



The 2023 State of Software Delivery



Contents

- 03 Executive summary and key findings →
- 05 Achieving elite status through a holistic software delivery practice →
- 06 What is an elite software delivery team? →
- 07 History of continuous integration →
- 08 How do you compete? →
- 09 Software delivery performance in 2023 →
 - 10 – Duration →
 - 16 – Mean Time to Recovery →
 - 21 – Success Rate →
 - 25 – Throughput →
- 30 Additional findings →
- 31 How Datadog achieves success through robust testing →
- 32 How do team size and company size affect delivery performance? →
- 33 Engineering productivity analysis →
- 35 Language trends →
- 37 To improve delivery outcomes, start with CI →
- 39 Methodology →

Executive summary

THE TECHNOLOGY INDUSTRY HAS ENTERED INTO A PERIOD OF DRAMATIC CHANGE. With the tech sector reeling from sharp declines in investment and valuations, mass layoffs, and macroeconomic uncertainty, many technology leaders are refocusing their efforts — ensuring critical value streams remain stable and employees stay productive and engaged.

In this environment, it is more important than ever that organizations have the tools and processes in place to rapidly deliver high-impact changes without introducing risk or disrupting platform reliability. To enable this, teams are building systems that support fast feedback and scalable processes, with guardrails that protect the values of efficiency, consistency, and reliability. Providing a stable product experience and responding quickly to failures have become crucial value metrics to engineering teams and will only continue to increase in importance in the coming year.

Automating processes in your software delivery practice will help your team solve problems more efficiently. An optimized continuous integration and delivery (CI/CD) practice that includes robust testing is one of the most impactful forms of automation that engineering teams can achieve. Teams that prioritize robust testing as part of their CI practice can save millions of dollars over time: enterprise-level customers of CircleCI, for example, save nearly \$14 million dollars over a three-year period* due to improvements in productivity, efficiency, and code quality.

So, what are some of the practices your team can begin to incorporate to increase your platform's stability, do more with less, and save money?

Want to improve? Start here.



Prioritize the ability to recover from failure quickly.

Without a stable platform, it's impossible to compete, which is not an option in today's tech sector. In our most recent data set, we observed that 50% of CI workflows recovered from a failed run in 64 minutes or less, which is a dramatic improvement from previous years where 73 and 74 minutes were the median recovery times on all project branches. The ability to recover quickly has become mission-critical for engineering teams.



Increase the scope of your testing for more actionable feedback.

Each year, our data validates that robust testing is critical to reaping the full benefits of CI. A robust testing practice will include unit tests, integration tests, UI tests, and end-to-end tests across all layers of your application. It will also take a test-first approach to software design, balancing speed with quality. And mindset matters: organizations that view failed tests as valuable feedback rather than signs of poor performance are likely to have more extensive test coverage, which in turn is more likely to surface issues and bugs before they reach your customers.



Assemble a platform team to scale critical processes and optimize developer experience.

Platform engineering teams are tasked with removing impediments to developer velocity as well as setting guardrails and enforcing quality standards across projects. As organizations seek to increase efficiency, reduce risk, and become more responsive to the demands of an evolving market, platform teams will play a critical role in aligning development practices with core business objectives.



“DevOps has unlocked unprecedented speed in software delivery, and the growth of platform teams in the enterprise continues to validate the need for DevOps at scale. As projects become more complex, the best performing organizations will leverage platform engineering to deliver maximum value not just quickly but consistently and under control.”

– **JIM ROSE**, CircleCI CEO



Where are platform teams making the biggest impact?

Talking to our customers, we consistently hear from platform teams whose primary charter is to accelerate time to value through better delivery of applications and services. To achieve this, platform teams are focused on three areas to maximize their impact:

1. **Self-service** - Platform teams are empowering downstream app and service developers to choose from pre-approved platform architectures that meet a range of performance and cost profiles. This shortens the planning and approval process for new infrastructure.
2. **Automation and tooling** - Platform teams are seeking to automate everything. We see teams with the most automation reaching 10x higher scale (measured in infrastructure resources per FTE) when platform tooling and automation share the engineering process used by the app and services teams: the same source control management, the same build-test-deploy process, etc.
3. **Security and reliability** - Nothing slows value delivery more than security issues and downtime. Platform teams are adding more proactive testing, policy enforcement, automated scanning, and observability to their infrastructure projects so they can identify more issues before deployment.

– **JOE DUFFY**, CEO, Pulumi

Achieving elite status through a holistic software delivery practice

Can we identify high-performing engineering teams through numbers alone? This is the question we set out to answer every year in this report. Testing our recommended baseline engineering metrics against millions of data points from real development teams on the CircleCI platform, we see a clear picture of what it means to have an elite software delivery practice.

2022 was a year of big ups and big downs for the technology industry. Early in the year, we saw billion-dollar valuations and rampant hiring sprees. In the twilight of 2022, we saw stunted cash flows and mass layoffs impact companies of all sizes. With budgets shrinking and uncertainty in the air, organizations need to focus on people, processes, and culture to drive value for customers.

How can organizations make the most of their team's productivity while keeping engineers happy? With the rise and rapid freefall of the quiet quitting era, does developer happiness matter? What does it take to enable and maintain high velocity and high quality at scale? Exactly what does "good" look like in this environment?

THIS YEAR'S STATE OF SOFTWARE DELIVERY REPORT WILL AIM TO ANSWER: what do high-performing engineering teams look like quantitatively, and how do you achieve this level of performance through a holistic software delivery practice?

To answer these questions, this year's report covers the standard engineering metrics of success rate, throughput, duration, and mean time to recovery (MTTR), plus our recommended benchmarks for each. But the sections have been expanded to show how both technological and cultural factors combine to influence software delivery performance, offering a more comprehensive view of team success throughout the delivery lifecycle.

We've also included highlights on how platform teams can help organizations achieve elite status through a more scalable and reliable delivery process. Across the industry, the discipline of platform engineering is strongly influencing organizational outcomes through tooling decisions and cultural impact on team communication, collaboration, and behavior. In each of our four Platform Perspective sections, we offer specific guidance to platform teams to help them monitor, measure, and improve their team performance on key indicators of success.

What is an elite software delivery team?

Throughout this report, we frequently use phrases like “high performing” and “elite” to describe software teams that have achieved a high degree of success in their delivery practices. But what exactly does success mean in this context, and what criteria do we use to measure it?

WE DEFINE SUCCESSFUL TEAMS as those who consistently meet our benchmarks across the four metrics measured in this report: duration, throughput, mean time to recovery, and success rate. These metrics are adapted from the four key indicators of software delivery performance outlined in the influential 2018 book *Accelerate* by Google’s DevOps Research and Assessment (DORA) team, and our benchmarks for each metric largely correspond to the “elite” performance tier identified in DORA’s annual State of DevOps research report.

This study diverges from the DORA report in a few key ways: Whereas DORA uses survey data to analyze software delivery performance on an organization’s “primary application or service,” this report is based on actual performance data recorded across millions of workflow runs on CircleCI across all projects and all branches. This gives us an opportunity to validate, and sometimes challenge, the self-reported information in DORA surveys by looking at how teams actually build, test, deploy, and iterate upon every aspect of their codebase.

Building the *State of Software Delivery* report on CircleCI platform data also means that this study is tightly focused on how effective teams leverage continuous integration to deliver better software, faster. The metrics reflect this. For example, instead of lead time for changes, which measures how long it takes for a committed change to be successfully

deployed to production, we measure duration, or the average time required to complete any CI workflow, regardless of whether it results in a deployment to a production environment. This gives a fuller picture of team velocity and the critical feedback loops that occur earlier in the development lifecycle.

We also recognize that team performance involves a number of factors that can’t be measured by the four key metrics...

... and that there are many cultural, economic, and technological influences on a team’s ability to meet our recommended benchmarks. Moreover, software delivery performance matters only in the larger context of business performance. To that end, we have been careful to highlight organizational factors that play into each of the metrics and to encourage teams to set their own benchmarks based on their individual circumstances and priorities.

Ultimately, success is a subjective measurement, and every team must evaluate what it means to them. Our goal is to support this process by providing meaningful insights into how the world’s best engineering teams use continuous integration to unlock their full potential and drive key business outcomes with quality software.

