2020

State of Visual Testing

Learn how teams utilize visual testing to build and deploy flawless frontends

percy

Introduction

Visual testing has emerged in recent years as a modern solution for teams developing, testing, and deploying frontends. As more teams adopt visual testing and the ecosystem matures, we at Percy want to share our experience, guidance, and insights.

By analyzing our platform data along with collected survey data, we've set out to provide the most comprehensive visual testing report yet.

Founded in 2015, **Percy** has pioneered the path forward for visual testing. Engineering teams from companies like Condé Nast, Basecamp, Intercom, and Canva use our all-in-one visual testing and review platform.

Learn more at percy.io >

Whether you're just getting started with visual testing or you're looking for guidance to mature your current efforts, this report is for you. In this report, we're sharing an overview of how visual testing works, as well as our benchmarks and best practices for implementing visual testing. We're thrilled to publish this information and hope it starts a meaningful conversation about the future of visual testing.

Table of contents:

A word about the data

In compiling this report, we collected and analyzed data from survey responses and our platform.

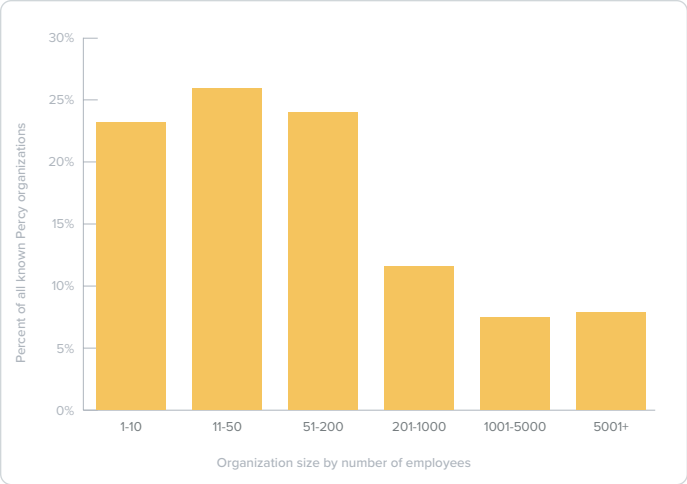
Percy platform data

The Percy platform data referenced in this report includes data from January 2018 to December 2019.

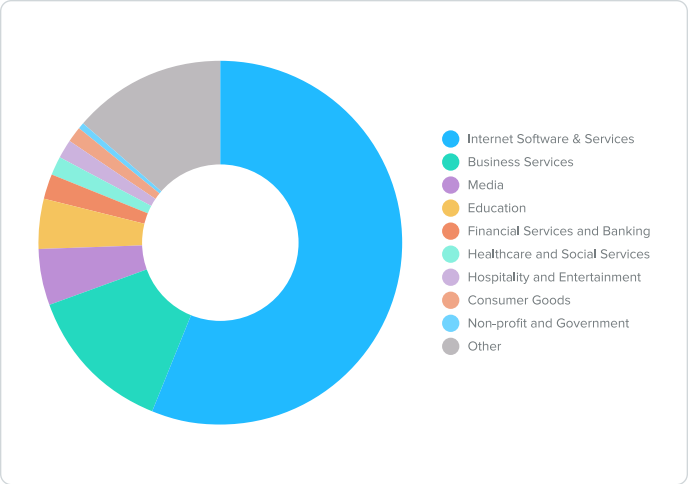
That data is made up of:

- **35,000 unique projects.** Percy projects align with organizations' unique applications, websites, or component libraries that they're testing. Percy builds are organized in projects.
- **350 million screenshots rendered.** A screenshot is a rendering of a web page or component. Individual screenshots get grouped into snapshots that include multiple permutations across widths and browsers.

We have a wide range of users and teams utilizing Percy—from open source projects to Fortune 500 companies. Here is a breakdown of organizations using Percy by size and industry:



Known Percy organizations by size



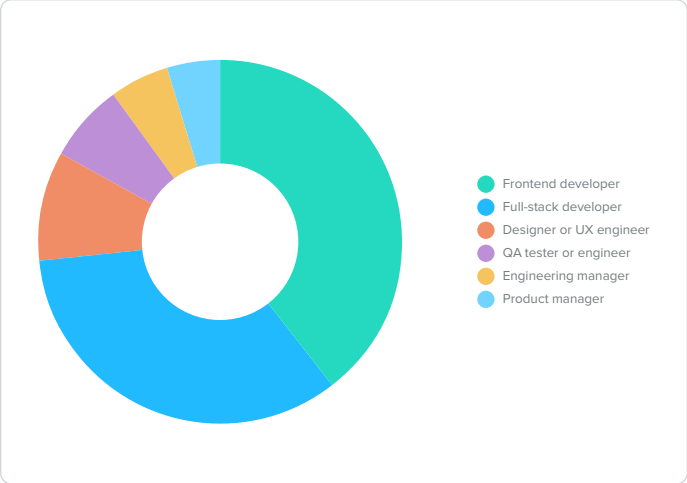
Known Percy organizations by industry

The Percy platform data presented in this report will help us gain an understanding of how teams are currently utilizing visual testing and help us to provide benchmarks for several aspects of visual testing.

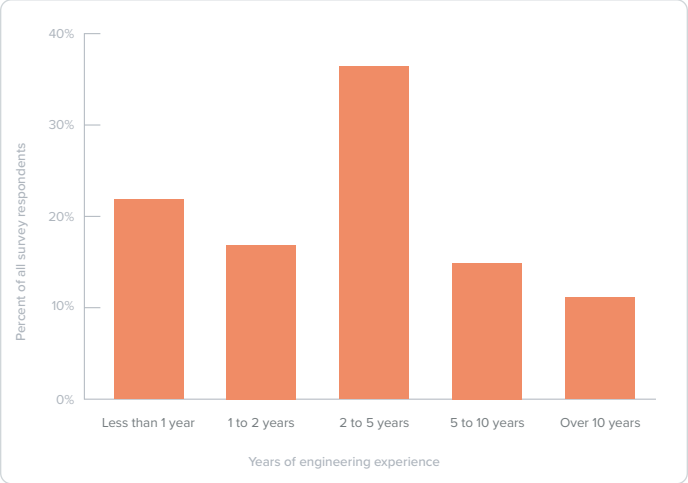
Frontend testing survey data

In November and December 2019, we received over 1500 responses to our frontend testing survey. We distributed our survey via Twitter and to our email database.

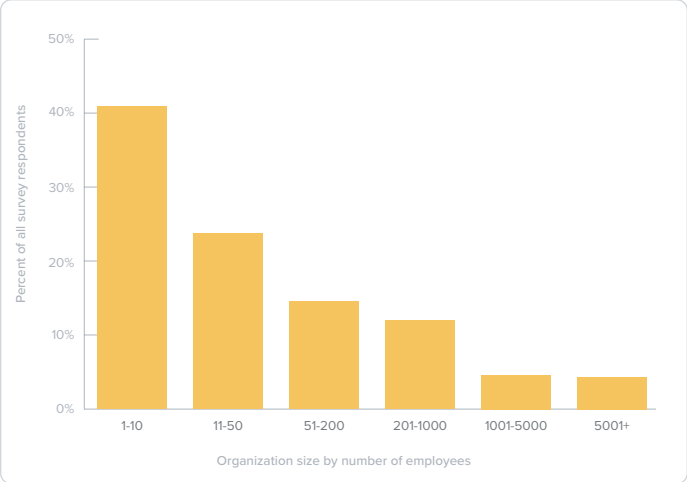
The survey data presented in this report is a diverse and representative look at the developer community with over 100 countries represented. Here is a breakdown of survey respondents by job title and years of experience, as well as their organizations by size and industry:



