seaborn` for visualization. A few choices to note:

- Percentages were calculated with `pandas value_counts(normalize=True)`, and multi-select questions were split so each option could be counted on its own.
- Ordinal fields like **goal delivery**, **onboarding-time**, **job satisfaction**, etc. were converted to numeric scores using `.map()` (e.g., **goal delivery** was mapped from **No benefits delivered** (0) through to **All benefits delivered** (4))
- **The statistical tests** (`spearmanr`, `mannwhitneyu`, `fisher_exact`, and `chi2_contingency`) are all non-parametric, chosen because the survey data is mostly ordinal or categorical rather than normally distributed.
- Confidence intervals for effect sizes were estimated by bootstrap resampling (i.e., recalculating the effect size across 3,000 resampled version of the data, so the width of the interval reflects precision) rather than a parametric formula.

Associations between two ranked measures were assessed with **Spearman's rank correlation (ρ)**. Other tests cited in the report are described below.

Mann-Whitney U test

This test is a non-parametric test that compares two independent groups by ranking their values and asking whether one group tends to score higher than the other **(Mann & Whitney, 1947)**. It makes no assumption that the data is normally distributed (**Hart, 2001**, and **Kerby, 2014**).

Why it was used: it was used for outcomes like **goal delivery**, which is ordinal and skewed, and many of our questions split respondents into two groups (e.g., teams where platform adoption is mandatory compared to optional). Mann-Whitney U test is the appropriate test for comparing an ordinal outcome across two groups (**Hart, 2001**).

Rank biserial correlation

A p-value shows whether a difference is likely to be real, but not how big it is. To capture size, each Mann-Whitney U test was paired with its natural effect size, the **rank-biserial correlation**. It is the difference between the proportion of favorable and unfavorable pairs (rank = favorable - unfavorable) when every value in one group is compared with every value in the other. It runs from -1 to +1, so 0 means the groups are indistinguishable, and values near ±1 mean one group almost always ranks above the other (**Fiel Peres, 2016**).

Why it was used: it gives the size and direction of a group difference in a simple, bounded number, and it matches the Mann-Whitney U statistic directly. Each estimate is also accompanied by a confidence interval (a plausible range of the true value) to assess reliability. It is reported with bootstrap confidence intervals (3,000 resamples) so the reader can see how precise each estimate is. This range was worked out by bootstrapping, which re-runs the calculation on 3,000 randomly reshuffled versions of the data: a narrow range means the estimate is precise, and a wide one means it is less certain (**Fiel Peres, 2016**).

Chi-square test

The chi-square test checks whether two categorical variables are related. It compares how often each combination of answers actually occurs with how often it would occur if the two were unrelated, then boils that gap down to a single number and a p-value (the chance of seeing a gap this big if nothing were really going on) (**McHugh, 2013**).

Why it was used: it is the standard test for whether two categorical or yes/no variables go together (**e.g.,** whether offering a particular feature tends to go with being a high-performer). It is dependable when there are enough responses in each cell of the grid (a common rule of thumb is at least 5 in each cell).

Fisher's exact test

Fisher's exact test asks the same question as chi-square for a small two-by-two grid, but instead of estimating the p-value, it works out the exact probability directly. Chi-square only gives an approximate answer, and that approximation is less reliable when numbers are small.

Why it was used: with a sample of around 100 split across many yes/no comparisons, some grids end up with only a handful of responses in a cell, where chi-square is unreliable. Fisher's exact test provides a reliable p-value, so it is used for small or sparse 2x2 comparisons, with chi-square used when cell counts are sufficiently large (**Mays & Stark, 2026**).

Bonferroni correction

Running many tests in the same group raises the chance of at least one false positive. The **Bonferroni correction** guards against this by dividing the significance threshold (α, normally 0.05) by the number of tests being run, keeping the overall false-positive risk at or below α (**Frost, n.d.**).

Why it was used: the report tests many features, benefits, and obstacles at once, so uncorrected p-values would overstate how many associations are real. Correction was applied within each group. Bonferroni is deliberately conservative, which suits when wanting to highlight robust important significant findings. A finding is called **confirmed** only if it survives the correction; everything else is reported as **directional** (**Armstrong, 2014** & **Bind & Rubin, 2020**).