History of continuous integration

The term “continuous integration” (CI) was first introduced by American computer scientist and software engineer Grady Booch in 1994. In his book, *Object-Oriented Analysis and Design with Applications*, Booch highlights the importance of the “micro process,” in which there are many more internal releases to the development team than there are external releases to end users.

This framework of thinking was born out of the Agile method of developing software, which prioritizes being responsive to change. Agile processes have helped teams become more comfortable with failure because they test and deliver software in smaller pieces more frequently, enabling them to catch bugs at earlier stages and prevent errors and downtime. Continuous integration optimizes for shorter delivery cycles and minimizes risk by breaking work into smaller pieces.

CI automation has also been critical in allowing development and operations teams to work together seamlessly, often under a unified DevOps role. By creating test suites that run on every change to the codebase, DevOps teams can innovate faster because those tests return a high degree of confidence. When enabled through CI, proper test coverage gives teams the ability to deploy at will and release working software any time with little effort.

When the macroeconomic environment tightens and the pressure is on for engineering teams to deliver, the benefits of confidence and speed that CI brings to the development process become non-negotiable. At the same time, the way we build software has dramatically changed over the past 10 years.

Our traditional notion of CI is that a test is triggered by a pull request: a developer makes a change in code, and our automation asks: ‘is that change good? Does it pass the tests we’ve written to validate that it creates the

desired outcome?’ And yet, with the rise of microservices architectures, third party dependencies and libraries, open source, and third party services, the vast majority of changes to your application are coming from the broader ecosystem. The number of changes driven by developers on your team changing your own repositories is dwindling.

Can CI still guarantee stability and reliability if we are testing only a fraction of changes made to an application?

How do we give engineering teams the freedom and flexibility to innovate, while managing the unrelenting drumbeat of change?

This is why many companies have adopted platform engineering – internal teams dedicated to building reliable toolchains and efficient, repeatable processes. To stay competitive, organizations need to combine the speed, agility, and empowerment of the DevOps revolution with the consistency, control, and cost management of enterprise-scale companies.

How do you compete?

How do you achieve elite status through a holistic software delivery practice? Before we begin to answer this question, let's cover a few key terms that will be important as we define the baseline engineering metrics we recommend for delivering software at scale.

CONTINUOUS INTEGRATION (CI): The automated building and testing of your application on every new commit.

CONTINUOUS DELIVERY (CD): A state where your application is always ready to be deployed. A manual step is required to actually deploy the application.

CONTINUOUS DEPLOYMENT: The automation of building, testing, and deploying. If all tests pass, every new commit will push new code through the entire development pipeline to production with no manual intervention.

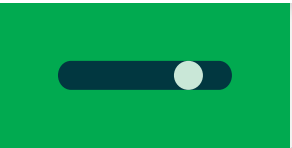

DURATION is the length of time it takes for a workflow to run.





THROUGHPUT is the average number of workflow runs per day.

MEAN TIME TO RECOVERY (MTTR) is the average time between a single workflow's failure and its next success.

SUCCESS RATE is the number of successful runs divided by the total number of runs over a period of time.

Measuring and then optimizing duration, throughput, mean time to recovery, and success rate gives teams a tremendous advantage over organizations that do not track these key metrics.



		MEDIAN PERFORMANCE	ELITE BENCHMARK
DURATION		3.3 minutes	10 minutes
MEAN TIME TO RECOVERY		64.3 minutes	< 60 minutes
SUCCESS RATE		77% on default branch	+ 90% on default branch
THROUGHPUT		1.52 times per day	On demand

Software delivery performance in 2023

Duration

Duration

BENCHMARK: 📌 5-10 MINUTES

MEDIAN PERFORMANCE: 3.3 MINUTES

Duration is the foundation of software engineering velocity. It measures the average time in minutes required to move a unit of work through your pipeline. Importantly, a unit of work does not always mean deploying to production – it may be as simple as running a few unit tests on a development branch. Duration, then, is best viewed as a proxy for how efficiently your pipelines deliver feedback on the health and quality of your code.

The core promise of most software delivery paradigms, from Agile to DevOps, is speed: the ability to take in information from customers or stakeholders and respond quickly and effectively.

These rapid feedback and delivery cycles don't just benefit an organization's end users; they are crucial to keeping developers happy, engaged, and in an uninterrupted state of flow.

Yet an exclusive focus on speed often comes at the expense of stability. A pipeline optimized to deliver unverified changes is nothing more than a highly efficient way of shipping bugs to users and exposing your organization to unnecessary risk.

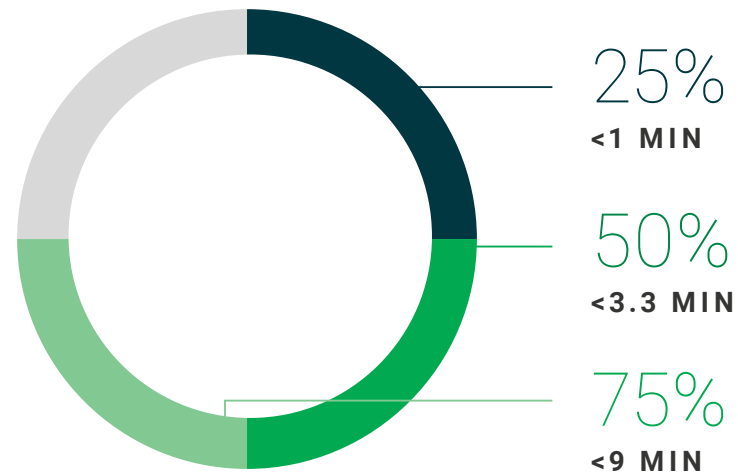
To be able to move quickly with confidence, you need your pipeline to guard against all potential points of failure and to deliver actionable information that allows you to remediate flaws immediately, before they reach production.

The only way to achieve this level of productive feedback is to implement comprehensive testing at all stages of your pipeline. The optimal pipeline duration is the shortest length of time required to run a suite of tests robust enough to confirm your code is free from defects and security vulnerabilities.

What is the ideal duration?

TO GET THE MAXIMUM BENEFIT FROM YOUR WORKFLOWS, we recommend aiming for a duration of 10 minutes, a widely accepted benchmark that dates back to Paul M. Duvall's influential book *Continuous Integration: Improving Software Quality and Reducing Risk* (2007). At this range, it is possible to generate enough information to feel confident in your code without introducing unnecessary friction in the software delivery process.

WHAT SHOULD YOUR WORKFLOW DURATION BE?



- 25% of workflows complete in under a minute.
- 50% of workflows complete in under 3.3 minutes.
- 75% of workflows complete in less than 9 minutes.

WHAT SHOULD YOU AIM FOR?

Balance speed of completion with meaningful signal:
do fast workflows matter if they don't have any tests?

DURATION

What the data shows

Among the workflows observed in our dataset, 50 percent completed in 3.3 minutes or less, far below our 10 minute benchmark and nearly 30 seconds faster than we saw in 2022. The fastest 25% of workflows completed in under a minute, and 75% of all workflows completed in under 9 minutes. The average duration was approximately 11 minutes, reflecting the influence of longer-running workflows at the edges of the dataset: workflows in the 95th percentile require 27 minutes or more to complete.

Why does the typical workflow duration clock in well under our recommended benchmark? Simply put, many teams are still biased toward speed rather than robust testing. The number one opportunity we've identified for improving performance across all four software delivery metrics is for organizations to enhance their test suites with more robust test coverage.

Some ways you can improve test coverage include:

- Adding unit tests, integration tests, UI tests, and end-to-end tests across all layers of your application.
- Incorporating code coverage tools into your pipelines to identify parts of your code base that aren't being adequately tested.
- Including static and dynamic security scans to catch vulnerabilities throughout your software supply chain.
- Incorporating test-driven development practices by writing tests during the design phase of your development process.



"When security testing is easy to integrate and use by **being fast, accurate, and informative**, companies are more likely to incorporate it into a robust testing strategy, helping to improve software quality and the organization's risk posture. We find risk reduction and ease of use go hand-in-hand as top considerations enterprise customers cite when evaluating the success of their security tooling.

By analyzing the build logs from Snyk's CircleCI orb when we "break the build", it's both informative for security teams and actionable to developers, as they are responsible for fixing any issues. By surfacing the same information that would break the pipeline in the developer's IDE, developers can proactively address issues, preventing broken builds, and therefore reducing interruptions to pipelines. With this model, developers become a first line of defense, proactively reducing their organization's risk exposure and providing security teams confidence that security tests are happening during delivery."

