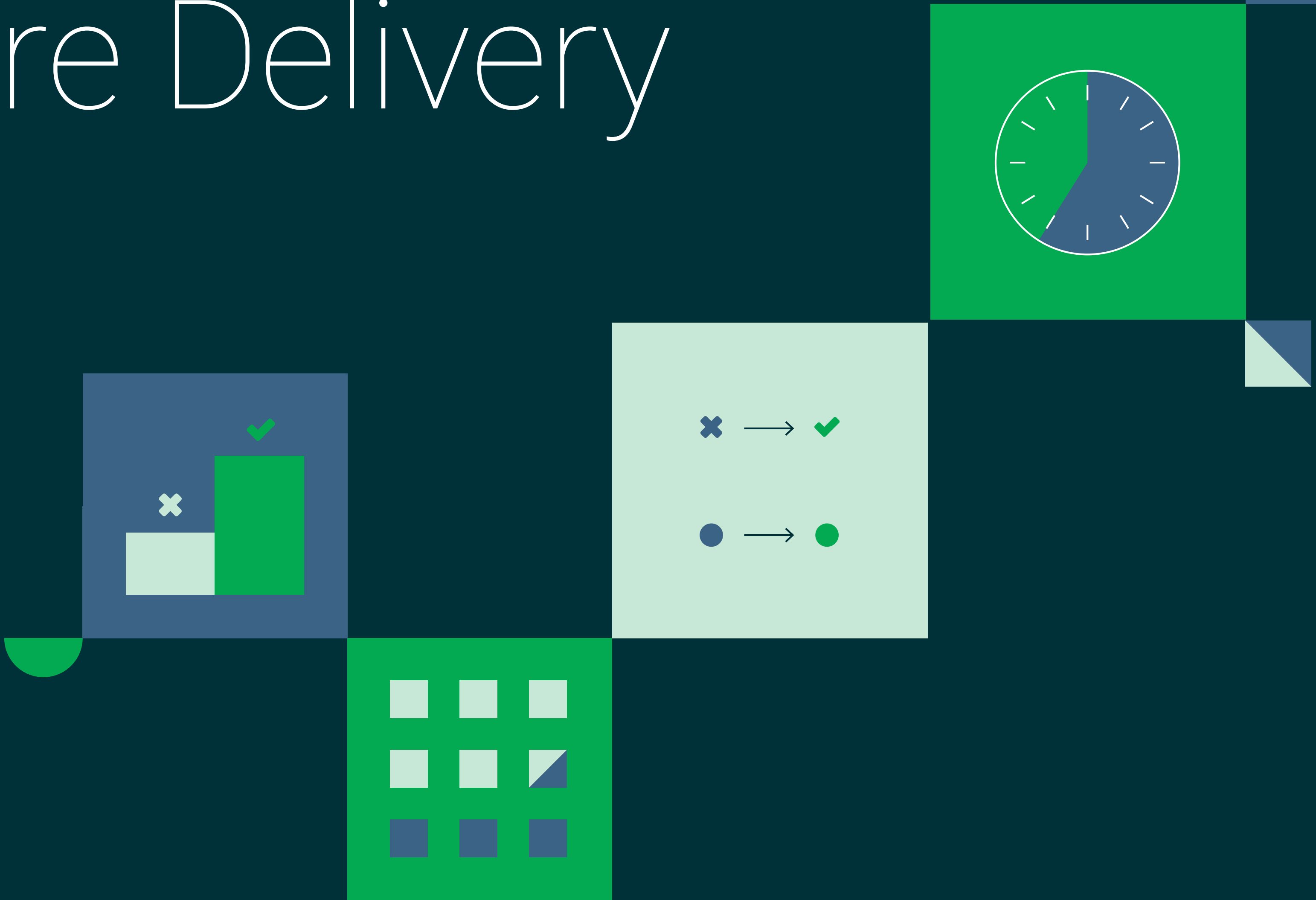
circleci

2022

The 2022 State of Software Delivery

by Ron Powell



Executive Summary

Software delivery has never been more critical to the success of business in every industry. It's also never been more complex. Expectations for delivering high-quality software incredibly fast have skyrocketed. Yesterday's most exceptional product experiences are expected today to be more beautiful, intuitive, powerful, and affordable. But the landscape of tools, platforms, and architectures is constantly evolving. With this rapid pace of change and the growing challenges of complexity, how can engineering teams not only succeed but beat out the competition?

The 2022 State of Software Delivery Report represents the largest collection and evaluation of developer engineering productivity data in the world: over two years of data from over a quarter of a billion workflows, representing almost 50,000 organizations from more than 100 countries, building over a quarter of a million projects on the CircleCI platform. Our research team examines this data each year to gain insight into the DevOps practices used by software teams globally.

The data shows that the most successful engineering teams routinely meet these 4 benchmarks:

- They prioritize being in a state of deploy-readiness, rather than the number of workflows run
- Their workflow Durations are between five to ten minutes on average
- They recover from any failed runs by fixing or reverting in under an hour
- Their Success Rates are above 90% for the default branch of their application

We've been publishing this research for three years, and we're pleased to say that more teams than ever before are hitting these benchmarks. This means that more teams have identified that delivering software successfully at high velocity is a competitive differentiator for their product, and have prioritized the key components of hitting this goal. By hitting these benchmarks, these high achieving teams are getting maximum value from their software delivery pipelines.

To achieve top-performing status costs time and money but it's clear that more organizations are realizing that it's worth the investment. Business leaders that outfit their teams with the most performant and powerful tools allow their software teams to be engines of innovation, unlocking new ways for their entire company to operate more effectively and opportunities to get better products to customers sooner.

Key Findings

Our recommended baseline metrics continue to define industry standards.

Top delivery teams continue to show key similarities on our platform. Every organization is different and many have business-specific reasons for choosing certain benchmarks, but the teams hitting our recommended baseline metrics are among the highest-performing each year.

Delivering software successfully with high engineering velocity remains the number one goal for teams on the CircleCI platform. Those that meet these baseline benchmarks get incredible value from their software delivery pipelines, and more teams are meeting them and entering the elite tier.