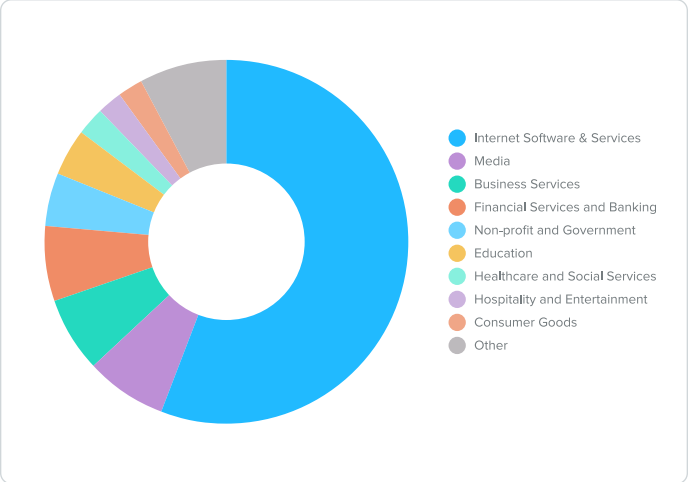
Survey respondents by job title



Survey respondents by years of experience



Survey respondents by organization size



Survey respondents by industry

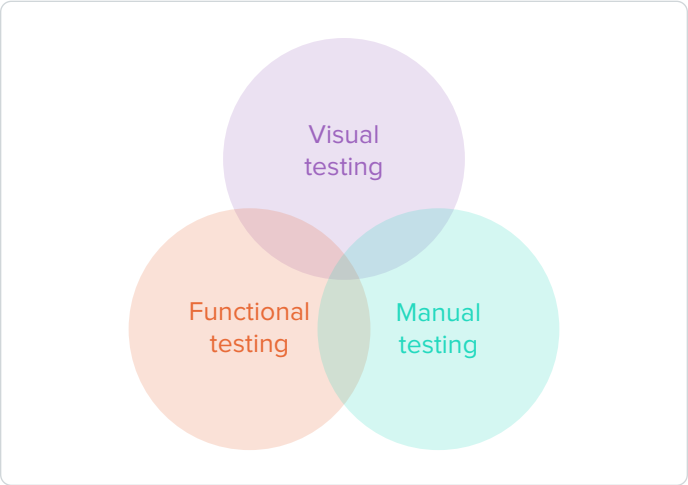
We really appreciate the contributions from the developer community and believe it provides an accurate look at how teams approach frontend testing and perceive visual testing.

Visual testing overview

Sometimes referred to as automated UI testing or visual regression testing, visual testing checks software from a purely visual standpoint.

Visual testing is all about what your end users see and interact with.

Visual testing is a great complement to functional and manual testing but provides distinct added value. In this section, we'll examine the commonalities and differences.



Visual testing, functional testing, and manual testing

Visual testing vs. functional testing

Functional testing checks that software is behaving as it should. By combining test automation and image processing, visual testing checks that software *looks* as it should.

Functional testing refers to software testing practices such as unit testing, acceptance testing, integration, and end-to-end testing.

Functional testing is a critical part of the software development lifecycle but is not equipped to check applications' visual elements.

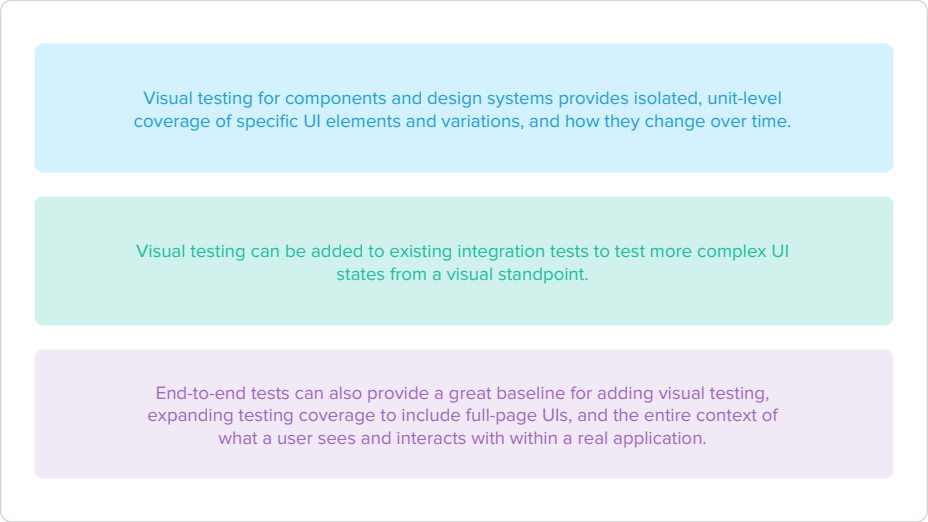
Trying to assert visual "correctness" with test assertions results in a test suite filled with brittle tests that constantly need to be rewritten. There are so many things that can make your tests "pass" while resulting in visual regressions—CSS class attributes can change, overriding classes can be applied, and the list goes on. By asserting that certain classes are applied or checking for styles, you're not actually testing your frontend visually. It's also hard to account for visual bugs caused by rendering in different browsers or across different screen sizes.

Visual testing solves those challenges by testing software from a purely visual standpoint regardless of the code underneath.

Visual testing is similar to functional testing in that it's designed to be an automated process that runs alongside code reviews. Unlike functional testing, however, visual tests don't pass or fail. Visual testing simply detects visual changes and provides a review process to determine whether or not the changes are correct.

Visual testing can be done at different levels of granularity, from components to end-to-end tests, as outlined in the figure on the right.

In many ways, visual testing is more flexible than our traditional categorizations of functional testing and can be done on static sites, HTML templates, PDFs, and more.



Where visual testing fits in with unit, integration, and end-to-end testing

Visual testing vs. manual testing

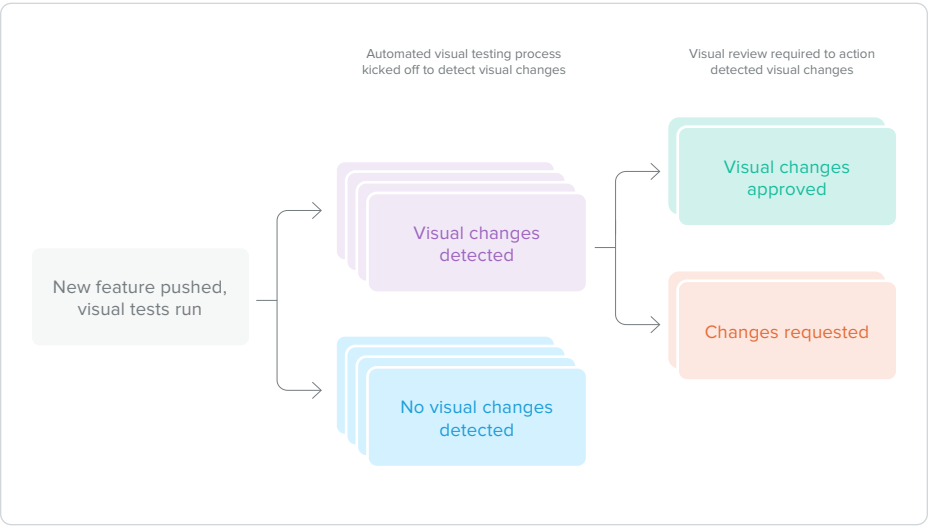
Whether done by individual developers, a dedicated internal team, or a hired external firm, manual QA is inherently resource-intensive and error-prone.

By combining human review processes with workflow automation and technology, visual testing provides a more accurate and scalable approach to UI testing.

Similar to manual QA, the human component is crucial in visual testing. Although automation offloads the initial work of detecting visual changes, visual testing requires human judgment.

Without the same biases and limitations that humans have, visual testing is better at preventing even the most subtle visual regressions.