– **TOMAS GONZALEZ BLASINI**, Partner Solutions Engineer, Snyk

Why should teams invest in robust testing?

While these changes may result in longer durations, they are hallmarks of high-performing organizations. Once you have implemented a testing process that allows you to consistently deliver secure and reliable code, the next most important step is to maximize the efficiency of your pipelines.

To identify ways in which you can improve durations in your organization, consider both the cultural and technological influences.



Cultural influences on duration include:

- Collaboration and trust: open, collaborative teams make smaller, more frequent commits to a shared mainline, which can shorten merge processes and reduce merge conflicts.
- Test-driven development practices: a test-first approach to software design ensures the right balance of quality and velocity.
- Failure tolerance: organizations that view failed tests as valuable feedback rather than signs of poor performance are likely to have more extensive test coverage. More testing often means more failures, but smaller ones with faster remediation and less operational impact.



Technological influences on duration include:

- Test coverage: more tests mean longer duration but higher quality, preventing costly and time-consuming outages in production.
- Application complexity: complex distributed applications with many dependencies require longer, more extensive tests than compact monoliths.
- Pipeline complexity: the number and types of jobs performed in your CI workflows, as well as your use of optimizations like caching and parallelism, will strongly influence duration.

AS WITH ALL THINGS IN DEVOPS, IT IS IMPORTANT TO ADDRESS ANY CULTURAL OBSTACLES BEFORE YOU TACKLE TECHNOLOGICAL CHALLENGES.

Balancing workflow speed and test coverage is ultimately a matter of prioritization: which features are part of the critical path? Where can you afford more experimentation and risk?

Once you have buy-in from important stakeholders on how to best balance test coverage and workflow speed, you can optimize your workflow duration with these techniques:

- Use test splitting and parallelism to execute multiple tests simultaneously across separate compute nodes.
- Cache dependencies and other data to avoid rebuilding unchanged portions of your application on successive runs.
- Use Docker images that are custom made for use in a CI environment to reduce spin-up times and ensure stable, deterministic builds.
- Evaluate the resource requirements for your workflows and choose the right machine size for your needs

The path to optimizing your workflow durations is to combine comprehensive testing practices with efficient workflow orchestration. Teams that focus solely on speed not only spend more time rolling back broken updates and debugging in production but also face greater risk to their organizational reputation and stability.



DURATION

Platform perspective

DEVELOPERS WILL NATURALLY GRAVITATE TOWARD SPEED. And while platform teams are tasked with identifying and eliminating impediments to developer velocity, that is not their only mandate. Another, perhaps more important, responsibility of the platform engineer is to set guardrails and enforce quality standards across projects.

To ensure every project in your organization gets the right amount of test coverage with the least amount of impact on duration, you should partner with your product engineering groups to determine how to best integrate various testing concerns across the stages of your development cycle. Certain tests, like unit tests and code quality scans, may need to run on every commit to the development branch, while longer running tests like SAST and DAST scans or user acceptance tests may need to run less frequently, such as on merges to QA branches or during nightly builds.

To encourage best practices and optimize your durations, use shareable configuration templates and configuration policies so that each team has easy access to the tools and optimizations that deliver the best results.

Another valuable contribution that platform teams can make to pipeline durations is to foster a culture in which failed pipelines are welcomed – as long as they fail fast.

Make sure all default pipeline templates include separate test environments for development, staging, and QA builds, increasing the comprehensiveness of your test suites the closer you get to production.

That way, developers can get immediate feedback on their commits and you can be sure your application has been thoroughly tested for all functional and nonfunctional requirements before it ships to users.

Finally, it is important to actively monitor pipeline durations across your organization and prioritize optimizations that will have the largest business impact. A test runs longer when it's poorly written, needs heavy resources, or because it is under-optimized. Consider whether sending the test back to the development team for refactoring will yield improvements that outweigh the labor and opportunity costs. In many cases, allocating more compute or concurrency can deliver a faster signal and save you valuable developer minutes.



Mean Time to Recovery

Mean Time to Recovery

BENCHMARK: 🟩 60 MINUTES

MEDIAN PERFORMANCE: 64 MINUTES

Mean time to recovery measures the average time required to go from a failed build signal to a successful pipeline run. This metric is indicative of your team's resilience and ability to respond quickly and effectively to feedback from your CI system.

Mean time to recovery is the best indicator of your organization's DevOps maturity.

From an end-user perspective, and for most organizations, nothing is more important than your team's ability to recover from a broken build. While customers may not notice the steady stream of granular updates that you ship throughout the week, you can be sure that they will notice when your application goes offline after broken code slips through your test suite.

The good news is that, if you've done the work to set up a pipeline that provides a complete picture of your code health and any potential failure points, the burden of getting back to a deploy-ready state is significantly lowered. Diagnosing the failure and implementing a fix becomes a matter of evaluating test output and correcting or reverting defects rather than embarking on an endless bug hunt.

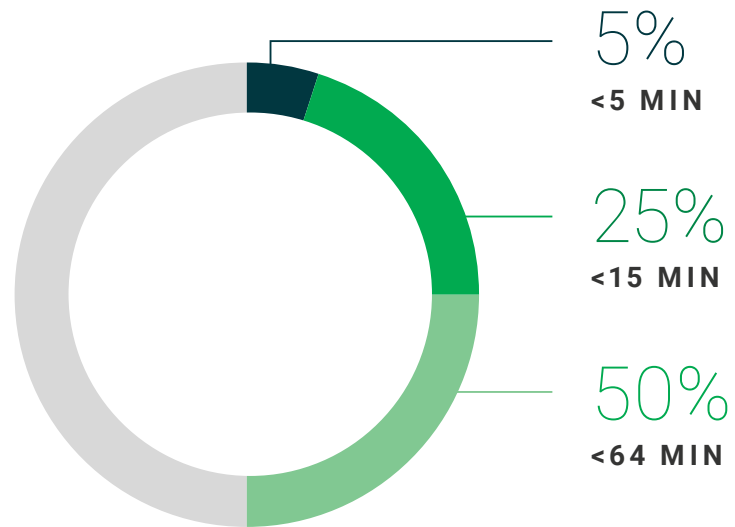
What is the ideal mean time to recovery?

In a 2006 blog post, Martin Fowler described the north star for software teams' MTTR: "Fix broken builds immediately." Does this mean your team should aim for resolving failed workflows within a matter of seconds? Or better yet, avoid build failures at all costs?

Not at all. Broken builds happen, and with proper tests in place, the information from a red build has as much (if not more) value for development teams as a passing green build. What is most important is that your organization makes resolving failed workflows its top priority, particularly when a failure occurs on the default branch. After all, the goal of continuous integration is to keep your mainline code in a consistently releasable state.

For this reason, we recommend that you aim to **fix broken builds on any branch in under 60 minutes**. Depending on your branching strategies, the scale of your user base, and the criticality of your application, your target recovery time may be significantly lower or higher. However, the ability to recover in under an hour is strongly correlated with other indicators of high-performing teams and will allow your organization to avoid the worst outcomes of prolonged failures.

WHAT ARE THE BENCHMARKS FOR MEAN TIME TO RECOVERY?



- The fastest 5% of workflows recovered in <5 minutes.
- 25% of all workflows recovered in <15 minutes.
- 50% of all workflows recovered in 64 minutes.

WHAT SHOULD YOU AIM FOR?

Faster recovery is better. And remember: duration is the foundation of time to recovery. You cannot recover faster than your workflow runs.

MEAN TIME TO RECOVERY

What the data shows

On CircleCI, 50% of workflows recovered in 64 minutes or less, very nearly equal to our benchmark of 60 minutes. This is a significant improvement from our previous two reports, which showed median times of 73 and 74 minutes on all branches, and is the most notable change among any of the four metrics observed in this year's data.

What is driving this marked decrease in recovery times?

WE SUGGEST THERE ARE TWO FACTORS AT PLAY:

1. Economic pressures in the macro environment, coupled with rising competition in the micro environment, have motivated teams to add stronger guardrails that enhance stability and reliability without disrupting innovation.
2. High performers' increasing reliance on platform engineering teams to achieve steadier and more resilient development pipelines with built-in recovery mechanisms.