Spearman's rank correlation

Spearman's rank correlation measures whether two ranked measures move together. Rather than using the raw values, it ranks each measure from lowest to highest and asks how closely the two sets of ranks agree (**Spearman, 1904**). The result runs from -1 to +1: values near +1 mean the two rise and fall together, values near -1 mean one rises as the other falls, and 0 means there is no consistent relationship (**Schober et al., 2018**).

Why it was used: most of our measures are ordinal rather than continuous with the use of likert style questions (e.g., from strongly disagree = 0; to strongly agree = 5) so they rank easily and can be placed on a single ordinal scale. Spearman is the appropriate test for association between two measures, and is the basis for the thematic analysis in section 6, with all pairs of ranked measures tested. Pairs having strong associations or reaching significance are displayed in the figures.

Qualitative interviews

Interviews were conducted with a small cohort of platform engineering professional, following a loose guide created around the survey headline findings (i.e., adoption strategy, features, high-performing teams, barriers to success, and outcomes associated), while leaving room to follow each participant's experience. Interviews were analyzed and quotations are used throughout the report to give practitioner voice to the quantitative results.

Some limitations of the study

Throughout the report we do point out limitations of our study and why they matter. Here is a brief summary of the main ones to keep in mind, when reading the results: Several constraints should be kept in mind when reading the results.

- **Self-reported responses:** All responses and measures in the survey are self-reported and subject to change depending on social-desirability effects (**Zaal et al., 2026**).
- **Sample size:** The results are based on 379 platform practitioners, but questions were optional and response count varies, therefore findings about smaller subgroups should be treated as indicators only.
- **Correlations:** The survey captures one moment in time, and therefore all associations are correlational, using only associational language, rather than causation language.

Demographic & firmographics

Geographic region

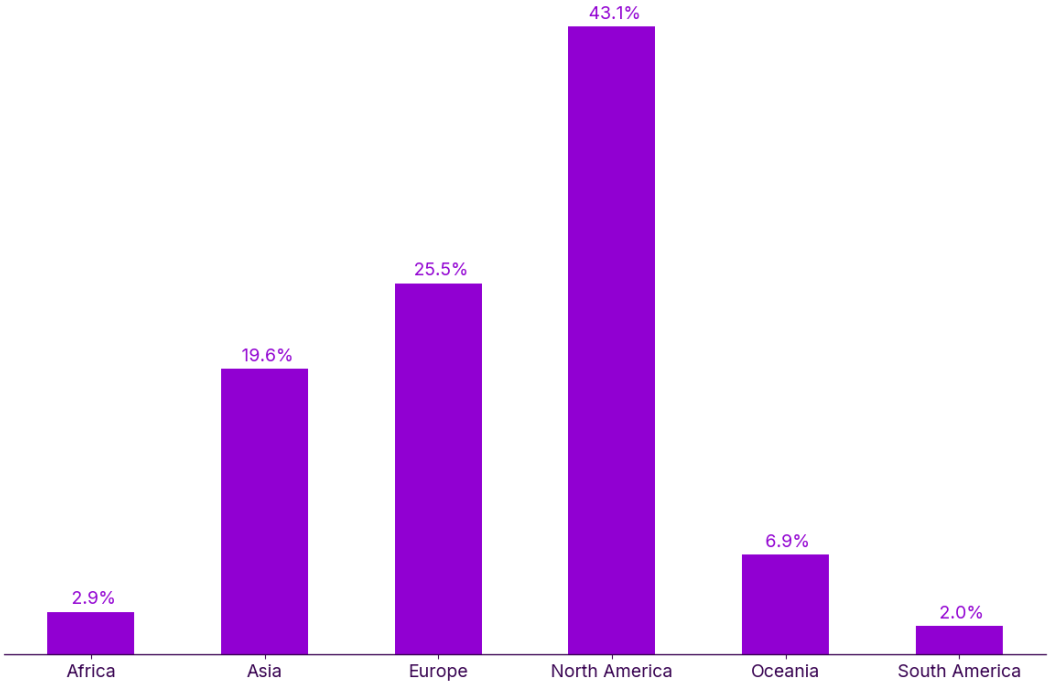


Figure 9.1: *Percentage distribution of respondents geographic location.*

We gathered survey responses from people in every continent, with the most responses obtained from people in North America (43.1%), followed by Europe (25.5%), Asia (19.6%), Oceania (6.9%), Africa (2.9%) and South America (2.0%).