As your software product matures, your pipelines will run for longer Durations on average.

Teams on CircleCI are running more advanced pipelines than ever before: we're seeing workflows that include sophisticated test suites and we're seeing growth in the use of deployment tools and technology year-over-year. This indicates that our users are becoming experienced enough at Continuous Integration to implement Continuous Delivery and Deployment scenarios as their business demands, signaling that the DevOps industry is growing and maturing overall.

The goal is not DevOps maturity alone, it's also product maturity. Achieving high performance and product maturity are only possible with code that is well-tested in the cloud, and that means your pipelines will take longer to complete. There's an inflection point to consider where additional Duration starts to impede progress. As your Duration increases, so does your Mean Time to Recovery (MTTR).

Small teams can compete with the enterprise by prioritizing test-driven development

Typically, early stage organizations with small engineering teams do not have the testing in place to properly mitigate failure quickly. They try to solve failures by having their engineers join forces to resolve problems together. However, due to their size, they are not able to consistently commit many resources to recovery. This is particularly challenging during the end-of-year holiday season when demand for software products and services is often at its peak, at the same time that staff want to be home with their families. Major market swings happen throughout the year with similar impacts to your organization: a high demand for your products and services at a time when staff need time off.

In our 2020 report, we saw that teams were able to maintain high velocity and average recovery times throughout the onset and then the new reality of a global pandemic. The teams most adept at CI, who could innovate with confidence knowing that rich output would accompany any error, were able to continue to innovate throughout, despite being faced with the same difficulties as all developer teams around the world. Similar impacts to our teams happen every year around the holidays and again, teams most adept at CI are not slowed down by those challenges.

When engineering teams at early-stage companies achieve maximum benefit from their CI/CD pipelines, they can still gain a competitive advantage, whether it be during the seasonality of the business cycles or during large-scale impacts to the market as a whole.

Practices like test-driven development (TDD), which includes extensive testing, quality checks, and systems that prevent bad code from being put into production, act as a fail-safe when headcount is low. Small teams can compete with larger ones if they prioritize TDD because they can confidently rely on their tooling during seasonal fluctuations and times of uncertainty.

To confidently rely on your tooling, it is essential to create an optimal team structure and culture.

Our four recommended baseline metrics individually measure important aspects of software delivery efficiency but collectively, they inform team dynamics. For example, Throughput isn't the measure of a single developer's productivity; it's all the changes to the product made by the whole team. Similarly, Success Rate is not measured in terms of one developer, but the rate at which successful changes are made by the entire team.

To ensure team success, and therefore software delivery success, we need to help our developers to stay in flow. That means aiming for short workflow Durations so they provide signals about the most recent changes as quickly as possible. It also means scheduling meetings at times that don't conflict with peak productivity hours.

While it may seem convenient to have a meeting when most of your team is online, scheduling meetings during this hour may not be the best use of your developer's time. Asynchronous communication is key to keeping global teams aligned on project progress.

Keeping developers in flow is a priority for the highest-performing teams. It also means having the right team size to achieve your objectives while avoiding burnout, and then, building extensive testing into your DevOps practice so you can rely on your tools with confidence.

Introduction

What do high-performing engineering teams look like?

In 2021, the world continued to face daunting uncertainties. But as in 2020, the DevOps and CI/CD industries continued to experience hyper-growth. Why?

Microservices and dependencies have exploded in recent years making software development infinitely more complex. Today, very few applications are built by in-house engineers writing custom code. Instead, apps are stitched together by combining pre-existing libraries collected from across the internet.

20 years ago, if you ran into an issue with your software, your team could probably fix the problem because the team controlled the entire codebase. In 2022, even the most gifted team of engineers likely cannot fix many of the issues that arise because of the magnitude of components and moving parts that go into the products they build. Managing this complexity not only requires staying on top of all of your dependencies but your security and stability as well.

The most effective way to do this is by practicing CI/CD. Software teams that are among the highest performing are the ones that intentionally work to get the most value out of their CI/CD pipeline. As a result, they experience fewer bugs, defects, and setbacks, and they're able to get products into the hands of customers sooner.

But how do you get there in practice?

In our 2020 report, we showed you the baseline metrics your team needs to achieve to be among the top-performing software delivery teams in the world. We also highlighted that high-performing teams must be resilient, which means having the right size engineering team for your goals, supporting one another, and being flexible.

This year, we'll cover the basic metrics again but we'll also expand on them by providing insight into *how* your team can achieve them. Using our analysis of the data of tens of thousands of teams on CircleCI, we'll detail some of the systems, tools, team processes, structures, and everyday practices that empower engineering teams to go from good to great.

History of Continuous Integration

As software development became more complex over time, it became advantageous to integrate code early and often to ensure that all components of an application work together successfully.

Continuous Integration was born out of the Agile method of developing software, which prioritizes being responsive to change, optimizing for shorter delivery cycles, and minimizing risk by breaking work into smaller chunks. Agile processes helped teams become more comfortable with failure because they were testing and delivering software in smaller bits more frequently, enabling them to catch more bugs at earlier stages and prevent catastrophes.

Adding integration into the build pipeline meant that development and operations teams could create testing suites that run on every change to the codebase. Extensive automated testing allowed teams to innovate quickly 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.

How do you give yourself an advantage?

CircleCI is in a unique position as the world's largest networked CI/CD provider to access real data around how software is developed. We looked at over a quarter of a billion workflows, commit by commit, to see how teams were building and deploying software in practice.

Today, almost every company creates and deploys a software product. To get ahead of your competition, quality delivery, with speed and at scale, is the key to creating a competitive advantage. CI/CD pipelines are table-stakes for organizations that want to create differentiation from their competitors and deliver digital products as fast as the market demands.

Our goal is to answer these questions: What does a high-performing team really look like? What defines their success?

To avoid confusion, let's cover a few key terms that will be important as we define the baseline metrics for engineering teams to target 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
is the average time between a 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 are not as far along their path to DevOps maturity and not yet monitoring these key metrics.