It also provides feedback when elements haven't changed—something that is harder to achieve with manual QA alone.



Visual testing and review workflow

Most importantly, visual testing has virtually no limitations to the breadth of UI covered, the depth of states, browsers, and devices checked, or the frequency with which it can run.

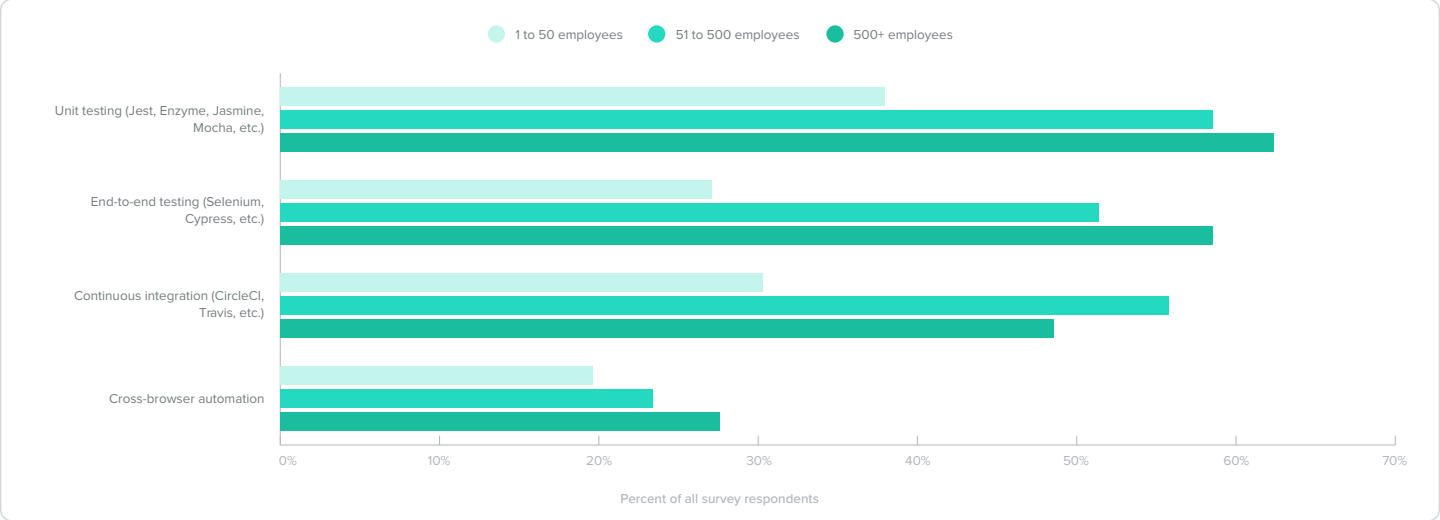
The evolution of visual testing

Software development teams face more pressure than ever to keep up with mounting consumer expectations and competition. Releasing software updates more frequently and across more devices increases the risks of releasing bug-ridden software.

To help software development teams minimize those risks, the software testing market has exploded with methodologies, tools, and processes over the last several years. Software testing and the culture surrounding it has been instrumental in helping teams build and deploy high-quality products.

From our survey, 58% of respondents utilize some form of software testing.

For respondents from companies with more than 500 employees, software testing is even more commonplace, with 76% utilizing unit testing or end-to-end testing in isolation or together.



Current software testing and automation practices by organization size

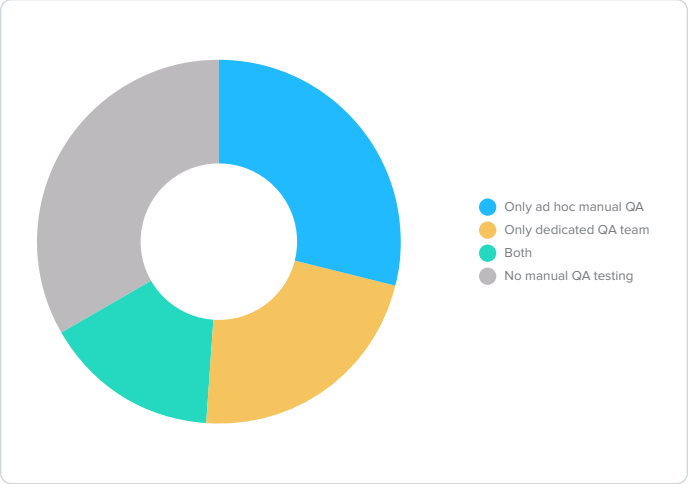
Additionally, test automation supported by continuous integration and continuous deployment (CI/CD) is a popular practice amongst more mature organizations. Cross-browser automation is gaining traction to help teams automate software testing efforts across fragmented browsers, operating systems, and devices.

Continuous integration (CI) helps teams integrate their code into a shared code repository seamlessly, while **continuous delivery** (CD) enables efficient deployments to production. CI/CD tools integrate with source control systems to commit, build, automate tests, and deploy with little human involvement.

Much of the current testing tooling and culture revolves around ensuring software works as intended, leaving gaps in how teams test frontends from a visual standpoint. To fill those gaps, manual testing has been the de facto frontend testing practice.

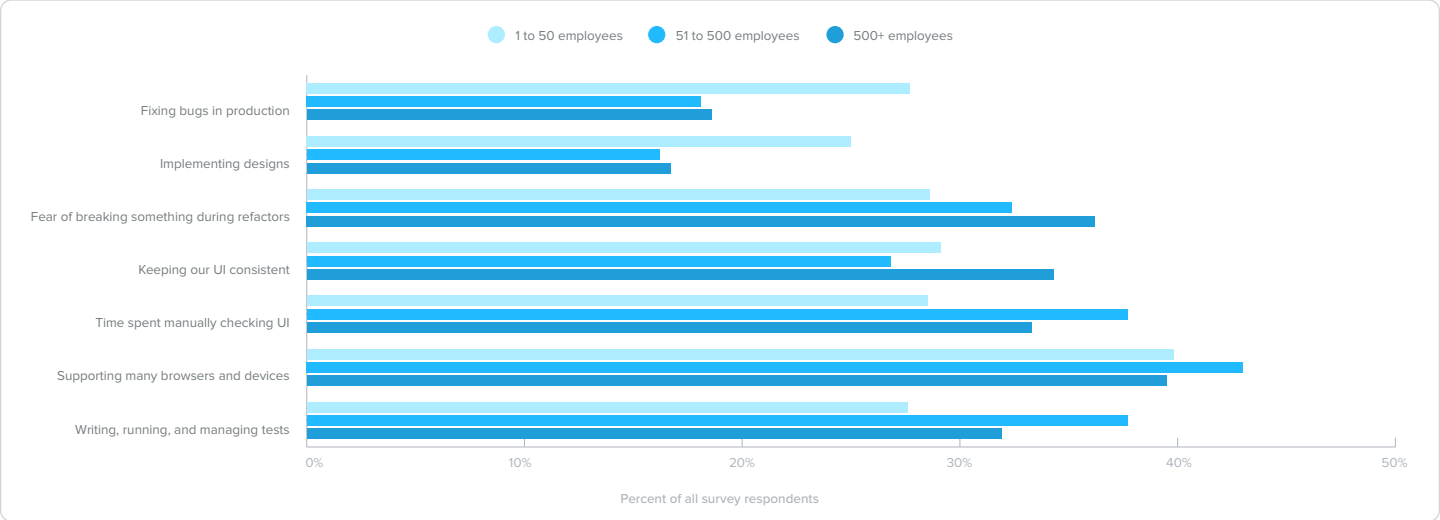
Over 67% of all respondents do some kind of manual QA, and for 27% of respondents, it's their only form of software testing.

With manual QA alone, many of the challenges of frontend testing persist. In our survey, we asked what the most significant challenges are when it comes to frontend testing.



