Continuous Testing Manifesto

rainforest

What is continuous testing?

Good QA enables developers to move faster, while having a consistent set of checks for the work they do. This holds them to a standard for quality, but frees them from having to manually check. Though teams can improve software quality using techniques such as pair programming or code-reviews, bugs will always slip through.

QA is a service to the development organization and should always be as lightweight as is practical. A QA strategy should continually strive for:

- Faster QA results
- More accurate QA results
- Fewer false positives
- More actionable results

The Continuous Testing Manifesto

The Continuous Testing Manifesto was developed to help your team ship higher quality software, faster by providing guidelines for what should be important in a QA process and what shouldn't.

This guide outlines each of the 8 pillars of continuous testing, with recommendations for how teams can start implementing them to the greatest effect.

The Continuous Testing Manifesto

1. **Modular Testing**
   Start with small, manageable chunks– aim for a modular system.

2. **Rightsize Your Approach**
   Good processes don't test everything; they take a balanced approach.

3. **Measure to Improve**
   If you don't measure results, you won't be able to show improvement.

4. **Shift Left**
   The earlier you can start testing, the easier most issues are to fix.

5. **Unit Test**
   Enable fast feedback, along with super-specific error reporting.

6. **Version Control**
   Keep your tests close to your developer workflow and create a record of tests.

7. **Continuous Integration**
   Create a safety valve for quality as early as possible.

8. **Pluggable**
   Integrate QA with development tooling to keep pace with development process.

1. Modular Testing

Make your tests as composable as possible by working out commonalities in your suite.

Most of the time, the common parts will be built upon to allow you to get in to the right state for your application. Avoid copy-pasting test steps — aim for a modular system.This applies for both automated and manual test suites. If using automation, make sure to have a common library of states you can reuse to get into the correct state. If using manual testing, you will need to find a test case management system that supports composable tests.

If starting from scratch with a new process, start with small, manageable chunks. Build your test suite in the following order for greatest effect:

Smoke tests
Covering the top three to five flows through your application. An example of an e-commerce site would be: login, sign up, checkout. Choose whatever might cause a headline in a newspaper or an email to the CEO if it broke.

Happy path tests
These are similar, but less important. Generally they are the most common paths through your application.

Regression tests
Bugs that people have already reported should not happen again. Add a test to your suite — whether a unit test or not — to make sure it doesn't.

Best Practice
Edge case testing can eat up an almost infinite amount of testing resources. Focus on edge cases sparingly to prevent them from slowing down QA.

2. Rightsize Your Approach

Good processes don't test everything; they take a balanced approach. This is often counterintuitive, as it's easy to assume that more is always better.

It is very common to want to execute every test, as well as every combination of browsers, or even every variant of a page. This will fast become impractical or even unnecessary once you have a large application.

If you do not have a fixed list of devices you support, you may use common tooling such as Google Analytics to aid you. From this, you should end up with a list of the most devices or browsers used by your customers. Focus on the top 95% of these, or 99% if you have a higher budget. Having an official policy on which browsers you support can also help you here.

Make sure you revisit this at least once per quarter, as the needed coverage may change.

Best Practice
Weight the priority of your bugs by browser popularity. Likewise, bugs on popular devices should be fixed first.

Which areas of your product should you test?

The two most practical ways to decide are by looking at usage data or bugs.

Using tooling such as Amplitude, Mixpanel or similar, work out which areas of your product are most active. These are often managed by a product team, who tracks feature usage. By looking at the most common paths, you may focus your testing efforts there first. If your tooling supports it, look at common flows through your application.

Developers often use error reporting software. This is a great source to use to focus testing efforts inside QA. Tracking defects by area of the product, developer, team and source of spec is a good start. This will often allow you to discover patterns. Any patterns found can guide process improvements and retrospectives with developers.

Best Practice
Run these tools in QA as well. You can use this to check that things are actually covering what's expected.

3. Measure to Improve

A QA process' main aim is to help the organization ship a higher-quality product. If you don't measure results, you won't be able to show improvement. Measure the fruits of your work and the current state of quality.

Implementing a more strategic approach to testing can have a huge impact on product quality, but measuring exactly how your QA strategy has made a difference can be challenging. The long-term success of any QA strategy depends on measuring change and communicating that change to the team at large, so it's important to measure the right metrics.

Primary QA Metrics

Number of Bugs

One of the most direct and essential measures of QA success is the number of bugs that reach customers. Log issues reported by (or which directly affect) customers with as much detail as possible, including date, product area, developer and team. Summarize this log on a weekly basis and report back to the team at the root.

Time to Fix

Tracking time-to-fix, or the amount of time between when something breaks and when it is fixed, is a critical measure of QA health. Time-to-fix provides insight into how effectively a development team is able to use the output from QA to triage and resolve bugs. The easiest way to measure this is the time between a failed build and the next passing build.