	2022 MEDIAN VALUES	BENCHMARK
DURATION	3.7 minutes	10 minutes
TTR	73.6 minutes	< 60 minutes
SUCCESS RATE	Average was 77% on default	Average should be +90% on default
THROUGHPUT	1.43 times per day	As often as your business requires—not a function of your tooling

What does each metric reveal about how teams are improving their software delivery?

Duration

Duration is the length of time it takes for a workflow to run. What is the appropriate length of time for a fast feedback cycle?

Nothing is more important in CI than fast feedback but rapid awareness of a workflow's failure or success is not the only consideration. Developers also need specific information from their failed builds. You cannot push a fix without first identifying a solution from the information provided by the failure, making a change, and then running a complete workflow to success. Writing rigorous tests for your software will ensure that you'll get exactly the information you need at the crucial moment.

So, what length of Duration should your team be targeting? Our industry-leading benchmark for Duration is 10 minutes because it's essential to maximize the amount of information you can get from your pipeline while still moving as quickly as possible. 10 minutes is where we feel developers can move fast without losing focus and will benefit from the volume of information generated through their CI pipelines — it's the optimal time for fast feedback, robust data, and speed.

In fact, a Duration benchmark of 10 minutes is a well-accepted industry standard that hasn't changed in 15 years. In his 2007 book, "Continuous Integration: Improving Software Quality and Reducing Risk," Paul M. Duvall and his co-authors recommend 10 minutes as the ideal pipeline Duration because the optimal volume of information you could generate for your pipeline took 10 minutes to complete on average. This is still true today but the volume of information that can be obtained during today's 10-minute pipeline far exceeds what could've been imagined by Duvall and his co-authors in 2007.

From December 2019, the earliest time observed for this year's report, through September 2021, the last month observed, Durations increased on average for all teams on CircleCI. This makes sense. As more teams mature their applications and increase their DevOps maturity on CircleCI, they include more-rigorous testing and increased use of third-party tools. This results in workflows taking longer to complete.

The average Duration for September 2021, the last month observed, came in between 12-13 minutes, which is just above our Duration benchmark of 10 minutes. The median was only four minutes, which is quite short, but not necessarily an advantage. If your pipelines are averaging 4 minutes, this likely means there's ample room to include more information-generating processes such as adding tests to your test suites, adding third-party tools for security or compliance scans, or measuring code coverage.

What can your team do to achieve an average Duration time of 10 minutes? Remember, an ideal Duration is one that allows you to gain maximum information in minimum time. The lists below provide tips to gain both efficiency and richness of data in your pipeline runs.

TO INCREASE THE EFFICIENCY OF YOUR RUNS AND REDUCE YOUR DURATION:

- Use test splitting to split tests and take advantage of parallelism. Splitting tests by timing data is particularly efficient.
- Use Docker images made specifically for CI. Fast spin-up of lean, deterministic images for your testing environment saves you time.
- Use caching strategies that allow you to reuse existing data from previous builds and workflows.
- Use the optimal size machine to run your workflow. Larger jobs benefit from more compute and run faster on larger instances.

TO INCREASE THE RICHNESS AND VALUE OF YOUR TEST DATA:

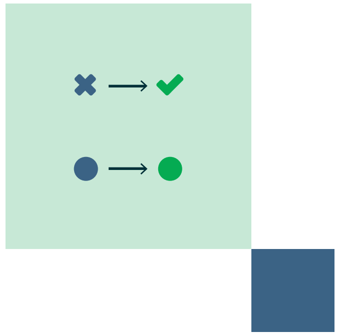
- Adding tests to your unit testing, integration testing, UI testing, and end-to-end testing layers can all increase test coverage and test comprehensiveness across your applications.
- Testing across the layers of your application is still not robust enough to get your service into production without error. To build, test, and deploy your product, you must also test how your service interacts with services outside of your organization. An additional layer of testing in a complete DevOps model involves testing the responsiveness and availability of these services too.
- Use security scanning tools to perform application security tests to find vulnerabilities. Before you ship to production, these scans decrease the chance of releasing vulnerable software to your users.

Every engineering team should aim to get the maximum benefit from their CI/CD pipelines, which means moving at high velocity *along with* the ability to recover quickly.

With more code continually deployed and pipelines that are automated all the way through to production, it is absolutely essential to be able to mitigate poor outcomes. Again, achieving DevOps maturity and product maturity means your pipelines will take longer than the bare minimum to complete, which is a good sign that your team's practices are maturing.

"Software quality is one of the most difficult elements to quantify in managing engineering teams. At StackHawk, we've started tracking Change Failure Rate, which is the percentage of changes we deploy to production that contains a customer-impacting bug or issue, or which fail any production verification tests. This metric has helped us have visibility into our software quality while continuing to embrace a continuous delivery model and move toward smaller, more frequent deploys. Combined with a robust set of automated tests, this approach gives us confidence that we can deliver features quickly to our customers, while still holding ourselves accountable for delivering high-quality, reliable software."

Jeremy Goldsmith
Head of Engineering, Stackhawk



Mean Time to Recovery

Mean Time to Recovery is the average time between a pipeline's failure and its next success. It is the most important metric on the list. The ability of your team to recover as quickly as possible when an update fails, time and time again, is the ultimate goal of Agile development teams.

The more robust testing and corresponding output you have, the faster it is to resolve issues and fix errors introduced to your codebase. Because Mean Time to Recovery (MTTR) improves with more comprehensive test coverage, this metric can be used as a proxy for how well-tested your application is — although it's important to note that error messaging plays a vital role and should be called out specifically.

The shortest that your MTTR can be is equal to your Duration. For example, if a workflow run resulted in a failed state and a developer immediately knew the solution, immediately made a change, and immediately pushed it to their shared repository, the full Duration of a successful workflow run would have to complete before the project can return to a green state.

In two years of data, we've observed that Mean Time to Recovery (MTTR) is impacted most by the end of year holidays. This indicates that annual cycles in the market and the predictable rhythm of the seasons create more powerful impacts on engineering productivity than global phenomena such as a pandemic.