Current manual testing practices

The top-cited frontend challenges are: supporting multiple browsers and devices, time spent manually testing, and writing, running, and managing tests.



Biggest frontend testing challenges by organization size

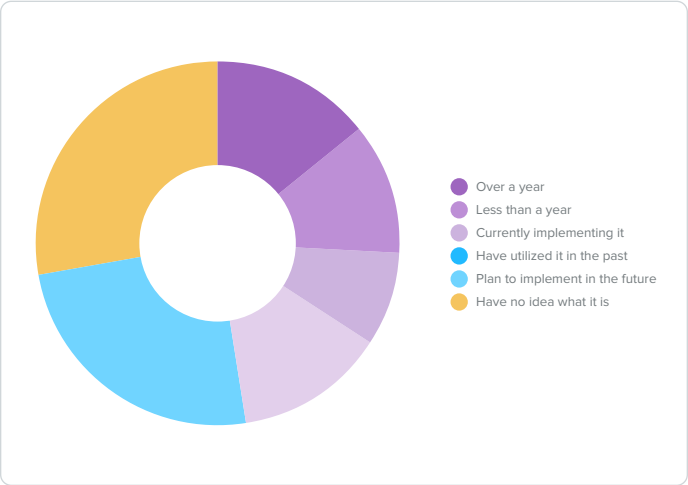
Sentiments across different organization sizes varied. While smaller organizations found it more challenging to fix bugs and implement designs, larger organizations cited the fear of breaking something during refactors and keeping their UI consistent as their most significant challenges.

Visual testing adoption

There have been a few attempts to automate UI testing in the last few years, including open-source screenshotting libraries and record-and-replay tools. As technology and tooling have evolved, however, visual testing has emerged as the most scalable and holistic approach to testing visual elements of frontends.

From survey data, 47% of respondents are currently involved in projects involving visual testing or have in the past—another 24% plan to in the future.

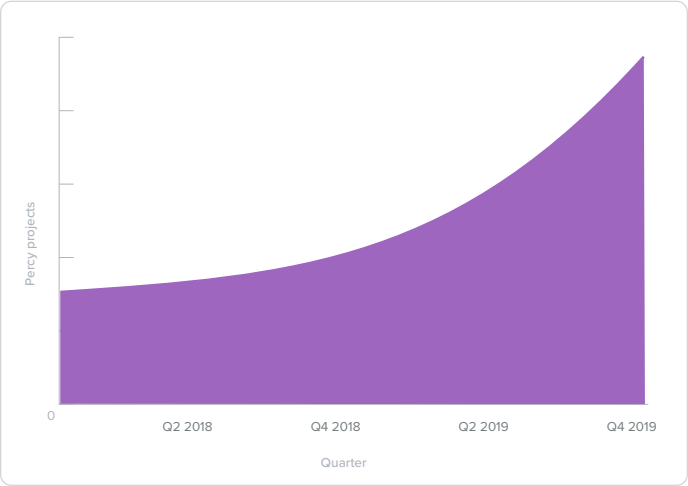
Visual testing has really only been commercially available for the past four years, so it's not too surprising that 28% of respondents weren't familiar with visual testing. Despite that statistic, we've seen adoption accelerate in the past few years.



Experience with visual testing

In the past two years, we've seen an average of 36% quarterly growth in new projects created on the Percy platform.

We foresee that growth to accelerate as teams automate more of their QA work and augment their functional tests with visual testing.



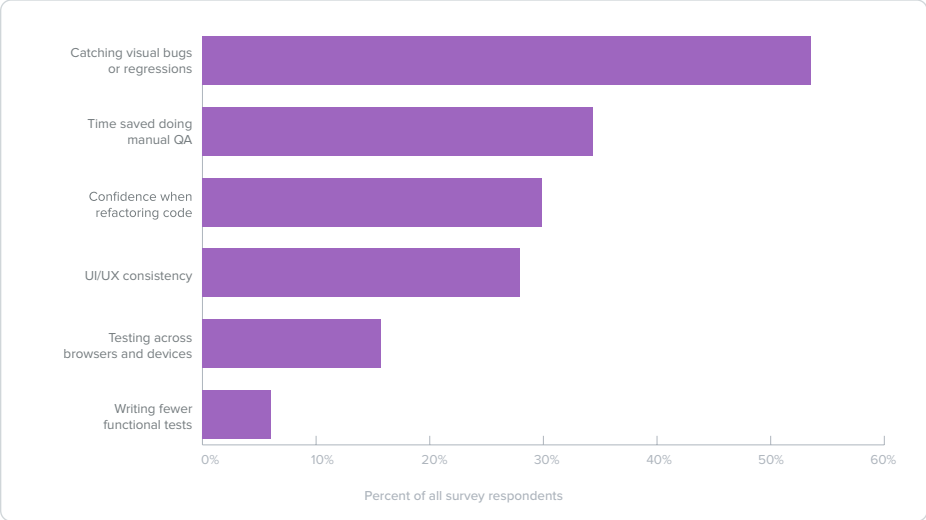
Cumulative project creation growth on the Percy platform

Benefits of visual testing

In this section, we'll look into the major advantages that visual testing has over functional testing and manual QA. From day-to-day operation to specific use-cases like website redesigns, visual testing helps teams save time and have more confidence in their UI. In our survey, we asked which benefits of visual testing are the most important.

54% of survey respondents cited catching bugs or regressions as a benefit of visual testing.

The next most important cited benefits were saving time spent manually testing, and confidence when doing refactors.



Perceived benefits of visual testing

Getting complete confidence with visual testing

When making code changes, it's often a fear of the unknown that causes the most stress—whether it's fear of breaking something or uncertainty about the full scope of a design implementation. That risk, and the inability to mitigate it, is why teams spend time and resources manually looking for visual bugs. It's also why teams end up trying to write functional tests to prevent visual regressions.

Visual testing automates that work, empowering you to merge and deploy with full confidence that your app will look exactly as it should across browsers and screen sizes.

In addition to preventing bugs, visual testing provides continuous visual feedback regardless of the context of the change or the "correctness" of the change.

In situations where your UI shouldn't change—like deleting CSS, refactoring CSS, or upgrading dependencies—visual testing gives you confidence that your entire UI has remained stable. That built-in coverage not only extends to the breadth of your UI—from your most critical flows down to your 404 page—but also to the combinations of those pages across browsers and screen sizes.

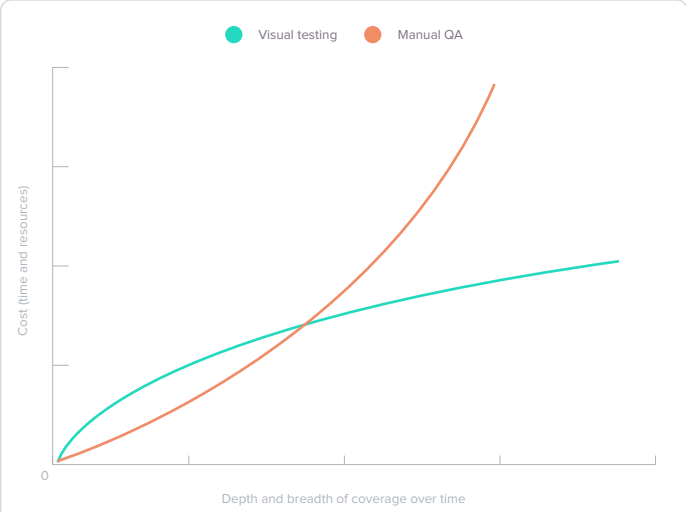
Saving time and resources with visual testing

Relying on humans to spot visual changes is time-consuming and asynchronous from development cycles. Placing the responsibility of manual QA on individual developers breeds fear of shipping updates, negatively impacts productivity, and ultimately leads to visual bugs slipping into production. To offset those downsides, organizations turn to dedicated manual QA functions. Hiring internal resources or working with external firms, however, can come with high overhead and operating costs.