THIS RENEWED FOCUS ON RESILIENCE in the software delivery pipeline is most evident among our highest performers: the top 25% recovered in 15 minutes or less, and the top 5% recovered in under 5 minutes. But even teams at the lower end of the spectrum were able to recover in less than a day, with the 75th percentile achieving green builds within 22 hours on average from the previous failed run.

As with duration, both cultural and technological factors play into a team's ability to meet or exceed the benchmark on recovery times.



Cultural influences on MTTR include:

- Intent to resolve: the extent to which your organization prioritizes resolving broken builds will directly affect your recovery times.
- DevOps maturity: teams that use DevOps best practices like trunk-based and test-driven development are better positioned to resolve errors quickly.
- Geographical distribution: organizations in which all developers work in the same location and during the same hours are more likely to leave a build broken overnight or over the weekend than those with distributed team members working across different time zones.



Technological influences on MTTR include:

- Test coverage and error reporting: accurate and verbose error reporting makes debugging easier.
- Workflow duration: the fastest you can possibly recover from a failure is the time required to complete your next workflow run.
- Commit size: smaller, more frequent commits make it easier to find the source of failure and remediate quickly.

THE FIRST STEP TO LOWERING RECOVERY TIMES is to treat your default branch as the lifeblood of your project – and by extension, your organization. While red builds are inevitable, getting your code back to green immediately should be everyone's top priority. With a vigilant testing culture in place, your organization will be poised to leverage information from your CI pipeline and remediate failures as soon as they arise.

Bear in mind that recovery speed is inextricably bound with pipeline duration: the shortest possible time to recovery is the length of your next pipeline run.

To achieve faster recovery times, first optimize your duration using the techniques in the previous section, then, add the following:

- Set up instant alerts for failed builds using services like Slack, Twilio, or PagerDuty.
- Write clear, informative error messages for your tests that allow you to quickly diagnose the problem and focus your efforts in the right place.
- SSH into the failed build machine to debug in the remote test environment. Doing so gives you access to valuable troubleshooting resources, including log files, running processes, and directory paths.



MEAN TIME TO RECOVERY

Platform perspective

PLATFORM TEAMS OFTEN FORM THE BRIDGE between an organization's business goals and its software delivery practices. Equipping developers to recover from broken builds quickly can have a significant impact on an organization's bottom line in terms of both developer productivity and customer satisfaction.

Platform engineers can improve mean time to recovery in several ways. Begin by emphasizing the value of a deploy-ready default branch across all projects, and establish clear processes and expectations for failure recovery across your projects. Set up effective monitoring and alerting systems to quickly detect broken builds, notify the responsible teams, and track recovery times.

You can also limit the frequency and severity of broken builds by using controls and config policies that prevent unreviewed changes to critical workflows and build environments.

Likewise, configuration as code and infrastructure as code tools can help limit the potential for manual misconfiguration errors.

Finally, sophisticated deployment strategies like feature flags, blue-green deploys, and canary deploys can help limit the blast radius when a broken build leads to a production failure and allow you to immediately roll back to the last known good release.



Success Rate

Success Rate

BENCHMARK: 🟢 90% OR BETTER ON THE DEFAULT BRANCH

MEDIAN PERFORMANCE: 77% ON THE DEFAULT BRANCH

Success rate is the number of passing runs divided by the total number of runs over a period of time. It is another indicator, alongside mean time to recovery, of the stability of your application development process. However, the impact of success rate on both customers and development teams can vary according to a number of factors. Did the failure occur on the default branch or a development branch? Did the workflow involve a deployment? How important is the application or service being tested?

A failed signal is not necessarily an indication that something has gone wrong or that there is a problem that needs to be addressed on a deeper level than your standard recovery processes. Far more important is your team's ability to ingest the signal quickly (duration) and remediate the error effectively (MTTR).

In fact, there are many scenarios in which a broken build might be tolerated or even welcomed. For example, it might indicate that your team is working on a particularly challenging or innovative feature and is iterating its way toward success. Or it could be the result of adopting test-driven development, a fail-first approach in which developers write tests according to design requirements and add code in successive increments until the tests pass.

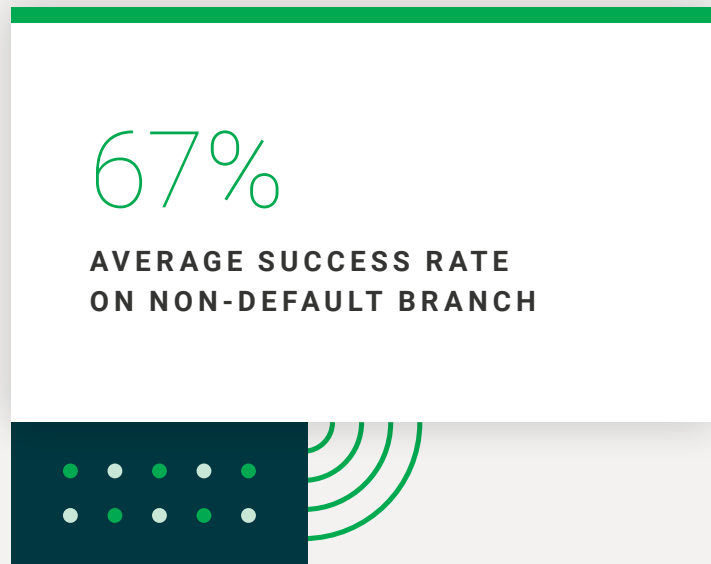
A failed workflow that delivers a fast, valuable signal is far more desirable than a passing workflow that offers thin or unreliable information.

What is the ideal success rate?

On the default branch, we recommend maintaining a **success rate of 90% or higher**. This is a reasonable target for the mainline code of a mission-critical application, where changes should be merged only after passing a series of well-written tests.

Failures on topic branches are generally less disruptive to software delivery than those that occur on the default branch. Therefore, we've focused our recommendation only on the primary branch used to house deployable code. Your team may want to set its own benchmark for success on various other branches depending on how you use them.

Keep in mind, though, that long-lived topic branches are considered an antipattern due to the increased risk of merge conflicts. To maximize your success rates on the default branch, it is best to use feature branches only for short-term experimentation, and to make frequent, small commits to the main line.



SUCCESS RATE

What the data shows

Among CircleCI users, success rates on the default branch were 77% on average. On non-default branches, they were 67% on average. Success rates on the default branch have held steady over the past several iterations of our report.

While neither number reaches our benchmark of 90%, the pattern of non-default branches having higher numbers of failures indicates that teams are utilizing effective branching patterns to isolate experimental or risky changes from critical mainline code. And while success rates haven't moved much over the history of this report, recovery times have fallen sharply. This is a welcome sign that organizations are prioritizing iteration and resilience over momentum-killing perfectionism.

Teams that want to improve their success rate without negatively affecting productivity should be mindful of both cultural and technological influences.



Cultural influences on success rate include

- Postmortem handling: how thoroughly your team reviews outages, and the preventative measures you put in place to avoid them in the future, can strongly influence how often failures occur (and reoccur).
- Trust and collaboration: high-trust, collaborative cultures will be more likely to experiment and iterate openly on shared branches rather than siloing work locally.
- Team experience and organizational knowledge: teams with high levels of individual and collective experience are more likely to follow proven, effective patterns and avoid introducing errors into the codebase.



Technological influences on success rate include

- Test coverage and quality: flaky tests and insufficient test coverage allow for more frequent outages.
- Application complexity: complex distributed systems with many dependencies are more fragile and prone to unanticipated breakages.
- Environment consistency: testing on environments that mirror your production infrastructure will reduce the likelihood of errors slipping into production.

The importance of success rate to your team will depend largely on how closely your team collaborates, the type of work you are doing in your pipelines, and your ability to recover quickly from failures. Prioritizing resilience from both a cultural and technological standpoint will help mitigate the effects of low success rates.



SUCCESS RATE

Platform perspective

It can be tempting as an organization to chase a success rate of 100% or to interpret high success rates as a sign of elite software delivery performance. And while it is desirable to maintain high rates of success on default branches, this metric means very little if you are not confident in the signal you are receiving from your CI system. As a platform engineer, you have a responsibility to look beyond surface-level metrics and uncover the most meaningful data about team performance.

If success rates in your organization seem low, look at your MTTR.

Focus your efforts on shortening recovery time first. Since this often involves encouraging smaller commits and more effective tests to mitigate future failures, your success rates, particularly on the default branch, are likely to improve as a result.