"I'm happy to see CircleCI's emphasis on TTR, as Thoughtworks has long preached that it's impossible to ensure a steady delivery cadence if the team isn't paying attention to the build health and that a red build should be like pulling the proverbial Andon cord. We often find that the most robust — and certainly the fastest — solution to a broken build is to simply revert the offending commit, allowing troubleshooting to happen in a way that doesn't interfere with the rest of the team. You can't know whether a new build works or not unless you're starting from a known good position, which means you should never allow a new build to start on a red build unless it's explicitly designed to fix it, and it's hard to imagine a commit more likely to fix a broken build than simply reverting the one that broke it to begin with. Often the biggest impediment to rollbacks is team culture: it's critical to create psychological safety so team members feel comfortable reverting someone else's commit to restore the health of the build and unblock the rest of the team."

Brandon Byars
Head of Technology for North America, Thoughtworks

MTTR increases across the board for the end-of-year holidays, putting small developer teams at a particular disadvantage. This disadvantage is due in large part to reductions in staff while organizations accommodate a healthy work-life balance and developers take time off. Fewer team members available to debug and fix problems result in longer recovery times over the holidays. It's no wonder that many teams decide to code freeze during this time.

What would it take to give your team time off to relax with their loved ones over the holidays, while still maintaining productivity and avoiding long recovery times? Tests! Investing in your product by building out your test suite, increasing your test coverage, and drafting detailed error responses involves some upfront investment and will lead to longer workflow Durations, but it will empower a smaller team of developers to continue work while their colleagues are offline, effectively ending code freeze, and will reduce the amount of downtime when errors do occur.

This end-of-year period is also when hackers are most likely to take advantage of perceived weaknesses and attempt entry into your systems. Security automation safeguards against this because it means that no single developer is responsible for protecting the entire system — it helps teams avoid hero culture. After all, hero culture is not so useful when the hero is on vacation.

Interestingly, the tools and practices that the most productive engineering teams have in place to buffer market fluctuations in real-time are the same ones that can enable small organizations and small teams to stay competitive and productive during annual downtime cycles. Again, it is all about the tests.

Your team's ability to resolve issues in a timely manner, a regular part of software development, should never be a deciding factor in whether or not to deliver additional value to your customers. Your software team should be able to respond easily to any changes to your business or in the market by releasing new features, updating old ones, or moving into new audiences no matter what time of year it is or the state of the global market.

Automation itself is the first step to improving MTTR. Automation provides you with fast pipelines. But the ability to deliver bad code faster is not a benefit. Teams can only realize the benefits of automation with a robust feedback loop for mitigating error. Without the valuable signal from robust testing created by successful and failing runs of your workflow, you lack the ability to resolve problems when they inevitably occur.

If teams were to employ better, automated testing practices, they could be even better prepared, not only to handle big ripples in the market like covid but also prevent this dramatic holiday slump each year. It is possible for small companies to compete with larger ones if they first prioritize TDD so they can confidently rely on their tooling during seasonal fluctuations and times of change and uncertainty.

WHAT ARE SOME THINGS YOU CAN DO TO LOWER YOUR MTTR?

- Duration is the most important factor to consider when it comes to MTTR so to optimize TTR, you must first optimize Duration.
- Using tooling that supports the rapid identification of failure information through the UI and through messaging like Twilio, Slack, and Pagerduty allows you to be notified as soon as possible when a failure occurs.
- Writing tests that include expert error reporting will help you quickly identify what the problem is when you go to fix it.
- Debug on the remote machine that fails. The ability to SSH (Secure Shell Protocol) onto the failed machine of a workflow is massively helpful for an engineer who is still looking for clues as to why an error occurred. Rich, robust, and verbose log output is useful without access to the remote machine.

Remember, one of the most important parts of software delivery is shortening the time between when a defect is introduced, discovered, and then fixed. There is no such thing as error-free software development. We must assume that failure will happen and be prepared to address it as quickly as possible.

Throughput

Throughput is the average number of workflow runs per day. A workflow is triggered when a developer makes an update to the codebase in a shared repository. A push to your version control system (VCS) triggers a CI pipeline that runs your workflow.

Large commits are pushed less often, so a lower Throughput may indicate that your commits are quite large. When it comes to the size of your commits, you want changes to your codebase to be small enough to debug quickly and easily, but large enough to reflect a meaningful update to your project. If your goal is to increase your Throughput, consider making smaller commits more often.

Throughput varies greatly depending on the needs of your business. Throughput is also dependent on Duration and MTTR. For example, if my pipeline takes eight hours to run (Duration), I can only complete three workflows per 24-hour period (Throughput).

On CircleCI, 50% of workflows in our data set were run less than once per day (0.7 times per day on average). However, the 95th percentile includes workflows that were run over 35 times per day or about once every 45 minutes throughout a 24-hour period.

Your software delivery cadence should be dependent on the software your team is building and on your business needs. Making changes to your project with an automated testing workflow means your organization is no longer blocked by waiting for manual quality assurance checks. With a fully automated software delivery pipeline, it is up to you how frequently (and when) updates are delivered to your end-users.

Measuring your baseline Throughput and then monitoring for fluctuations will tell you more about the health of your development pipeline than aiming for an arbitrary Throughput number or comparing your stat to others. A particular number of deploys per day is not the goal but continuous validation of your codebase via your pipeline is.

HOW CAN YOUR TEAM ACHIEVE OPTIMAL THROUGHPUT?

- Again, when it comes to Throughput, it is more valuable for organizations to see their own changes and progress week-over-week than it is to compare to industry standards. Once your development patterns have been decided, your Throughput baseline can be measured and then observed for health and efficiency.
- If increasing Throughput is a desire for your team, you should prioritize lean, Agile software development patterns that involve small, incremental changes to projects with a full suite of automated testing that runs on every commit. Testing removes the threat of changes introducing errors. By testing well and pushing small changes often, teams can innovate on their project knowing that any undesirable impact from a change can be investigated and mitigated quickly and with ease.