Either way, you're losing valuable engineering time or spending resources on overhead and operating costs that don't scale as you improve your coverage or grow your product.

Visual testing scales with a level of precision not feasible with the human eye, at a faster rate than the brain can work, and at a fraction of the cost.

The ability to scale across the full breadth of your product at the speed of development is crucial, but manual QA lacks that scalability. The cost of manual QA increases linearly but gets more expensive as you add more complexity. Visual testing has virtually no limitations to scale and has a low incremental cost that doesn't increase exponentially with complexity.



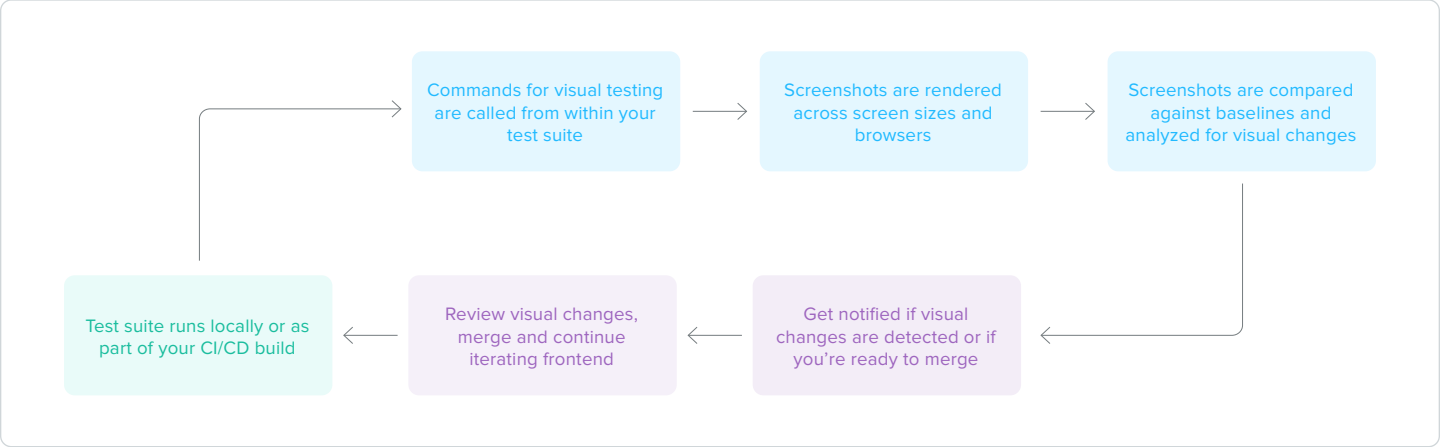
Cost of visual testing versus manual QA

The time-saving and confidence-boosting benefits of visual testing also extends to teammates other than developers.

For designers verifying the implementation of designs, product managers staying in the loop, or product marketers grabbing updated UI screenshots, visual testing provides immense value to cross-functional product teams.

How visual testing works

Visual testing works by comparing UI screenshots against baselines to see if anything has changed between the two. By integrating with your test suite and workflow, visual testing is designed to run on every commit, providing continuous visual feedback and detecting visual changes across browsers and screen sizes. The visual review process is vital to evaluate surfaced feedback.



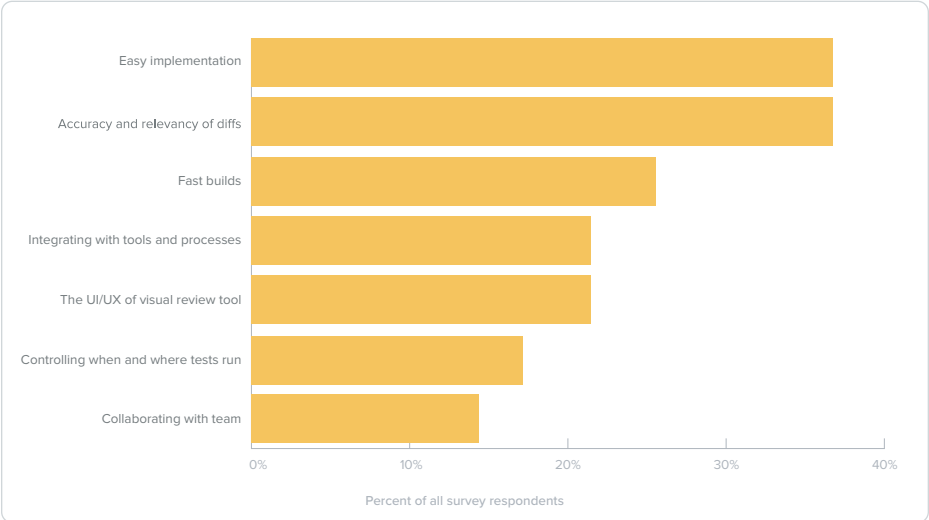
How the visual testing and review process works

To make the visual testing process as efficient and useful as possible, visual testing platforms like Percy handle the end-to-end process and technology. In our survey, we asked what was most important when it came to visual testing platforms.

Easy implementation and the speed and accuracy of diffs are the most important aspects of visual testing platforms.

In this section, we'll review how visual testing works and some benchmarks within each area:

- Integrating visual testing
- Running visual test suites
- Snapshot rendering
- Visual diff detection
- Visual review workflow



Most important aspects of a visual testing platform

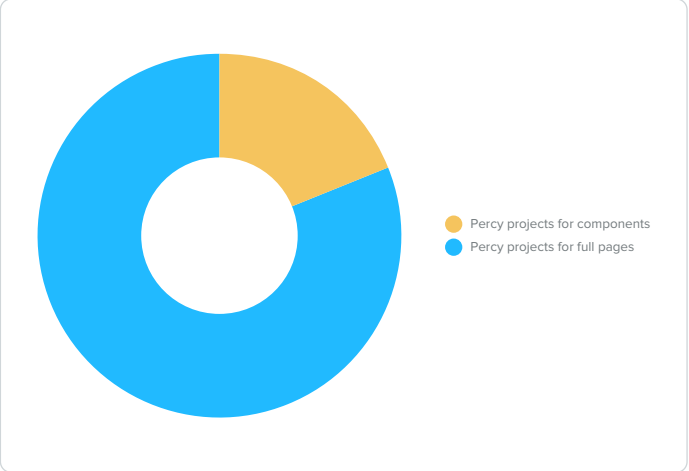
Integrating visual testing

When setting up your visual tests, there are two major paths—**visual testing for components or for full pages**. Component-driven development is getting more popular; 87.5% of our survey respondents reported that they utilize components as part of their development processes.

Our platform data, however, paints a different picture.

Just 19% of Percy projects test components rather than full pages.

That may be because the processes around component-driven development and the degrees to which frontends depend on components still vary widely. Depending on the nature of your project and your existing test suite, they both provide different levels of granularity and benefits.



Active Percy projects by type of integration

Visual testing for components

Testing components that get used in different variations and contexts across your app, or even across several properties, is a good way to get comprehensive visual testing coverage. It's also a good way to isolate visual changes on specific components rather than having to review a single change across several impacted screens.

If you have a well-maintained component library or utilize a component library like Storybook, this approach may be the fastest way for you to get started with visual testing.

Percy integrates directly with Storybook, consuming the built UI components for visual testing across browsers and widths. This approach is perfect if your application or website depends solely on your component library.

Learn about Percy's Storybook SDK >

Visual testing for full pages

Adding visual testing to full pages is a great way to get comprehensive test coverage with relatively low effort. Snapshots can be easily integrated into acceptance tests or end-to-end tests, providing screen renderings of all your essential flows and pages, as well as more complex views like menu dropdowns and different user states.