Organization size

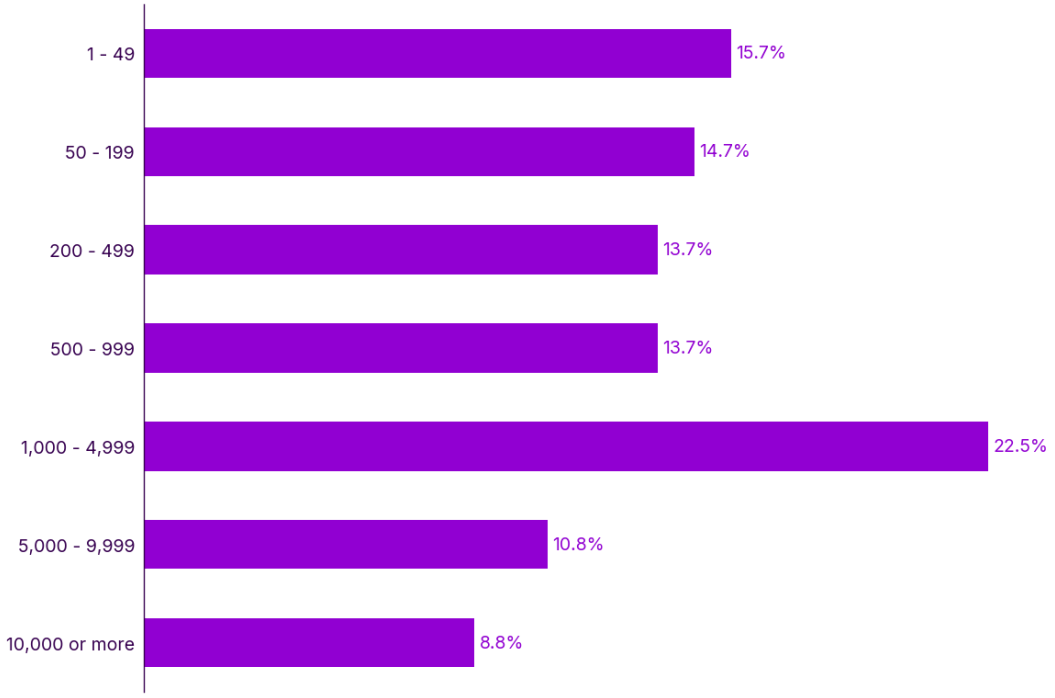


Figure 9.2: *Percentage distribution of respondents organization size.*

We asked respondents how many employees work at their organization. Respondents working at organizations with 1,000 - 4,999 employees (22.5%) were most common, followed by organization with 1 - 49 employees, 50 - 199 employees, 200 - 499 employees, 500 - 999 employees, 5,000 - 9,999 employees, and least common option being 10,000 or more employees (8.8%).