Success Rate

Success Rate is the number of passing runs divided by the total number of runs over a period of time. The Success Rate for any given project is not particularly interesting without knowing the development patterns that define your team's use of a version control system (VCS). These patterns, or Git-flow models, organize the work that teams perform on a project onto branches off of the main branch where the production code is hosted. This means that a topic or feature branch of a project will likely have a lower Success Rate that reflects increased innovation and experimentation, while the main branch that holds the production code will likely have a very high Success Rate that reflects stability.

On a feature development branch, failed builds are desirable. That means that information is being generated by the workflow and the feedback loop is working. The more information that is generated from a run, the easier it is to debug failures keeping the Mean Time to Recovery low.

Failed builds on the main branch are not completely undesirable. It is inevitable that there will be failed builds on the main branch. You need to know whether the information that you generate from a failed build on the main branch is sufficient to keep your time to recovery on this branch low. The Success Rate of the main branch needs to be actively monitored, and if it is not high enough (we suggest 90% or above), then it should be addressed through increased testing.

"Jez Humble recently tweeted that CI/CD is about more than 'taking whatever [stuff] you have in version control and shipping it into prod as fast as possible so you can test in prod.' At Katalon, we believe that TestOps is a critical element of effective software delivery. Increasing Throughput only matters if your software remains deploy-ready. Test automation can help you ensure that is the case. However, you can only keep your workflow Duration down if your tests are run efficiently — taking advantage of intelligent test selection and parallel test execution. Mean Time to Recovery can be reduced when you are able to identify and diagnose test failures quickly. Ultimately your success rate will depend on being able to maintain effective test suites as your software grows and changes. CircleCI makes integrating testing into the CI workflow easy and enables organizations to ensure they are doing more than shipping questionable quality quickly."

Coty Rosenblath
CTO, Katalon

The significance of Success Rate on topic or feature branches can be variable, depending on the structure and size of your team, and the projects they're working on. The larger the team working on the same topic branches, the more important it is to keep them green so that a red state is not blocking others from development. This is accomplished through robust testing.

The ability to measure the Success Rate of your current workflows will be essential in establishing targets for your team. Remember, failed builds are not a bad thing, especially if you are getting a fast, valuable feedback signal, and your team can resolve issues quickly

The goal isn't to make updates to your application; the goal is to constantly innovate on your software while preventing the introduction of faulty changes.

HOW CAN YOU ACHIEVE AN OPTIMAL SUCCESS RATE?

- Success Rate should always be high on the primary branch of your service. Choosing a Git-flow model such as short-lived feature branch development or long-lived development branches that allow your team to innovate without polluting the primary branch will keep your product stable and deployable. Feature branches do not need to have as high of a Success Rate as the primary branch, as it is expected that the changes involved in creating new features will result in failure.
- Feature branches can have lower Success Rates without negatively affecting the product or teams working on other branches. It's important to monitor Success Rate on these branches along with MTTR. Low success accompanied by long MTTR is a sign that your testing output is not sufficient for debugging and resolving issues quickly.

The bottom line is that the four metrics together provide a constant feedback loop to give you better visibility into your software development pipeline. Remember, the goal isn't to make updates to your application; the goal is to constantly innovate on your software while preventing the introduction of faulty changes.

What does an optimal team structure and culture look like?

To improve your software delivery metrics, it is essential to prioritize team structure and culture. While the ideal team structure and culture will vary depending on your goals as an organization, making sure your developers stay in flow is key to ensuring they stay as productive as possible. That means scheduling meetings at times that don't conflict with peak productivity hours. The peak time of work on the CircleCI platform is done between 6 a.m. and 7 a.m. PT (9 a.m. and 10 a.m. ET) on Wednesdays, meaning that's when most developers are online building software.

Identifying the right number of people for your team is also essential. Three out of four of our key metrics show a correlation between larger team size and better engineering performance. While the ideal team size differs based on your objectives, the scope of your responsibilities, as well as other variables, our data shows that the best place to aim is between 5 and 20 code contributors. A larger team is also the best way to avoid burnout.

Compared to last year’s data, there has not been a large increase in evening or weekend workflows, which could indicate that teams are prioritizing a culture that does not overwork their engineers. Burnout can be one of the biggest productivity killers, which translates to a loss of revenue.

Building a team for optimal flow gives you the confidence to rely on your tooling and improve your overall software delivery metrics. Once your team has built extensive tests into your CI/CD practice, the signals you get are more meaningful. A failed signal is not as useful without extensive testing in place.

Using CI correctly means solving the problem of putting bad code into production. As long as quality software relies on humans to write it, we need to develop practices that support those team members to do their best work and support their own wellbeing. We need incredibly detailed structures and built-out practices so developers can be successful without burning out along the way.

In this year’s analysis, we looked at the time of day that contains the largest volume of work performed on our platform.

There is no one time zone that includes enough developers to produce a single peak of productivity. This speaks to the global footprint of modern application development teams. The time when there are more developers online making changes to their software than any other time during the day is 15:00 UTC or 7:00 a.m. PT. This time is late in the day for Europe, the middle of the workday for the U.S. East Coast, and early morning on the U.S. West Coast.

What does an elite team look like?

Year after year, we've provided our analysis of the four metrics, as measured on our platform through millions of pipelines run. It makes sense to think of optimizing strategies for them individually because different processes may be needed to improve each one. But what does an optimized combination of these metrics look like and which might be the best ones to choose for your team?

At the heart of our quest for software development velocity is both speed and quality. Quality means that we are not pushing updates to our products that result in undesired behavior for our users. Speed means that we can do this not just quickly, but consistently.

What do speed and quality look like when it comes to these metrics? The most essential thing to aim for is an MTTR that is as close to your average Duration as possible.

Robust test coverage and helpful, verbose error reporting take time, but those who put the time in upfront reap the benefit of innovation with confidence. Prioritizing these practices allow teams to be creative, challenge themselves in an environment that allows them to fail, and respond to failure with confidence.