It can be integrated as an extension of your existing functional test suite or a simple standalone script like Percy's PercyScript that captures different parts of your application for visual testing.

With Percy, you can integrate visual testing into web applications built in any language or tested with any testing framework. Percy also has SDKs to integrate with your end-to-end test frameworks such as Cypress.

Learn more about Percy's SDKs >

Because components exist within the full context of your application—not as isolated elements—**visual testing for full pages may be a more straightforward approach to visual testing.**

Running your visual testing suite

In addition to integrating with your stack, visual testing is designed to run alongside your day-to-day workflow. To get consistent and truly automated visual testing, most teams opt to run their tests along with their continuous integration and continuous delivery (CI/CD) processes.

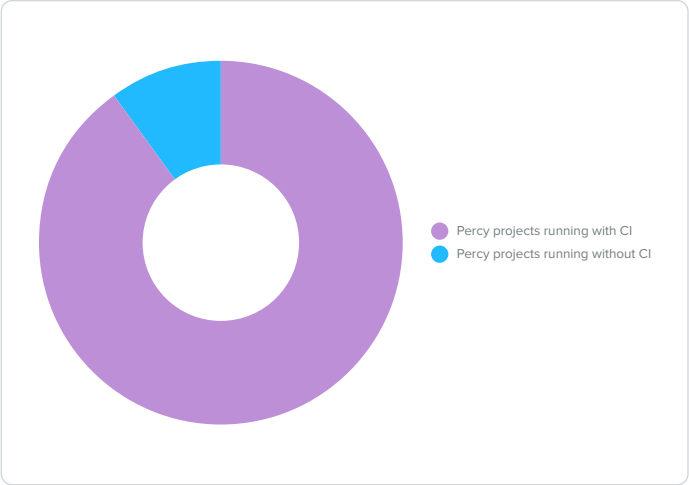
Percy integrates with all the popular CI/CD services such as CircleCI, Travis CI, Jenkins, Buildkite, and more.

Read about Percy's CI integrations >

90% of all Percy builds run as part of CI/CD.

By integrating visual testing into your CI/CD practices, you can get visual feedback earlier in the development lifecycle. Getting visual feedback earlier helps you catch visual regressions and bugs before they make their way to production and can aid in the design implementation and review process.

A fast and fully-integrated CI/CD pipeline provides a great foundation for adding visual testing. If you have a comprehensive test suite and your team is already accustomed to following CI/CD best practices, you're likely already well-positioned to get started with visual testing.



Active Percy projects by workflow integration

Screenshot rendering

Because browsers are complex and the technologies that websites depend on are equally so, screenshot rendering is incredibly resource-intensive. That's why cloud-based services handle the end-to-end rendering process, decoupling it from CI/CD with the infrastructure necessary to scale.

Percy takes a unique approach to snapshots and rendering. This is how it works:

- Snapshot commands are called from within a test suite
- Page assets including CSS and images are captured and sent to Percy, along with the DOM snapshot
- Percy freezes CSS animations and stabilizes dynamic elements for rendering
- Screenshots are rendered across browsers and screen widths
- Screenshots are compared against baseline screenshots to detect visual changes

Percy pioneered the use of the **DOM (Document Object Model) snapshot** to recreate the most deterministic rendering of screens and components in our own environment to avoid slowing down your tests and builds.

Visual diff detection

Once screenshots are rendered across selected widths and browsers, they are compared against baselines and analyzed for visual diffs.

There can be one of two outcomes for each comparison—either a visual change is detected or not. If diffs are detected, a review is necessary to determine whether they're intentional and ready to approve or unintentional and need to be remediated.

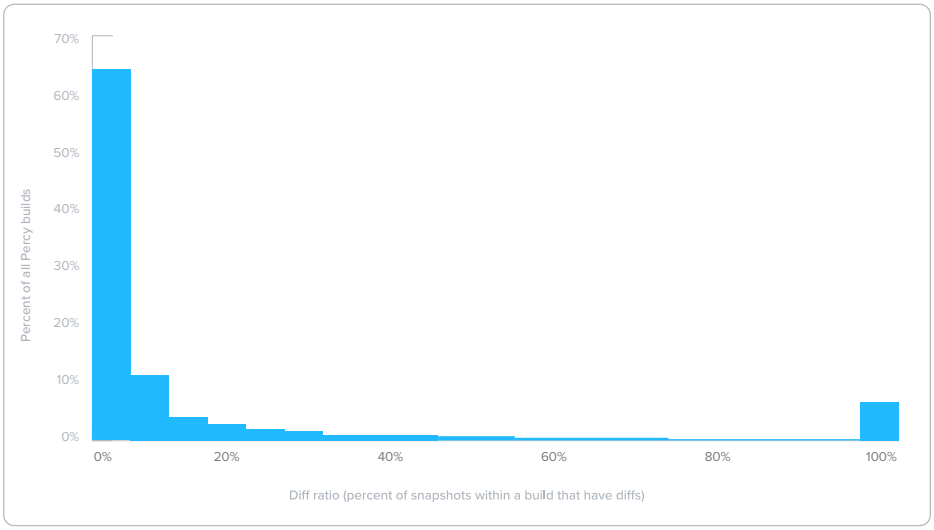
Sometimes referred to as perceptual diffs, pdiffs, CSS diffs, and UI diffs, **visual diffs** are the computational difference between two snapshots. In the Percy UI, diffs are highlighted in red so you can easily see what has changed.

Keep in mind that when taking snapshots across various screen sizes and browsers, you are comparing individual screenshots across permutations. All comparisons might not have diffs, and the diffs might vary from permutation to permutation—some may be intentional styling changes, while others might be unintentional visual bugs.

Visual diff detection is a big part of visual testing, but it's also valuable when visual changes are not detected. Having the assurance that your UI has remained stable and unchanged is a benefit of visual testing that's often overlooked, but it's by far the most common visual testing scenario.

Of all Percy builds, 64% are visually unchanged. Of builds with diffs, the average diff ratio—the percent of snapshots within a build—is 12%.

The distribution of builds by diff ratio shows a spike at 100% because it's common for a shared style change to result in changes to all snapshots within a build.



Distribution of Percy builds by diff ratio

Visual review workflow

A quick and easy visual review process is crucial to the success of your visual testing efforts. Getting relevant feedback and timely alerts are also important to consider when implementing visual testing.

Workflow and integrations

In addition to being kicked off as part of CI/CD processes, visual testing needs to provide relevant feedback through the most accurate screenshot comparisons. Because visual testing is so embedded into developer workflows, it depends heavily on your branch history and the master branch to pick the correct baseline.

Percy's baseline picking logic aims to include only changes made in the branch itself by looking at the branch history and picking the "common ancestor" as the baseline.

Learn more about baselines >

Integrating with a source code repository also keeps visual tests in sync with code reviews. A source control integration is also the best way to get notifications where they're most helpful and actionable—in your commit and pull request or request checks.

Percy offers several **source control integrations** for GitHub, Bitbucket, and Gitlab to provide synchronized visual reviews and notifications. Our webhooks and Slack notifications provide additional mechanisms to stay up-to-date.

Learn more about Percy's integrations >

Reviewing visual changes

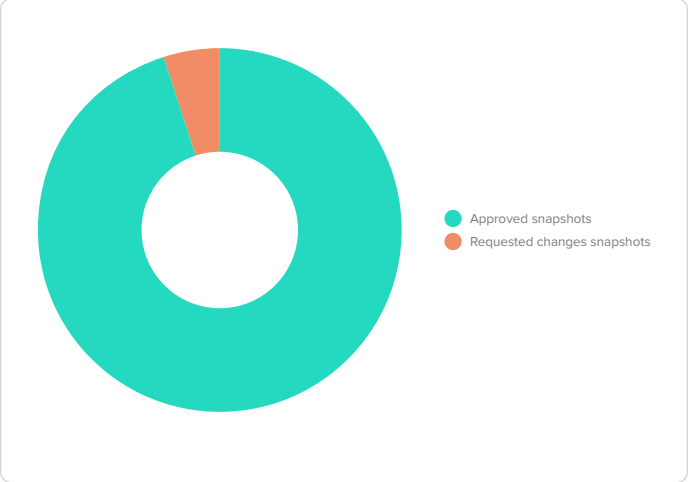
When a visual change has been detected, it's up to you to discern whether the changes are intentional or not.