Industry

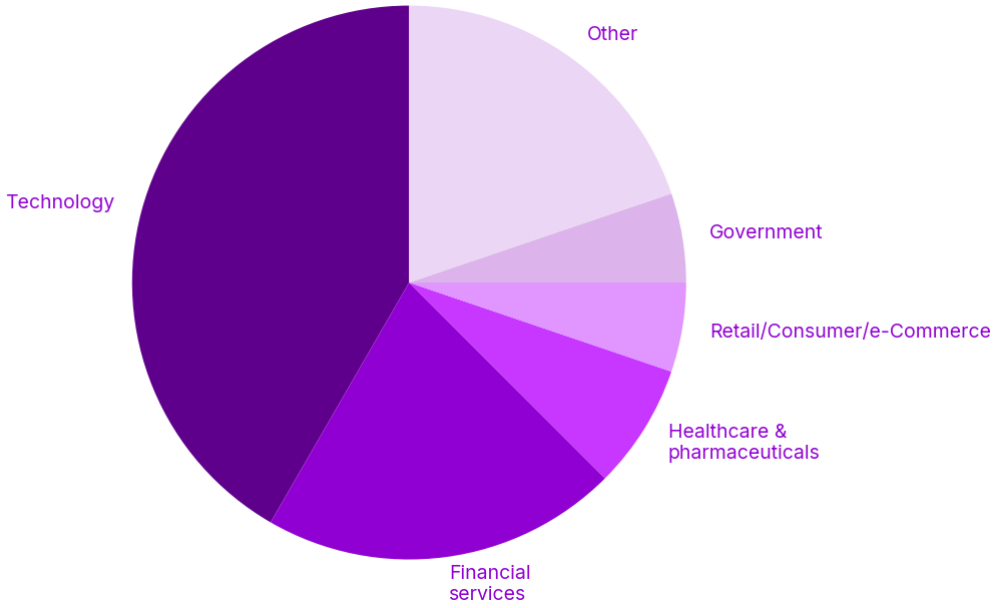


Figure 9.3: *Distribution of respondents industries.*

We asked respondents to identify the industry where their organization most closely resembles, across 11 categories (including an "other" option for free text). The most common industries where respondents worked were Technology (41.7%), followed by Financial Services (20.8%). All other industries founded respondents were below 10.0%.

Job role

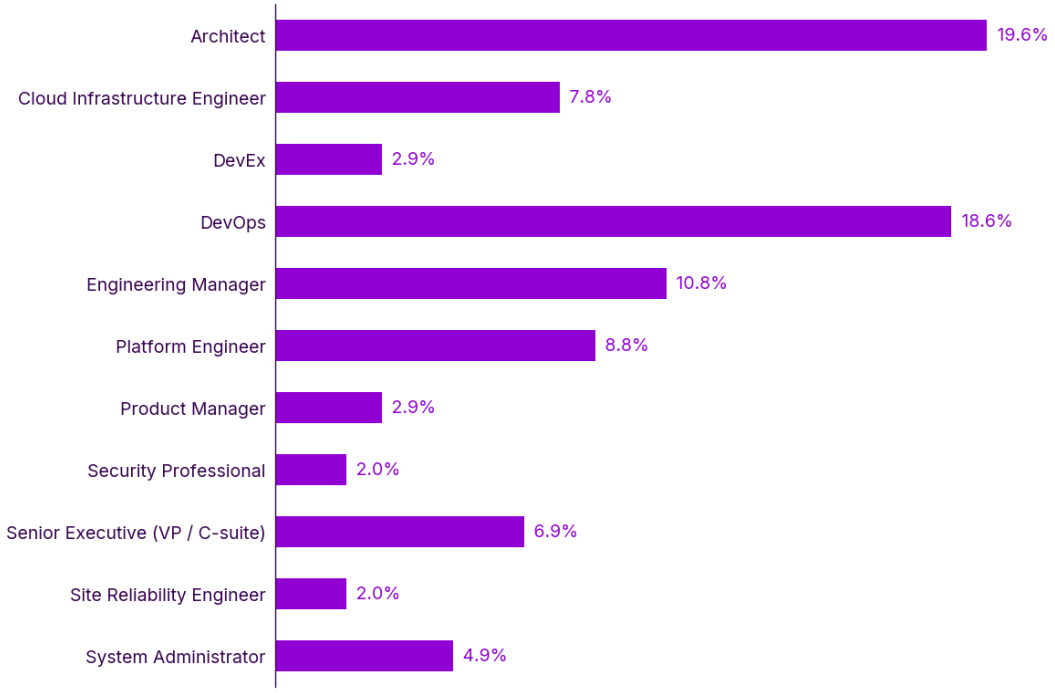


Figure 9.4: *Percentage distribution of respondents job role.*

We asked respondents, to most closely describe their job role, by providing a list of options (including an "other" option for free text) to capture the variety of ways people may be involved with Platform Engineering. The most highly represented roles from our survey data were Architect (19.6%), and DevOps (18.6%), followed closely by Developer (12.7%) and Engineering Manager (10.8%).