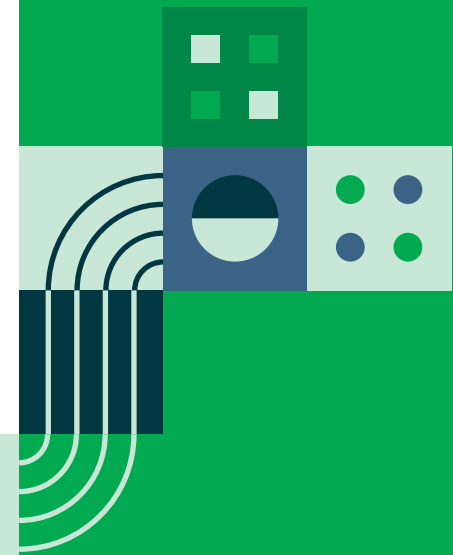
As with other metrics, it is important to be able to accurately capture data on success rates across your various projects. Set a baseline and aim for continuous improvement. If your success rate declines or seems erratic, look left in your process: causes may include flaky

tests or gaps in test coverage, discrepancies between test environments and production environments, or human-centric challenges like alert fatigue and information overload.

Finally, be mindful of patterns and the influence of external factors. Does your team's success rate decline on Fridays? Around the holidays? Could burnout or organizational tumult play a role in the quality of code being produced? Investigating and addressing these non-technical influences could yield far greater results than looking at technical factors alone.

A team that is responsive to red builds and can get back to a deploy-ready state quickly is better positioned than a team that rarely breaks the build but is slow to recover when they do.

Throughput



Throughput

BENCHMARK: 📌 VARIES ACCORDING TO YOUR BUSINESS NEEDS

MEDIAN PERFORMANCE: 1.52 TIMES PER DAY

Throughput is the average number of workflow runs, successful or otherwise, that an organization completes on a given project per day. Traditionally, this reflects the number of changes your developers are committing to your codebase in a 24-hour period, but new automatic triggers unrelated to developer activity (such as a change in infrastructure health or a new version of an upstream dependency) may emerge as CI systems continue to evolve.

Throughput is useful as a measurement of team flow as it tracks how many units of work move through the CI system.

When performed at or above recommended levels, throughput puts the “continuous” in continuous delivery — the higher your throughput, the more frequently you are performing work on your application.

Of course, throughput tells you nothing about the quality of work you are performing, so it is important to consider the richness of your test data as well as your performance on other metrics such as success rate and duration to get a complete picture. As with duration, a high throughput score means little if you are frequently pushing poor quality code to users.

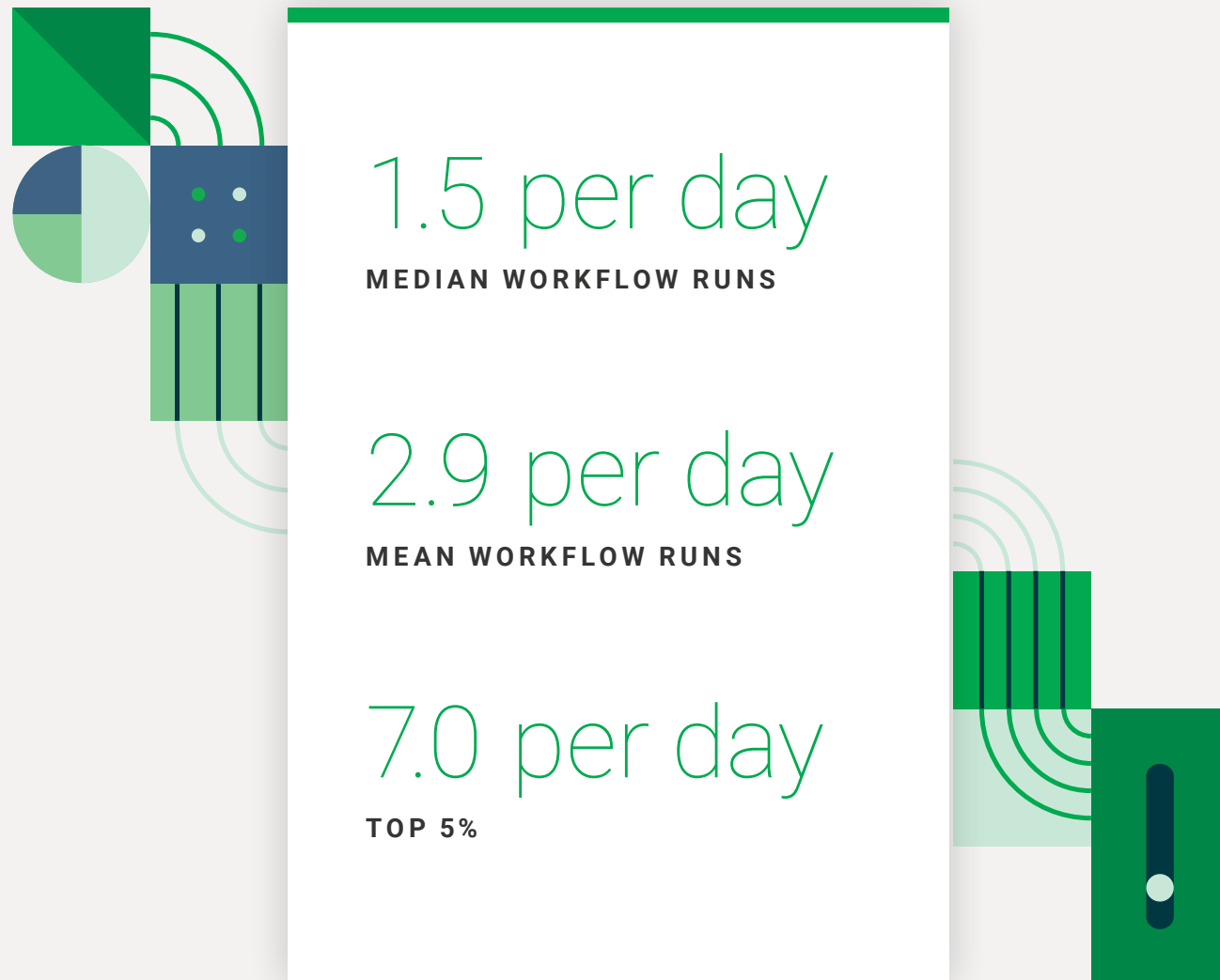
What is the ideal throughput?

Of all the metrics, throughput is the most subjective to organizational goals. A large cloud-native organization actively developing a critical product line will require far higher levels of throughput than a small team maintaining legacy software or niche internal tooling. Accordingly, there is no universally applicable throughput target, and **we suggest setting your own benchmark according to your internal business requirements.**

Traditionally, DevOps practitioners recommend that every developer aim to commit to the default branch at least once per day. This practice ensures commits remain small enough that breaking changes are relatively uncomplicated to resolve. It also helps development teams maintain momentum and fosters a collaborative spirit in comparison to work being siloed in long-lived topic branches and punctuated by infrequent, high-risk merge events.

Is once per day the right target for your project? It depends on the type of work you’re doing, the resources you have available, and the expectations of your end users. Far more important than the volume of work you’re doing is the quality and impact of that work.

Thoroughly testing your code and keeping your default branch in a deploy-ready state ensures that, regardless of when or how often changes are pushed, you can be confident they will add value to your product and keep your development teams **focused on tomorrow’s challenges rather than yesterday’s mistakes.**



THROUGHPUT

What the data shows

The median workflow in our data set ran 1.54 times per day, a slight increase from 1.43 times per day in 2022. At the upper end of the spectrum, the top 5% of workflows ran 7 times per day or more, on par with what we've seen in previous years. Overall, the average project had 2.93 pipeline runs per day in 2023 compared to 2.83 in 2022.

This uptick in productivity is not especially notable, but when taken in combination with the sharp decrease in MTTR discussed earlier in the report, it may show that teams are committing to smaller, more frequent changes to limit the complexity of outages and achieve faster, more consistent feedback on the state of their applications.

THROUGHPUT

Changes in throughput can stem from a number of factors in your organization, both cultural and technological.



Cultural influences on throughput include:

- Team size, structure, and experience: fluctuations in the size of your development team will naturally affect your organization's production capacity.
- Priorities and expectations: shifts in your business priorities may require shipping more or fewer updates on a given project.
- Morale and motivation: internal and external motivations, incentives, and rewards can have outsized effects on team productivity.