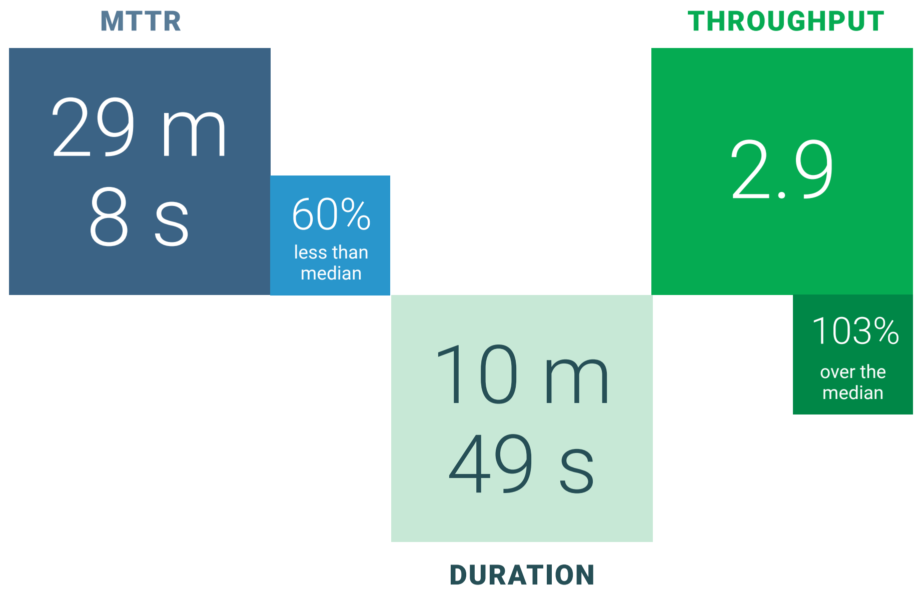
As open source projects have CI configuration files that are viewable to the public, we want to share some examples of teams that exemplify optimized performance metrics. These three OSS projects on CircleCI are meeting the ideal combination metrics of elite teams. They have Durations between 10 and 11 minutes, MTTRs well under an hour, and Throughputs from 2.4 times per day to almost 40.

Take a look at their CircleCI configuration files (the config.yml file in the .circleci folder), and note what they're doing.

	2022 MEDIAN VALUES	BENCHMARK
DURATION	3.7 minutes	10 minutes
MTTR	73.6 minutes	< 60 minutes
SUCCESS RATE	Average was 77% on default	Average should be +90% on default
THROUGHPUT	1.43 times per day	As often as your business requires—not a function of your tooling

Armada

Armada is using one of CircleCI's in-house-developed Docker images for Go. CircleCI builds these images to be the fastest and most deterministic images available. They are designed specifically for CI.



The Big Give Donate Frontend

The Big Give takes advantage of two different notification services in their workflow. The Slack orb and the Jira orb are used to keep their teams continuously updated on the status of projects. Early alert is key to keeping MTTR as low as possible.

MTTR

28 m 31 s

61% less than median

THROUGHPUT

2.4

65% over the median

DURATION

10 m 44 s

"The Big Give builds and maintains several web apps and microservices to support match-funded charity campaigns at scale. As a tiny technical team, automation is the only way we could achieve this while maintaining high quality, and Continuous Integration and Delivery plays a crucial role. Motivations for the updated web platform we launched in 2019 include prioritizing security and stability and reducing avoidable custom code. But with Continuous Integration, we've also increasingly been able to focus on getting improved experiences to our charities, donors, and champion funders as soon as possible.

Our platform is now flexible enough to respond to new funding needs without code changes. Keeping our CI pipelines in a good place also allows us to be pragmatic and to quickly and safely adapt our apps if needed, even when things change mid-sprint. This has helped us to accommodate new requirements in response to the unpredictable times facing our charities and partners in recent months."

— Noel Light-Hilary, Tech Lead at The Big Give

Bolt Checkout Plugin for Magento 2

Bolt also uses CircleCI-developed orbs to alert developers of the build status of the projects that they follow. In addition, Bolt is using a community-developed orb that includes a number of helpful commands. Using these commands, their team is able to create custom workflows that allow for running specific tests depending on which part of the project changed. This keeps the average Duration of a workflow for this project low, while still delivering the relevant testing output.

MTTR

47 m 20 s

36% less than median

THROUGHPUT

38.9

2618% over the median

DURATION

10 m 23 s

"With custom compute, my team builds, tests, and deploys applications that have unique system requirements without the need to also manage hardware. Scaling concurrency has also been huge for us because it makes our workflows insanely fast. The net effect of our optimized pipelines is not only fewer master breaks and triple the tests, but much faster builds. What used to take as long as two hours now takes about 30 minutes without increasing resources. So our costs are still the same but the builds take a third of the time and we run more tests. We're able to ship more reliable code without the overhead."

— Roopak Venkatakrishnan, Engineering Manager at Bolt

The move to more frequent deploys

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. It's used for resource allocation and traffic management for cloud applications and microservices.

For Kubernetes (K8s) to be useful to your team's DevOps practice, there are a few requirements:

- You're willing to operate more than one virtual machine
- You're able to assign people to do configuration and maintenance of Kubernetes
- You're managing more than a single service
- You need to automate (as much as possible) a mostly homogenous service deployment
- You need to be cloud (or hosting) provider-agnostic

The list of requirements needed to successfully deploy K8s is not trivial. Therefore, it's not surprising that the majority of organizations using K8s in our data are large, enterprise-sized organizations. The smaller orgs that find themselves in a position where K8s make sense in their stack do not sway the data. These orgs exhibit the same behavior: longer Duration, shorter MTTR, and higher Throughput.

When we looked at projects that were using Kubernetes, we saw that on average, those projects had longer Duration (46% longer at the 50th percentile), lower MTTR (46% lower), higher Throughput (57% higher), and similar Success Rate (about 77% success on the default branch) as those without K8s. We also see an increase in projects using K8s, year-over-year. What would explain these signals?