To make that process as efficient as possible and minimize false positives, visual testing platforms use advanced diff detection strategies and stabilization techniques. They also aim at clearly highlighting visual diffs, grouping similar changes, and giving you tools to debug if necessary.

It's also crucial to have a mechanism in place for you to designate whether the detected visual changes are intentional or unintentional. Approvals unblock code reviews, empowering your team to deploy with full confidence in your UI. By requesting changes, you can alert your team that something doesn't look right, so you can remediate before merging.

Of all actioned Percy builds with diffs, 96% were approved compared to 4% of builds with changes requested.

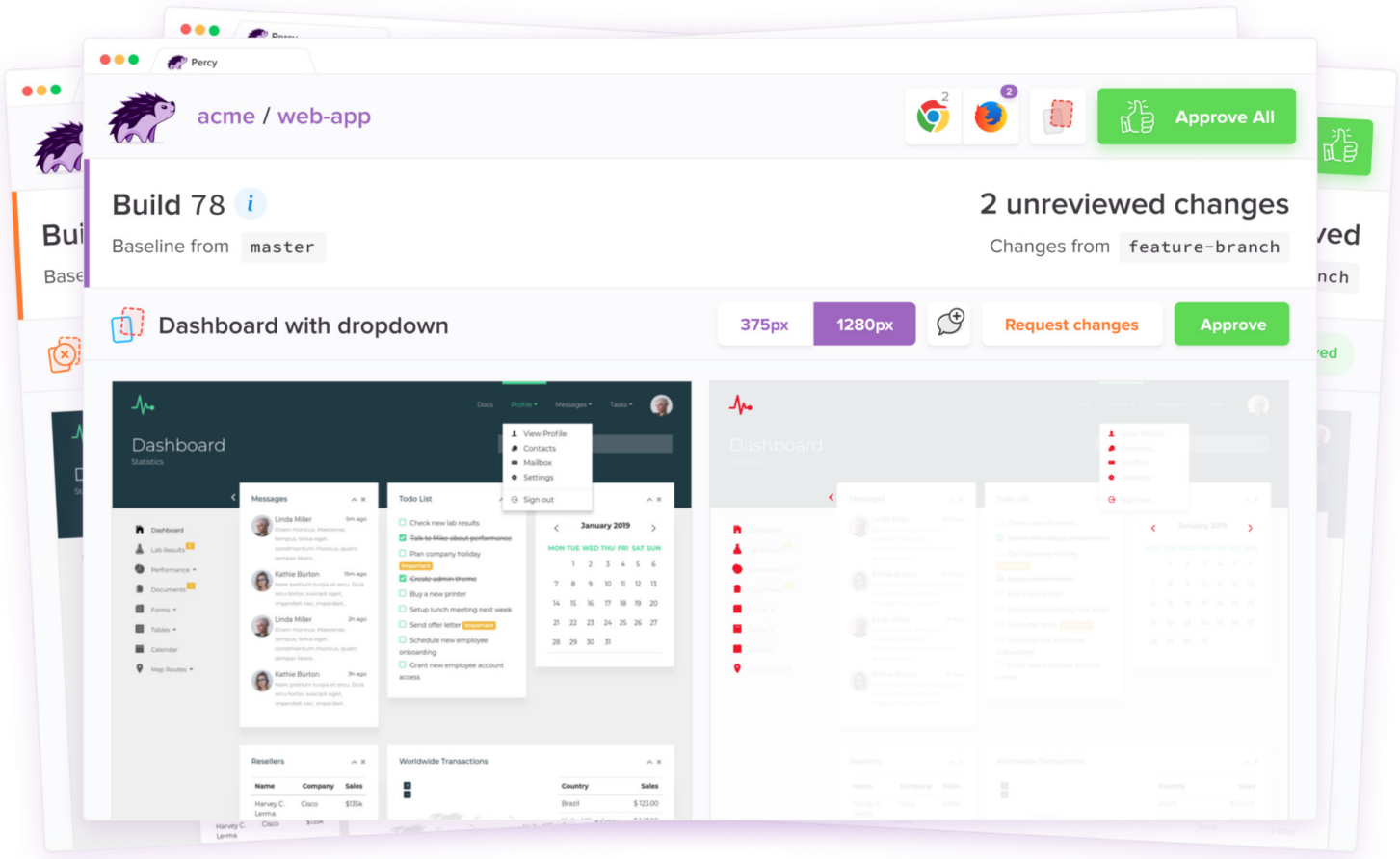
We've found that being able to communicate with your team about visual changes is key to keeping track of and prioritizing visual changes. Asking questions about designs, blocking approvals, and alerting your team of visual bugs are all crucial to visual reviews.



Reviewed snapshots by action since Percy released the request changes feature in August 2019.

Visual testing best practices

Our goal at Percy is to help teams get continuous, relevant, and actionable visual feedback at scale. With our visual testing and review platform, we've been able to help teams big and small automate manual QA, catch visual bugs, and deploy with complete confidence.



Screenshot of the Percy build UI

We also have years of experience building the processes and guidance necessary to implement visual testing effectively. In this section, we'll provide some recommendations on how to best implement visual testing for optimal coverage and workflow.

Visual coverage best practices

When getting started with visual testing, it's important to think about and plan the level of visual coverage that is feasible and useful to your team.

Depending on your test suite robustness and size of your app, we recommend starting with the pages that keep you up at night.

Often those are pages and elements that:

- Are most important to revenue and user experience
- Have the most teams working on them
- Are the most brittle or have the most legacy code
- Have the lowest test coverage

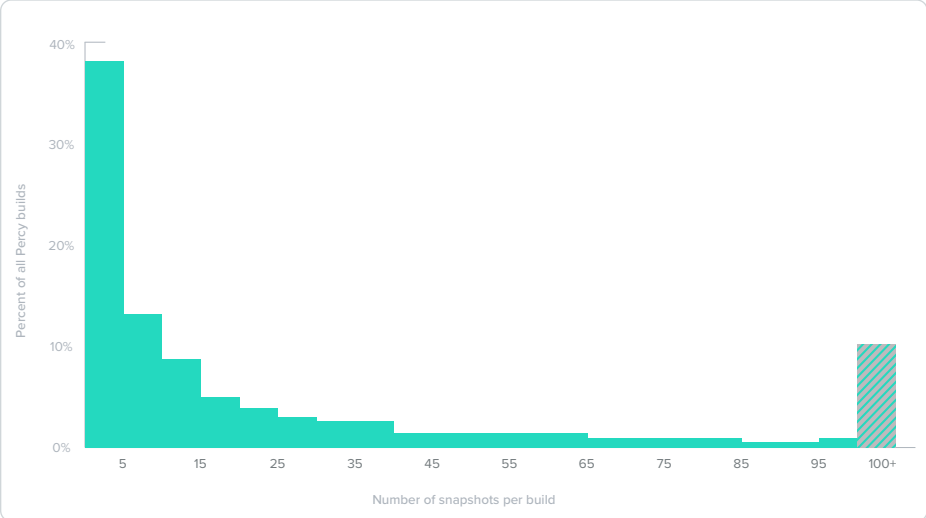
Similarly to how code coverage gives a sense of how well functional test suite covers your application, **visual coverage** is used to describe how much of an application is covered by visual testing. Rather than looking at how many lines of code are covered, visual testing is about which parts of your UI are most important to test.