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. American Psychological Association. (n.d.). *Burnout research*. **https://www.apa.org/members/content/burnout-research**
2. Andrews, N. (2025, June 30). *Ephemeral vs static environments: Why staging breaks in the age of coding agents*. Signadot. **https://www.signadot.com/articles/ephemeral-environments-vs-static-environments-a-modern-development-shift/**
3. Armstrong, R. A. (2014). When to use the Bonferroni correction. *Ophthalmic & Physiological Optics, 34*(5), 502-508. **https://doi.org/10.1111/opo.12131**
4. Association of Chamber of Commerce Executives. (2021, October 22). *The drivers of burnout*. **https://secure.acce.org/articles/operations-and-finance/the-drivers-of-burnout/**
5. Bedard, J., Kropp, M., Hsu, M., Karaman, O. T., Hawes, J., & Rosen Kellerman, G. (2026, March 5). When using AI leads to "brain fry." *Harvard Business Review*. **https://hbr.org/2026/03/when-using-ai-leads-to-brain-fry**
6. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. **https://agilemanifesto.org/**
7. Bind MC, Rubin DB. (2020, August 11). When possible, report a Fisher-exact P value and display its underlying null randomization distribution. *Proc Natl Acad Sci U S A*, 117(32), 19151-19158. **https://doi.org/10.1073/pnas.1915454117**
8. Bottcher, E. (2018, March 5). *What I talk about when I talk about platforms*. martinfowler.com. **https://martinfowler.com/articles/talk-about-platforms.html**
9. Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology, 3*(2), 77-101. **https://doi.org/10.1191/1478088706qp063oa**
10. Bridgwater, A. (2025, August 12). *Why onboarding is a ramp to platform engineering*. Forbes. **https://www.forbes.com/sites/adrianbridgwater/2025/08/12/why-onboarding-is-a-ramp-to-platform-engineering/**
11. Cloud Native Computing Foundation. (n.d.). *Platform engineering maturity model*. **https://tag-app-delivery.cncf.io/whitepapers/platform-eng-maturity-model/**
12. Cloud Native Computing Foundation. (2025). *Cloud native maturity model*. **https://maturitymodel.cncf.io/**
13. Cycloid. (2023, February 7). *7 platform engineering KPIs you should be tracking*. **https://www.cycloid.io/blog/7-platform-engineering-kpis-you-should-be-tracking/**
14. Deci, E. L., & Ryan, R. M. (2000). The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. *Psychological Inquiry, 11*(4), 227-268.