Technological influence on throughput include:

- Application scope and complexity: the more difficult your application is to understand and explain, the slower and more deliberate your team is likely to be in making changes.
- Tech stack: a modern, integrated, easy-to-use toolchain can greatly increase team throughput, while outdated or unfamiliar technologies will impair productivity.
- Pipeline effectiveness: an unoptimized CI pipeline can add friction that has a compounding effect on team throughput.

Of all the metrics measured in this report, throughput is the most dependent on the others.

How long your workflows take to complete, how often those workflows fail, and the amount of time it takes to recover from a failure all affect your developers' ability to focus on new work and the frequency of their commits. To improve your team's throughput, first address all the potential underlying factors that can affect team productivity.

Throughput is often a trailing indicator of other changes in your processes and environment. Rather than setting an arbitrary throughput goal, set a goal that reflects your business needs, capture a baseline measurement, and monitor for fluctuations that indicate changes in your team's ability to do work.

Achieving the right level of throughput means staying ahead of customer needs and competitive threats while also continuously validating the health of your application and development process.



THROUGHPUT

Platform perspective

Platform engineering exists primarily to remove friction from development processes and to abstract complexity in service of clearly defined value paths. An effective platform will enable developers to do more work, with more impact, in less time. Thus, throughput is often one of the top indicators used to evaluate the performance of not just the development team but of the platform team as well.

It is important that your throughput goals map to the reality of your internal and external business situation.

What expectations do your customers and business leaders have? What does the competitive landscape look like? How complex is your codebase? What resources do you have available? How mature are your delivery processes? When it comes to throughput, “good” is relative, and setting the right target requires clear-eyed introspection.

TO GET THE MOST VALUE from your throughput measurements, capture a baseline, then monitor for deviations. Consider measuring on a per-developer average to control for changes in the size of your workforce. A sharp increase or decrease in throughput warrants deeper investigation into the root causes. Use the other metrics at your disposal to gain more context into the change. A simultaneous decline in success rates, for example, may indicate that your team is struggling with a particularly challenging implementation detail that your platform can help solve.

To help keep your developers happy and on track, seek to alleviate as much cognitive load from their day-to-day work as possible. Standardize error-prone tasks like infrastructure deployment using declarative, code-based solutions, and be ruthless in automating away repetitive manual processes. Smoothing even the smallest bit of friction will pay compounding interest on your team’s ability to achieve a productive state of flow.

Additional findings



“Gaining visibility into every individual test has helped us proactively detect and fix flaky tests. Utilizing test visualization tools like distributed tracing alongside our weekly flaky test audits has decreased the number of our flaky tests by 88%. Ultimately, building fast and stable pipelines allowed our developers to work faster and more efficiently, delivering higher-quality code and reducing operational costs.”

– **WILLIAM MCMULLEN**, Product Marketing Manager, Datadog

AN INTERVIEW WITH DATADOG | SUCCESS THROUGH ROBUST TESTING

How Datadog achieves success

IN SOFTWARE DEVELOPMENT VELOCITY, BOTH SPEED AND QUALITY ARE CRITICAL.

When we push updates to our product, there should be no surprises or undesired user behavior. We should be able to push updates as expected both quickly and consistently.

When tracking speed and quality using metrics, the most essential thing to aim for is low MTTR. Robust test coverage and verbose error reporting will help your team meet this goal confidently.

As a first step, teams need visibility into how they are performing. Establishing a baseline for how your team is currently performing and then comparing it to the industry standard is a good place to start.

There are many metrics to observe and KPIs that can be formed from them, but the most impactful KPIs are derived by combining duration, time to recovery, throughput, and success rate.

Engineers and engineering managers should use metrics dashboards aimed at tracking the health and performance of their software delivery practice. These will be your core KPIs. When it comes to how quickly your team can recover from failure, you should be asking these questions:

- Do I keep my engineers in flow?
- Can they resolve issues quickly?

Engineering managers and business leaders should track KPIs that monitor value added, like uptime, a combination of duration, recovery time, and success rate. When it comes to the stability of your platform, you should be asking these questions:

- Can I respond to new business needs quickly?
- Can I deploy a security patch or recover from an outage quickly?

At Datadog, maintaining a robust testing strategy has helped us accelerate engineering velocity by fixing slow, unstable pipelines. Implementing Datadog CI Visibility into our own CI/CD workflows has helped us gain deep visibility into all branches, enabling our platform and product development teams to identify high-failure pipelines and slow jobs to continuously improve execution time. Small changes over time and scale have helped us reduce MTTR and improve time to deploy.

How do team size and company size affect delivery performance?



TEAM SIZE

The size of your development team can have a meaningful impact on your engineering performance. Our most recent data shows that duration, throughput, and time to recovery all continue to increase until teams reach about 100 contributors, at which point duration and recovery time begin to fall while throughput remains steady. In other words, the largest teams are faster and more responsive, while remaining just as productive as their mid-sized counterparts with 20 to 99 engineers. One explanation for this is that, when most organizations reach 100 engineers per team, they begin to centralize and consolidate tooling and process decisions for better efficiency and control, often under the guidance of a platform engineering team.

When companies are young and have only a few engineers per team, their durations tend to be short (under 2 minutes on average) because they're likely prioritizing speed above any other metric. Once teams hit 10 contributors, we see their duration start to increase as they begin doing more within each pipeline, like implementing better testing and security scans. Duration peaks at just under 6 minutes on average for teams with 51 to 100 engineers. Once teams exceed 100 contributors, duration decreases to around 5 minutes as they begin to standardize their processes for increased efficiency.



COMPANY SIZE

When it comes to company size (inclusive of all business functions, not just engineering), we see different patterns across our delivery metrics than we do when focused only on engineering team size. Contrary to what you might expect, delivery performance is not necessarily correlated with a larger company size.

In our data, the only metric that is positively correlated with company size is throughput: the larger the organization, the more capacity it has to push work through the pipeline.

However, the size of your engineering footprint and the role engineering plays in your organization will play a strong role in how software teams view and respond to other key performance indicators. In a tech-forward company, engineering teams are an important value driver and are treated as a profit center. Often these teams make up a larger percentage of overall company headcount and are well-resourced. In companies where software is not part of the core product mix or plays only a supporting role, engineering teams are viewed as a cost center and receive less institutional support. This can have a significant effect on engineering productivity and performance.

Our data shows that engineering teams in the IT sector, where software is a key value driver, perform on par with our global averages: durations are **3.4 minutes**, **throughput stands at 1.56 workflows run**, and **time to recovery stands at 1 hour and 8 minutes**. In contrast, teams in the automotive, retail, and insurance verticals, where software plays a less prominent role in the overall product mix, are markedly less responsive to outages, **with times to recovery at or above 4 hours**.

Engineering productivity analysis

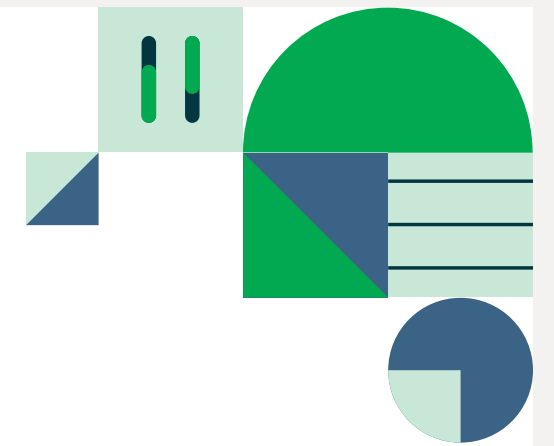
PRODUCTIVITY, EFFICIENCY, AND COST CONTROLS ARE TOP OF MIND for software delivery organizations in 2023. Inflation, rising interest rates, tightening global equity markets, and falling valuations have drastically shifted focus away from growth and toward profitability.

Of particular concern for cost-conscious organizations has been the global decline in productivity since the onset of the pandemic in 2020. In the United States, for example, productivity fell by 7.4% in Q2 2022, the largest such drop on record. Some business leaders have speculated that inflated budgets, intense demand for labor, and lax work-from-home policies were the driving force behind these drops. But does the data bear this out?