More companies adopting Kubernetes aligns with several industry trends. First, many organizations are migrating their monolithic applications to microservices. Second, organizations are automating their software pipelines beyond building, testing, and leaving artifacts in a deployable state — a situation that would have been considered advanced in the past. Finally, more and more teams are using Continuous Integration to deliver updated products all the way to their consumer's hands at breakneck speeds.

To get your updates into the hands of your users as fast as possible, all of the quality checks that were once manual, tedious processes need to be automated. This scenario puts a tremendous amount of pressure on the testing suite created for the application. Robust testing takes longer to do, therefore, pipelines using K8s take longer on average, but the longer Duration is made up for by a shorter MTTR, making the investment worth it.

The use of K8s also involves cloud service providers. It takes time for these workflows to connect to and communicate with these services, spin up new products in the cloud, and move artifacts from one place to another. Connecting to these services increases Duration, but that increase does not aid engineers in identifying errors and pushing fixes. Interacting with these services introduces a host of new testable scenarios. The output of these additional tests will be critical in determining a fast resolution to a failed build signal.

Organizations making the investment into automated testing, automated infrastructure deployment, and wholesale adoption of the DevOps paradigm from commit to deploy, are reaping exceptional benefits. Increased Throughput is made possible by maintaining low MTTR. It means keeping the developers on these projects building new things instead of chasing down bugs without clear signals as to what went wrong, where, and how to find a solution — all things that come with robust testing.

To improve your delivery outcomes, start with CI.

Adopting Continuous Integration's thorough, automated testing practices alongside Agile product development is key in differentiating your organization from competitors. Automation has crept into almost every development team to some degree and many teams are beginning to excel at CI. If your team is not a leader in developer velocity, then they are, at best, only on par with their competitors. Losing an edge in a competitive marketplace is a huge disadvantage.

Our data shows that many, many teams are able to meet or exceed our industry-best benchmarks for engineering velocity. Average users of CircleCI rank among some of the highest-performing teams in the industry.

But what can you do to beat the average? How do you maintain your advantage when your competitors have access to the same tools that you do?

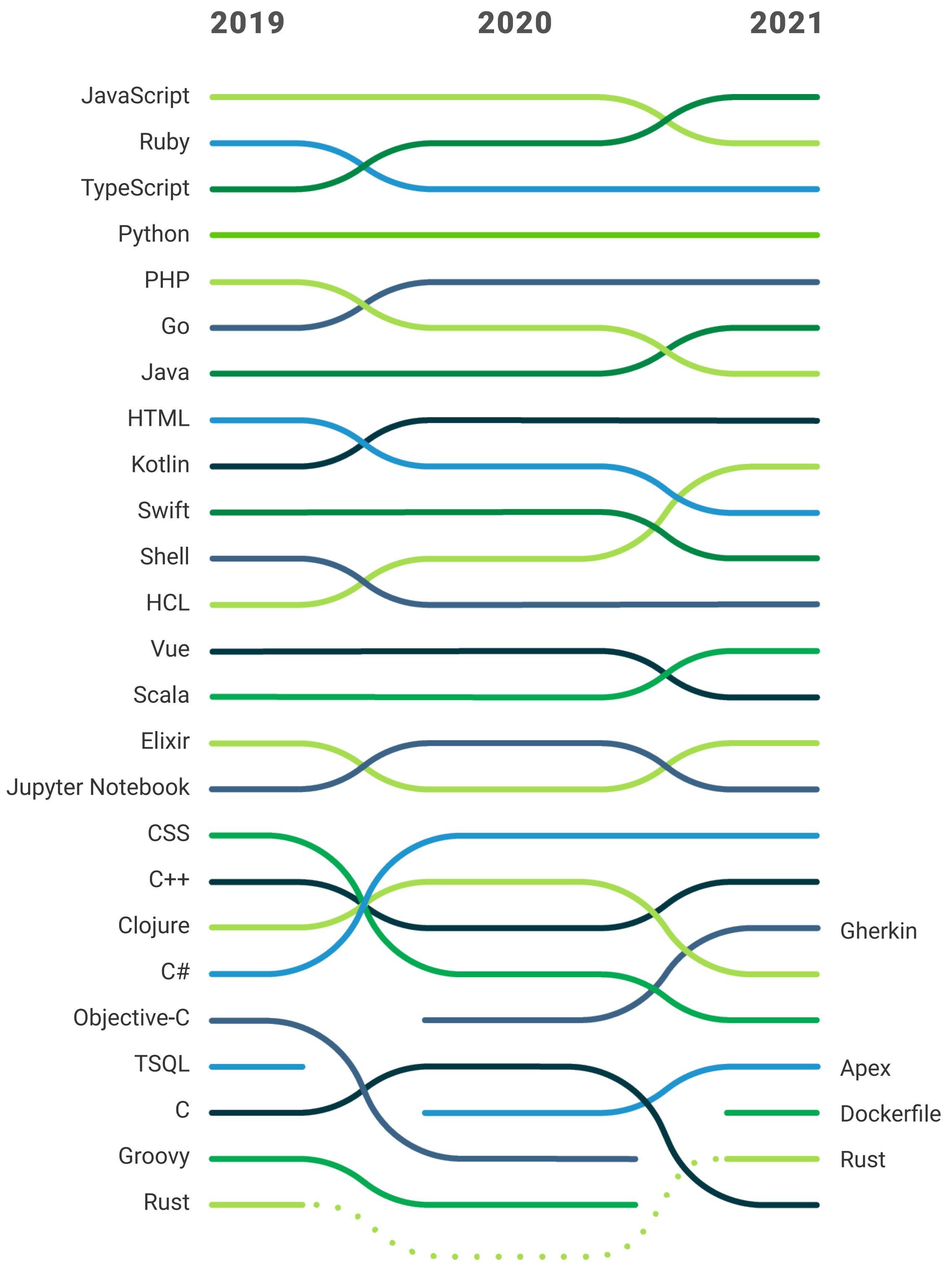
Based on our analysis, we know that average use of CircleCI enables elite stats for performance. We also measured workflows that far exceeded these median values and we have some ideas about how they get there.