https://doi.org/10.1207/S15327965PLI1104_01

15. DiGirolamo, M. (2026, March 16). *AI fatigue statistics 2026*. Shibumi. **https://shibumi.com/blog/ai-fatigue-statistics-2026/**
16. DORA. (n.d.). *DORA's research program*. Google Cloud. **https://dora.dev/research/**
17. DORA. (2022). *Accelerate state of DevOps report 2022*. Google Cloud. **https://dora.dev/research/2022/dora-report/**
18. DORA. (2024). *Accelerate state of DevOps report 2024*. Google Cloud. **https://dora.dev/research/2024/dora-report/**
19. DORA. (2025). *State of AI-assisted software development*. Google Cloud. **https://dora.dev/research/2025/dora-report/**
20. DORA. (2026). *Platform engineering*. Google Cloud. **https://dora.dev/capabilities/platform-engineering/**
21. Faros AI. (2025). *The AI productivity paradox: AI coding assistants increase developer output, but not company productivity*. **https://www.faros.ai/blog/ai-software-engineering**
22. Fenton, S. (2025, August 4). *Measuring platform satisfaction: The 3 most helpful techniques*. Octopus Deploy. **https://octopus.com/devops/metrics/platform-satisfaction/**
23. Fiel Peres F. (2026 February 15). Effect sizes for nonparametric tests. *Biochem Med (Zagreb)*, 36(1), 010101. doi: 10.11613/BM.2026.010101.
24. Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: Building and scaling high performing technology organizations*. IT Revolution Press. **https://search.worldcat.org/title/1031484927**
25. Forsgren, N., Storey, M.-A., Maddila, C., Zimmermann, T., Houck, B., & Butler, J. (2021). The SPACE of developer productivity: There's more to it than you think. *ACM Queue, 19*(1), 20-48. **https://dl.acm.org/doi/10.1145/3454122.3454124**
26. Frost, J. (n.d.). *Bonferroni correction*. Statistics By Jim. **https://statisticsbyjim.com/hypothesis-testing/bonferroni-correction/**
27. Furnell, G. (2024, July 20). *Chesterton's fence - and the secular view of time*. Australian Chesterton Society. **https://chestertonaustralia.com/article/chestertons-fence-and-the-secular-view-of-time/**
28. Greiler, M., Storey, M.-A. D., & Noda, A. (2022). An actionable framework for understanding and improving developer experience. *IEEE Transactions on Software Engineering*. **10.1109/TSE.2022.3175660**
29. Haigh, M. (2026, January 13). *Platform engineering maturity in 2026: What the data tells us*. Platform Engineering. **https://platformengineering.org/blog/platform-engineering-maturity-in-2026**
30. Hart, A. (2001). Mann-Whitney test is not just a test of medians: Differences in spread can be important. *BMJ, 323*(7309), 391-393. **https://doi.org/10.1136/bmj.323.7309.391**
31. K N, V. (2026, August 11). *AI agents vs traditional automation: Differences & enterprise*. eZintegrations. **https://ezintegrations.ai/ai-agents-vs-traditional-automation/**

32. Kanani, R. (2026, March 30). *Platform engineering trends in 2026: 11 shifts redefining internal developer platforms*. LeanOps Technologies. **https://leanopstech.com/blog/platform-engineering-trends-2026/**
33. Kerby, D. S. (2014). The simple difference formula: An approach to teaching nonparametric correlation. *Comprehensive Psychology, 3*, Article 11.IT.3.1. **https://journals.sagepub.com/doi/10.2466/11.IT.3.1**
34. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations*. IT Revolution Press. **https://dl.acm.org/doi/10.5555/3044729**
35. Kropov, V. (2026, February 26). *Top technology trends in 2026 to watch and adopt*. N-iX. **https://www.n-ix.com/technology-trends/**
36. Kwaśniewski, M. (2024, July 22). *How feature flags help with progressive delivery*. Unleash. **https://www.getunleash.io/blog/progressive-delivery-with-feature-flags**
37. Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics, 18*(1), 50-60. **https://doi.org/10.1214/aoms/1177730491**
38. Maslach, C. (2024, March 19). *Meeting the challenge of burnout* [Conference presentation]. SREcon24 Americas, Santa Clara, CA, United States. USENIX Association. **https://www.usenix.org/conference/srecon24americas/presentation/maslach**
39. Mays, S., & Stark, S. (2026). The use of Fisher's exact test in contingency table analysis in palaeopathology. *International Journal of Paleopathology, 52*, 135-139. **https://doi.org/10.1016/j.ijpp.2026.01.005**
40. McHugh, M. L. (2013). The chi-square test of independence. *Biochemia Medica, 23*(2), 143-149. **https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900058/**
41. Milanović, M. (2026). *Hofstadter's law*. Laws of Software Engineering. **https://lawsofsoftwareengineering.com/laws/hofstadters-law/**
42. Millan, J. (2025, November 29). *Context switching is costing your team 6+ hours a week*. Super Productivity. **https://super-productivity.com/blog/context-switching-costs-for-developers/**
43. Netesanyi, S. (2026, February 26). *Platform engineering trends: Why enterprise CIOs are rebuilding developer experience*. N-iX. **https://www.n-ix.com/platform-engineering-trends/**
44. Noda, A., Storey, M.-A., Forsgren, N., & Greiler, M. (2023a). DevEx: What actually drives productivity. *ACM Queue, 21*(2). **https://dl.acm.org/doi/10.1145/3595878**
45. Nygard, M. T. (2018). *Release it! Design and deploy production-ready software* (2nd ed.). Pragmatic Bookshelf. **https://pragprog.com/titles/mnee2/release-it-second-edition/**
46. Octopus Deploy. (2025). *The Platform Engineering Pulse report*. **https://octopus.com/publications/platform-engineering-pulse**
47. Piaggio, T. (2026, March). *Ephemeral environments: What they are, why they matter, and how to build them*. Autonoma. **https://getautonoma.com/blog/ephemeral-environments**