In the 2020 version of this report, we found that software delivery teams were largely able to maintain and even improve productivity through the initial months of the pandemic. In the 2022 version, we found that the end-of-year holiday season was responsible for the most notable declines in team productivity and responsiveness. In this year's report, we expanded our research to test the impact of major cultural events — from public holidays to consumer shopping events to the year's most eye-catching headlines — on developer activity on our platform.

Overall, we saw little to no productivity impact for most of the events we measured.

Almost all declines in platform activity were directly attributable to major public holidays, predictable events that give organizations plenty of time to prepare.



THIS DATA IS GOOD NEWS FOR SOFTWARE DELIVERY ORGANIZATIONS

Developer productivity is very predictable with a calendar and is largely unaffected by external events.

When you see big drops that aren't related to holidays, the cause is something internal. What is happening within your organization to make productivity decline?

In an era where stability and reliability are paramount, development systems have demonstrated a high degree of resilience in the face of major news stories and events. And with this year's data showing teams achieving the highest levels of throughput and the shortest recovery times we've ever measured, it's clear that teams leveraging automation and continuous integration and delivery are well positioned to survive and thrive.

HERE ARE SOME OF THE KEY RESULTS FROM OUR RESEARCH

The largest productivity declines were concentrated around public holidays.

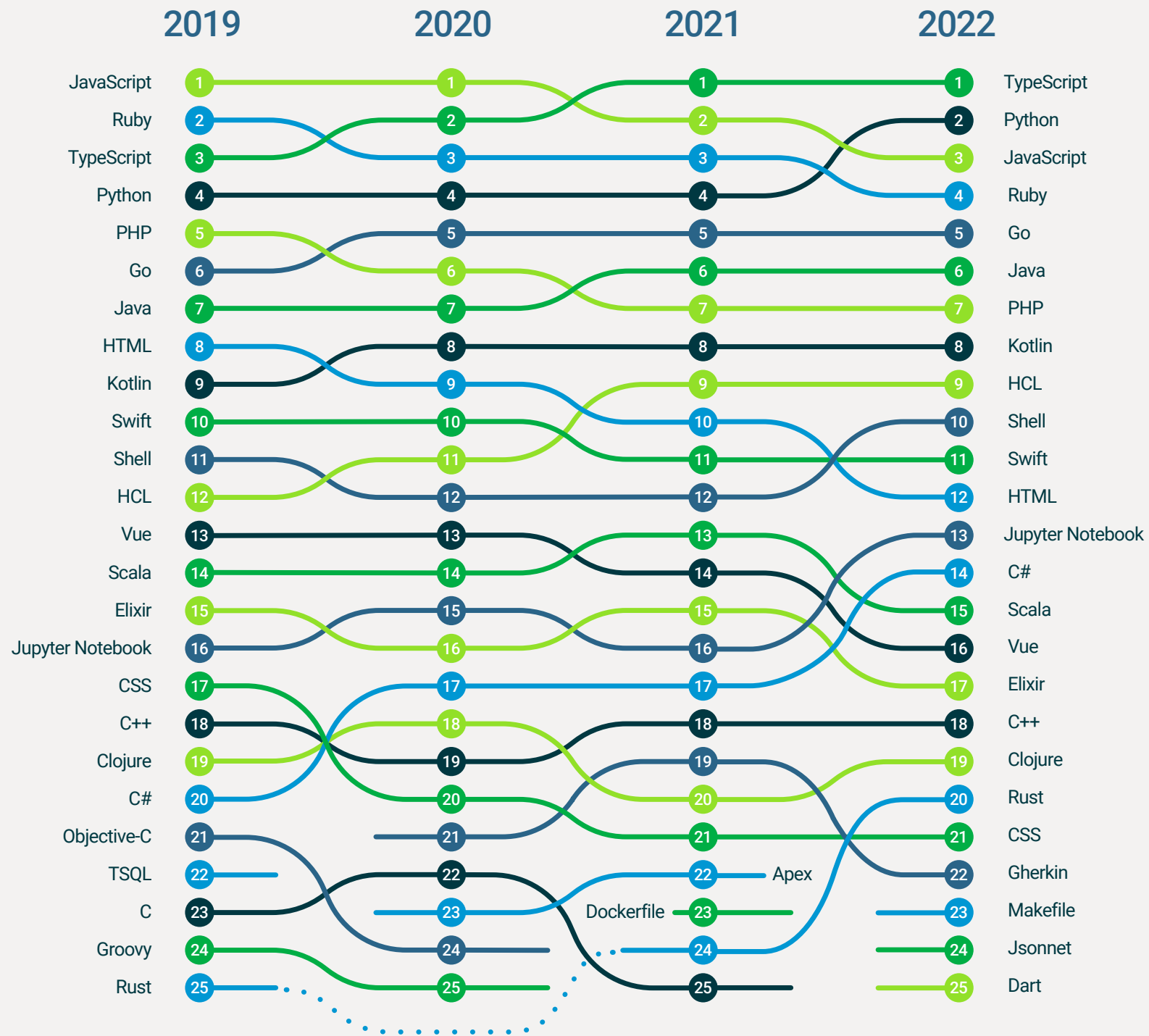
Confirming last year's findings, the largest activity decline we observed on our platform occurred on Monday, December 26, 2022 (Christmas Day observed). Developer activity fell 68.8% globally, and by more than 90% in both the US and the UK. We also saw a 40% global drop in activity on Friday, April 15, a day in which the religious holidays of Good Friday, Passover, and Ramadan overlapped for the first time in 30 years. Some regional holidays predictably had more localized effects, including Thanksgiving and Independence Day, which decreased activity in the US by approximately 85% and globally by about 30%.

Events of major national significance also resulted in localized productivity drops.

Perhaps the biggest news story of 2022, Russia's invasion of Ukraine on February 24, 2022 led to an 82% decline in activity in Ukraine as well as related declines in surrounding countries such as Poland (64.6%) and Finland (41.7%). Globally, activity fell by just 6.4%. The passing of Queen Elizabeth II on the afternoon of September 8 led to a 9.4% decrease in activity in the UK and an 11.3% decrease in Ireland the following day, while the Queen's funeral on September 19 (declared a national holiday in the UK) led to an 82.9% drop in UK activity but only 19.9% globally. The assassination of former Prime Minister of Japan Abe Shinzō corresponded to a 12.9% drop in activity in Japan, with no significant global impact.

Politics, tech and cultural events, and major shopping days had no real impact on developer activity.

Political events such as the overturning of Roe v. Wade in the United States on June 4, the resignation of Liz Truss as Prime Minister of UK on October 20, and the US congressional elections on November 8 showed either no measurable local impact or slight increases in developer activity. (For comparison, the US presidential election of 2020 corresponded with a 19.3% decrease in activity in the US.) Likewise, events such as Elon Musk's acquisition of Twitter and the Cyber Monday and Amazon Prime Day shopping events saw either flat or slightly elevated rates of activity.



Language trends

Your organization's choice of programming language can affect all aspects of your software delivery process, from developer productivity and happiness to application performance and maintainability.

While trends in language usage are unlikely to spur wholesale migrations among established teams, they can provide useful perspective on the tools and technologies being adopted to help solve emerging software delivery challenges – challenges your team will undoubtedly face as you build the applications and services of the future.

	Duration	MTTR	Success Rate	Throughput
1	Makefile	Gherkin	Mustache	Hack
2	LookML	JavaScript	Perl	Jsonnet
3	Shell	PHP	Smarty	Dart
4	HCL	HCL	Go	Swift
5	Mustache	Go	PL/pgSQL	Elixir
6	Nix	Ruby	HCL	Ruby
7	SaltStack	TypeScript	Vue	Mustache
8	Open Policy Agent	Perl	Scala	Jupyter Notebook
9	Smarty	Python	Makefile	TypeScript
10	Dockerfile	HTML	Elixir	Python
11	Jsonnet	Java	Shell	Elm
12	Batchfile	Clojure	HTML	Liquid
13	Liquid	CSS	Jupyter Notebook	Haskell
14	VCL	Elixir	Rust	Starlark
15	EJS	Vue	RobotFramework	PL/pgSQL
16	Jinja	Shell	C#	Jinja
17	PLSQL	Kotlin	Python	Lua
18	PowerShell	C#	Clojure	HTML
19	SCSS	Rust	TypeScript	Clojure
20	Haml	Dart	Ruby	Apex
21	R	Jupyter Notebook	Jinja	XSLT
22	CSS	Jinja	C	Perl
23	Python	PL/pgSQL	PHP	C++
24	C#	C	Kotlin	PureScript
25	Vue	C++	Dockerfile	Gherkin

Unsurprisingly, most of the languages with the fastest workflow durations are scripting or configuration languages with few build steps and less rigorous testing requirements. Python, the only language to appear in the top 25 for all four metrics, benefits from a language-specific Docker image custom built for speed and efficiency on CircleCI.