Alternatively, focus your efforts on getting visual coverage on screens and templates that get the most traffic or the components that get utilized the most. For a small marketing website, that might be 10 snapshots, but for a complex web app, it might be 100 or even 1000 snapshots.

In addition to obvious pages or components, you should also consider their different variations. More complex states such as dropdowns, logged in states, and page interactions tend to be the most commonly overlooked during manual QA and thus are more vulnerable to regressions. They're also more time-consuming to manually check, so the initial investment may be worth the return.

While every project is different, we found that projects have an average of 62 snapshots—excluding variations across widths and browsers—per build.

The average number of snapshots per project varies drastically by the type of project. Projects testing component libraries have 102 snapshots on average compared to 26 for projects testing full pages.



Distribution of the number of snapshots in all Percy builds

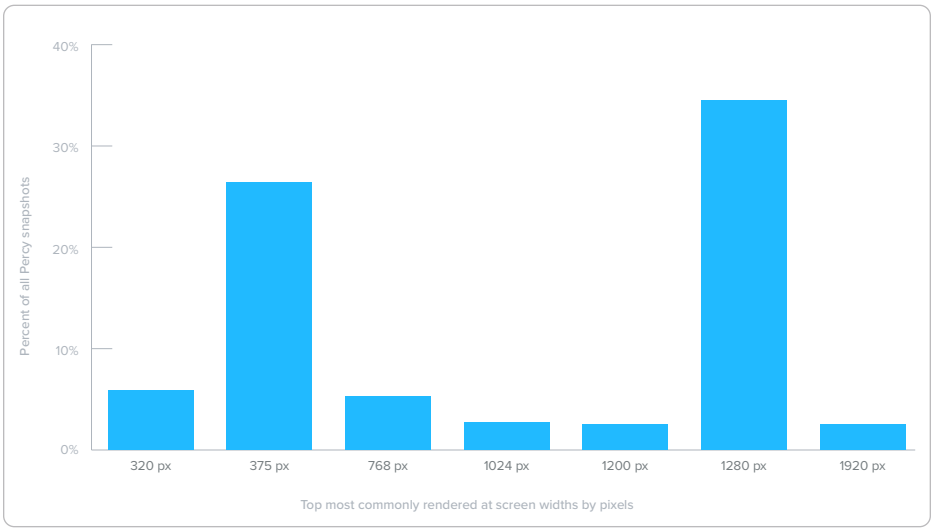
Regardless of how much visual coverage you start with, you should always be improving and expanding it over time, just as you continuously write new functional tests as you build new features.

Responsive visual testing best practices

It can be difficult to manually test your UI across different devices or screen sizes, which is why responsive visual testing is hugely valuable and a feature of visual testing to make the most of. When setting up your visual testing suite, we recommend testing across all the widths that you currently support.

Across all Percy snapshots, the two most popular width sizes are 1280px and 375px.

Percy builds test snapshots across an average of 2.6 widths, and 93% of projects include two or more widths.



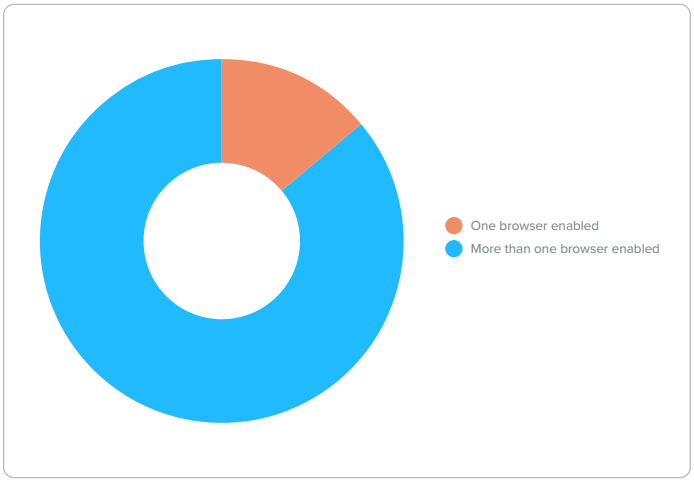
Distribution of Percy snapshots across screen widths

Cross-browser visual testing best practices

Supporting and testing across multiple browsers is also a huge challenge for frontend developers. Being able to catch bugs caused by subtle rendering differences across different browsers is also an important benefit of visual testing.

86% of Percy builds test across more than one browser.

Testing across multiple browsers is recommended for most teams getting started with visual testing, as it is a great way to expand your visual coverage with no additional effort.



Percy builds by number of browsers enabled

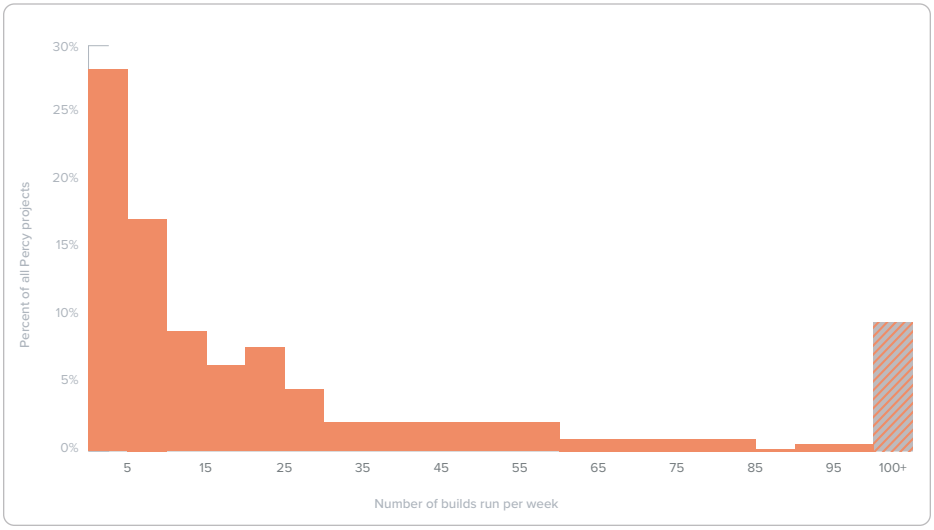
Visual testing frequency best practices

In addition to balancing the depth and breadth of visual coverage, getting the right visual testing frequency is critical.

You should strive to run your visual testing suite whenever code is pushed. For larger teams or teams working across disciplines within a monorepo, however, it may be cumbersome to run visual testing on every single pull request or commit. In those cases, it's easy to adjust your visual testing frequency to meet your needs.

From Percy project data, teams run 46 builds per week, on average.

As you can tell from the distribution on the right, more projects run fewer than 25 builds per week, but there are several projects that run more than 100 builds per week.



Distribution of Percy projects by frequency of builds per week

Regardless of how frequently your visual testing suite runs, the goal is to surface timely and relevant visual feedback to help your team deploy faster and with more confidence.

Conclusion

As visual testing adoption has gained traction over the past several years, the benefits have become clear.

When utilized alongside traditional testing methods, visual testing bolsters productivity and confidence for teams building and maintaining frontends.

We hope this report has taught you more about the mechanics of visual testing, as well as given you a glimpse into the visual testing ecosystem. By providing benchmarks and best practices, we hope to start a meaningful conversation about the future of visual testing.

In 2020 and beyond, we are looking forward to helping more teams get continuous visual feedback and close the visual confidence gap.

We'd love to hear from you and are happy to answer any questions you may have. You can reach us at hello@percy.io.

Ready to get started with visual testing? We welcome you to sign up for a free Percy account and check out these resources to help you integrate visual testing.

Percy resources:

Getting started with Percy overview

Percy SDKs overview documentation

Running Percy alongside CI/CD

Percy's source control integrations

Percy's visual testing plans and pricing