48. Red Hat. (2024). *State of platform engineering in the age of AI*. **https://www.redhat.com/en/resources/state-of-platform-engineering-age-of-ai**
49. Santos Paulo, R. (2026, April 4). *How leadership drives platform engineering success in 2026*. Everyday IT. **https://www.ai-infra-link.com/how-leadership-drives-platform-engineering-success-in-2026/**
50. Schober, P., Boer, C., & Schwarte, L. A. (2018). Correlation coefficients: Appropriate use and interpretation. Anesthesia & Analgesia, 126(5), 1763-1768. **https://doi.org/10.1213/ANE.0000000000002864**
51. Scott, J. C. (1998). *Seeing Like a State: How Certain Schemes to Improve the Human Condition Have Failed*. Yale University Press. **https://yalebooks.yale.edu/book/9780300078152/seeing-like-a-state/**
52. Skelton, M., & Pais, M. (n.d.). *Team Topologies: Organizing for fast flow of value*. **https://teamtopologies.com/**
53. Spearman, C. (1904). The proof and measurement of association between two things. The American Journal of Psychology, 15(1), 72-101. **https://doi.org/10.2307/1412159**
54. Storey, M.-A., Zimmermann, T., Bird, C., Czerwonka, J., Murphy, B., & Kalliamvakou, E. (2021). Towards a theory of software developer job satisfaction and perceived productivity. *IEEE Transactions on Software Engineering, 47*(10), 2125-2142. **https://doi.org/10.1109/TSE.2019.2944354**
55. Tekkesinoglu, S., Wagner, M., & Runeson, P. (2026). Platform engineering and internal developer portals: A multivocal literature review. *Frontiers in Computer Science, 8*. **https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2026.1814498/full**
56. Tuite, D. (2025, December 23). *Platform engineering in 2026: Why DIY is dead*. Roadie. **https://roadie.io/blog/platform-engineering-in-2026-why-diy-is-dead/**
57. von Grünberg, K., & Galante, L. (2026). *Thinking in platforms*. Weave Intelligence. **https://weaveintelligence.io/thinking-in-platforms-book**
58. Wahid, S. (2026, June 11). *Progressive delivery for CI/CD pipelines*. DEV Community. **https://dev.to/safdarwahid/progressive-delivery-for-cicd-pipelines-3mlm**
59. Wardley, S. (2015, March 19). *An introduction to Wardley 'value chain' mapping*. CIO. **https://www.cio.com/article/196094/an-introduction-to-wardley-value-chain-mapping.html**
60. Wilsenach, R. (2015, July 9). *DevOpsCulture*. martinfowler.com. **https://martinfowler.com/bliki/DevOpsCulture.html**
61. Witmer, E. (2025, June 17). *Beyond static setups: What are ephemeral environments and how Testkube unleashes their power?* Testkube. **https://testkube.io/blog/what-are-ephemeral-environments-testkube-guide**
62. Wolfe, B. (2025, September 16). *Too many tools, too little time: How context switching is killing team flow*. Lokalise. **https://lokalise.com/blog/blog-tool-fatigue-productivity-report/**

63. Zaal, E., Ongena, Y., van der Velden, N., Loughnan, D., & Hoeks, J. (2026). Unraveling honest responding: A systematic review on the effectiveness of social desirability bias reduction methods in survey research. *Quality & Quantity, 60*(3), 10359-10391. **https://doi.org/10.1007/s11135-026-02664-7**

PLATFORM ENGINEERING

Octopus Deploy

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

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