Many of the languages with the fastest recovery times have a long history in the developer community and also appear near the top of our most popular languages list. With maturity and popularity comes better tooling, documentation, and active community support, making debugging failed builds faster and less burdensome.

Many of the languages with the highest success rates also have the shortest durations, suggesting that they succeed most often due to low testing. Languages like Perl, Go, and Vue offer convenient package managers that make it easy for developers to access reliable, battle-tested modules and libraries that reduce the likelihood of errors.

Hack, a PHP superset focused on developer speed, has shown up at the top of our throughput rankings for two years running. Dart and Swift, two developer-friendly languages used for cross-platform and native application development, are built to enable rapid iteration cycles and team productivity.

To improve delivery outcomes, start with CI

Competition, economic pressure, security threats, ever-increasingly complex applications: today's software delivery organizations face a long list of challenges that demand innovative solutions. A robust CI practice provides the foundation on which you can build a fast, secure, and most importantly, sustainable software delivery process that will keep your organization on target and your customers happy.

Research shows that teams who invest in best-in-class CI tooling meet or exceed industry benchmarks for engineering performance.

Average users of CircleCI rank among the highest-performing teams in the industry and **save up to \$14 million dollars over a three-year period** due to improvements in productivity, efficiency, and code quality.

THE DATA AND RECOMMENDATIONS IN THIS REPORT OFFER A ROADMAP to achieve and even exceed these results at your organization. Along with fostering a collaborative, high-trust culture and putting people and resources in place to optimize developer experience and productivity, you can use these CircleCI tools and features to get the most out of your pipelines:



VISIBILITY

Being able to record and monitor your team's performance across the four key metrics is an essential first step toward improving your delivery process. CircleCI users have access to the Insights dashboard, which measures the four key metrics for all of your workflows and also offers time- and money-saving data on credit spend, resource usage, and test flakiness.



FLEXIBILITY

CircleCI offers the industry's largest selection of execution environments to support your application development, including Docker, Linux, macOS, Windows, GPU, and Arm. You can easily adjust the size of your machines to find the right balance of cost and workflow performance. With the option to manage your pipeline using our official VS Code extension, web UI, or CLI, you can code where you are comfortable and stay informed and in flow. And with support for major code hosting platforms including GitHub, GitLab SaaS and self-managed, and Bitbucket, CircleCI fits into your software delivery pipeline no matter where your code lives.



CONTROL

With support for role-based access controls, config policies, secure environment variables, and OpenID Connect tokens, CircleCI gives you a number of options for restricting who has access to critical project and configuration data, as well as what types of tools and workflows are allowed in your pipelines.



SPEED

CircleCI offers all users access to a fleet of custom-built Docker images. These images have been optimized for CI so that they are more deterministic and faster to load, cutting down on build minutes and saving you from having to build your own images.



STANDARDIZATION

CircleCI orbs are sharable, reusable packages of configuration that you can use to import your favorite tools and components across multiple workflows with just a few lines of configuration. CircleCI users have access to a library of open source orbs created by trusted technology partners and community members, or you can create your own private orbs to share among authorized members of your organization.



PRODUCTIVITY

Intelligent test splitting, matrix jobs, parallelization, and concurrency options significantly speed up test execution. By running multiple jobs simultaneously across separate build nodes, you can increase test coverage and success rates without sacrificing on duration.



EFFICIENCY

With advanced caching options, you can store and reuse data from previous jobs to reduce the duration of your workflows. Docker layer caching, for those building custom Docker images, can allow for an even greater reduction in workflow duration. And with dynamic configuration, you can use jobs and workflows not only to execute work but also to determine the work you want to run for more dynamism within your pipelines.



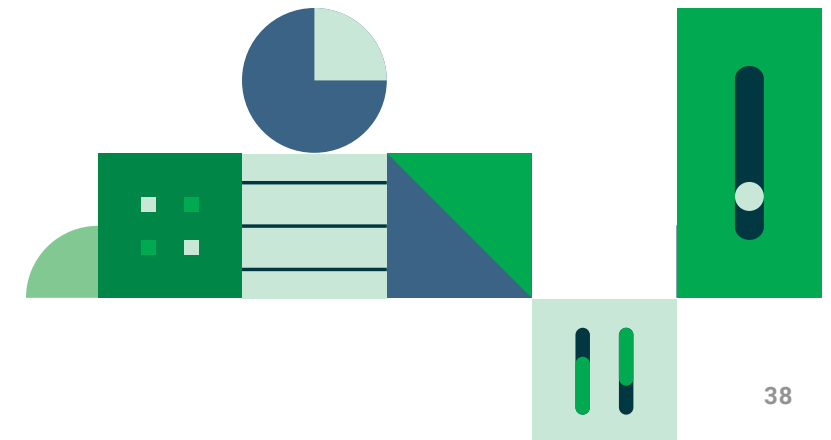
RESILIENCY

CircleCI offers the ability to rerun a failed workflow and to use SSH to access and debug the machine that fails. Troubleshooting in the remote build environment gives you the power to quickly reproduce, diagnose, and remediate issues, saving countless developer hours and shortening your MTTR.



SUPPORT

Premium CircleCI support includes the option for one-on-one training and configuration reviews by DevOps experts at CircleCI. These reviews find optimization opportunities that can greatly reduce workflow duration and other configuration bottlenecks.



Methodology

IN ORDER TO CREATE THIS REPORT, we pulled data from nearly 15 million CircleCI workflows within the first 28 days of September 2022. We also pulled data for days in 2022 with major cultural events for our productivity analysis. We filtered this to only include projects that use GitHub as their VCS. In an attempt to restrict our analysis to real companies and repeatable workflows, we restricted the dataset to projects that have at least 2 contributors and workflows that ran at least 5 times on CircleCI. The number of contributors is also total time on CircleCI, not just the analysis time.

When analysis restricts to the default branch of the project, it is using the current value for the default branch, possibly missing some older data for projects that changed their default branch during the analysis window. Industry data is sourced from Clearbit and is not available for all organizations.

Data details:

- Every day between September 1, 2022 and Sept 28, 2022
- Major cultural event days: February 24, 2022, June 24, 2022, September 8 & 22, 2022, October 20 & 27, 2022, November 8 & 28, 2022
- Only GitHub projects
- Only projects with more than one contributor
- Only workflows that ran at least 5 times
- 14,157,214 workflows

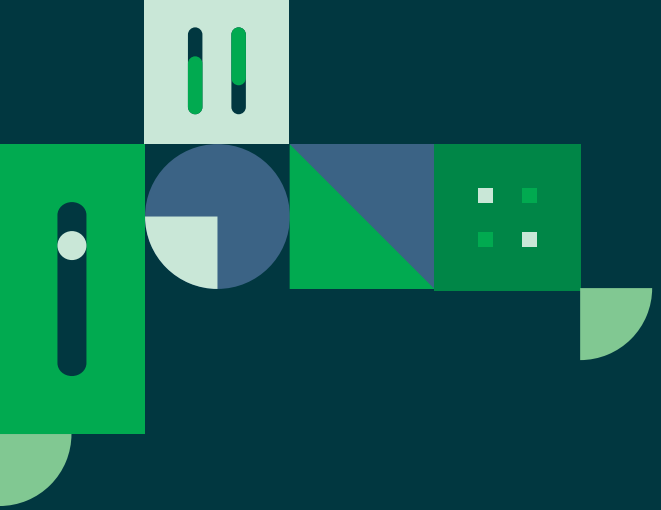


REPORT AUTHORS Ron Powell
Jacob Schmitt

REPORT EDITOR Molly Fosco

REPORT CONTRIBUTORS Joe Duffy, CEO at Pulumi
Tomas Gonzalez Blasini, Partner Solutions Engineer at Snyk
William McMullin, Product Marketing Manager at Datadog

ACKNOWLEDGEMENTS Yasser Nadeem



 circleci