- The first step to improving your team's metrics is to record where they fall today. You have to know how well your team is doing to set realistic targets. Making incremental progress across the metrics will have a measurable impact, but only if you can establish your team's baseline first. If you're a current CircleCI user, you have access to the Insights dashboard, and measurements of each of the metrics in this report, for all of your workflows.
- CircleCI offers a wide range of execution environments. The most commonly used are Medium Docker and Medium Linux. Selecting even larger machines to run your workflow can reduce the time it takes for that workflow to run.
- Speed: CircleCI offers a fleet of Docker images for our users. These images have been optimized for CI for your convenience so that they are more deterministic and faster to load.
- Intelligent test-splitting, matrix jobs, parallelization, and concurrency options allow for robust test suites to run in significantly shorter times. Due to this, increasing your testing does not cause a proportional increase in workflow Duration.
- Dynamic configuration allows for users to use jobs and workflows, not only to execute work but to determine the work they want to run for more dynamism within their pipelines.
- Scheduled pipelines let users run pipelines at regular intervals — hourly, daily, or weekly for enhanced pipeline control and efficiency.
- Advanced caching options available on the platform significantly reduce the Duration of a workflow when optimized. Many packages used to build your application can be cached and reused, saving you the time involved with downloading these packages on every run. Docker layer caching, for those building custom Docker images, can allow for an even greater reduction in workflow Duration.
- The fastest way to debug a failed build is to gain access to the machine where the workflow failed. CircleCI offers the ability to rerun a failed workflow and to use SSH to gain access to the machine that fails. Getting a signal fast is only one side of the CI feedback loop. The other side is the ability to quickly recover. Nothing lets your developers get closer to the problem than access to the machine that failed.
- CircleCI offers orbs, which are reusable packages of configuration. Abstracting layers of code from CI configuration files into open-source, community-created, and validated components, or private components for use exclusively within your organization, allows you to add or replace services without the risk of failure. Using well-tested configuration components reduces the sources of error, and using orbs to integrate testing suites into your CI pipeline means getting more information from your runs. We observed that Throughput for workflows increases with orb usage (from 0 to 1 orb and from 1 orb to many).
- Premium support includes the option for configuration review by DevOps experts at CircleCI. These reviews find optimization opportunities that can greatly reduce workflow Duration and other configuration bottlenecks.

A fun aside on language

Few teams will look at the popularity of a new language and decide to rebuild their entire codebase, but it is always interesting to observe the trends. These trends highlight changes in the broader industry for development teams at the leading edge of application development. After all, entirely new services will have to be built in the future and the languages popular for building today's services may not be the most ideal for building the services of the future.

Here are the most common languages used on our platform in 2019, 2020, and 2021:



Duration

1. Batchfile
2. SaltStack
3. Makefile
4. Smarty
5. Jsonnet
6. Shell
7. Mustache
8. HCL
9. FreeMarker
10. Dockerfile
11. PLSQL
12. Jinja
13. Elm
14. Lua
15. Liquid
16. VCL
17. Open Policy Agent
18. Groovy
19. Go
20. Starlark
21. API Blueprint
22. Roff
23. HTML
24. R
25. Python

The interesting thing here is not the fact that languages with few build steps are finishing first — many shell scripts run and exit without robust testing. What is interesting is seeing Go show up favorably among mostly scripting languages.

MTTR

1. Gherkin
2. HCL
3. JavaScript
4. Go
5. Clojure
6. C#
7. Vue
8. TypeScript
9. Ruby
10. Python
11. PHP
12. Perl
13. Shell
14. Kotlin
15. Elixir
16. HTML
17. Scala
18. Jupyter Notebook
19. Java
20. Swift
21. Apex
22. CSS
23. C++
24. Rust
25. C

Go, also with a favorable showing in Duration, appears near the top in MTTR, too.

Success Rate

1. Dockerfile
2. Vue
3. Shell
4. Go
5. SCSS
6. HTML
7. TypeScript
8. PHP
9. Python
10. C#
11. HCL
12. JavaScript
13. Elixir
14. Clojure
15. Jupyter Notebook
16. Java
17. Scala
18. CSS
19. PLpgSQL
20. Kotlin
21. Ruby
22. Makefile
23. Groovy
24. TSQL
25. Gherkin

The high Success Rate of some of the languages at the top reflects low testing. These languages are not known for robust testing — they likely represent steps producing artifacts and output. Go, again, is an outlier here as dynamic languages include build and test steps for each of those languages.

Throughput

1. Hack
2. Slim
3. Elm
4. Mustache
5. Haskell
6. Jinja
7. Gherkin
8. Jsonnet
9. Jupyter Notebook
10. Apex
11. TypeScript
12. Swift
13. Ruby
14. Dart
15. Elixir
16. Go
17. C#
18. Kotlin
19. Blade
20. Scala
21. Python
22. LookML
23. Lua
24. CoffeeScript
25. Clojure

Hack sells itself as the language for fast development and high Throughput, and projects using Hack on our platform show up at the top. Interestingly, Apex shows a growing number of Salesforce projects adopting CI.

Methodology

- Every day between Dec 1, 2019 and Sept 30, 2021
- Only GitHub projects
- Only projects with more than one contributor
- Only workflows that ran at least 5 times
- 257,389,673 workflows
- 43,468 orgs
- 290,400 projects

In order to create this report, we pulled data for every day between December 1, 2019, and September 30, 2021. 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. (To be clear: if the workflow of that name for that project ran 5 times in the full history observed, we included it. The number of contributors is also over all-time on CircleCI, not just the analysis time.) In all, this constitutes data on 250 million+ workflows, from more than 43,000 organizations, and over 290,000 projects.

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.

circleci

REPORT AUTHOR

- Ron Powell

REPORT EDITOR

- Molly Fosco

REPORT CONTRIBUTORS

- Michael Stahnke
- Jeremy Goldsmith
- Brandon Byars
- Coty Rosenblath

REPORT DESIGNER

- Alex Moran

ACKNOWLEDGEMENTS

- Melissa Santos