Issue Source

Want to take these key measurements to the next level? Split your issue tracking out by source. This level of detail will help you better understand the overall quality of the product, and identify weak areas in your process that need a boost. Examples of this include: external (i.e., customer), internal (i.e. missed by QA), automatic (e.g., error reporting), or test-case failures.

Secondary QA Metrics

Flakiness

Broken or unreliable tests aren't providing useful quality feedback to your team, so identifying poor quality tests is critical. If you're using automated testing, make sure to track which tests pass or fail intermittently. Tracking test failures over time will allow

you to identify the root cause of these failures, whether that's poorly written tests, human QA tester issues or test environment failures.

NPS

While NPS is a great end-measurement for your entire product, it's a trailing indicator. Because NPS takes a holistic view of customer satisfaction, it can be challenging to trace fluctuations in NPS to specific quality issues. As a result, NPS can be a useful indicator of overall quality, but it shouldn't be considered your primary measurement for the success of your QA strategy.

Best Practice
Note that there are no good leading indicators for production quality, only trailing indicators.

Test Coverage

Test coverage is one of the most popular measures of QA health, but it can be misleading and even dangerous if misunderstood. Test coverage by itself is not a measure of the quality or thoroughness of those tests. By relying too heavily on test coverage, teams can easily throw their effort into chasing down endless edge cases or overtesting features that aren't mission-critical.

Use test coverage to work out which areas are being completely untested and to determine where your team should focus their efforts most effectively.

The Key to Measuring QA Effectively

The measurements you use will change as your product evolves; make sure to regularly evaluate the metrics you're using to measure QA success and product quality. Refining and reevaluating how you measure your QA process and output is an important component of ensuring that your team continually hits a high bar for product quality, especially as your organization goes through periods of growth.

4. Shift Left

Testing earlier surfaces errors more quickly, tightening the feedback loop between QA and development. Shortening the entire development cycle brings other benefits too. A major one is reducing the risk of shipping the wrong thing, as you get feedback from users quicker.

Traditionally, QA gets involved later in the software development lifecycle. But slower feedback cycles create distance from the problem. Developers become forgetful and shift to other tasks, losing context. Additionally, code can change under them, complicating the process of fixing the issue.

Keep in mind that shorter release cycles aren't always ideal. A few potential reasons to avoid releasing more frequently include:

- Regulatory issues around changes.
- Shipping embedded or on-premise software.
- Having customers that are highly averse to changes.

How to Start QA Earlier in Development

Pull Request-based development combined with automatic environment creation. Every pull request automatically has its own environment setup. This can be home-rolled, or provided by an external service. This allows human-powered QA, or automated integration tests, earlier access to changes in the SDLC. Setting this up can be a pain, as following good dev/ops practices are a prerequisite.

Best Practice
SaaS services for PR-based dev include:
- Heroku Review apps
- Runnable

Generally, manual QA by non-developers isn't practical before pushing to a pull-request. This is due to speed and cost of the resources needed; developers are fast, expensive; manual QA can be slow. Automation helps solve this, if used correctly, as it may be executed on a developer's machine. This allows developers to get feedback as early as possible in the SDLC, while coding. They still have context around the errors as they occur, which makes fixing them faster.

Best Practice
Leverage tagging of tests to allow developers to run subsets effectively, allowing for even faster pre-push testing.

Many teams are now "shifting left," and starting their QA process earlier in the development cycle. But in most cases shift left testing is confined to unit testing alone. While unit testing is a core component of a continuous testing strategy, teams that want to optimize for speed and quality should shift functional testing left as well.

5. Unit Testing

While unit testing isn't usually done by QA, it is a great way to improve your product's quality. Although this takes effort, it will start paying dividends as soon as you have more than one developer or a non-trivial product.

Unit testing enables fast feedback, along with super-specific error reporting. Having good tests enables your developers to be more brutal with code changes, yet still be confident in the results.

On a non-trivial project or a non-trivial amount of developers working on the same codebase, unit-testing becomes essential to keep moving fast. Why? Developers will have to change things they didn't write and don't understand. Figuring out what might break without large amounts of risk, research or experience is hard.

To get greatest leverage from your unit tests, run them in the following order:

- The tests for files changed.
- The entire feature changed. Structure or tag tests by feature to help here.
- All tests. Usually this should be for every push and run via CI only, due to speed.

rainforest

6. Version Control

Version control brings advantages to manual and automated tests alike. Keep tests close to or within your developer workflow, along with the product's code.

While version control use is the de facto standard within dev teams, and more recently ops, it's much less common within QA teams. This shouldn't be the case.

Best Practice
Great QA processes are always tracked in a version control system, such as Git, Mercurial, or Subversion.

The Benefits of Version Control

1. Knowledge of what the expected behavior is for that code.

Version control of test cases means that you'll always be able to roll back to the test case versions that fit the version of the code you're working with.

2. Ability to review and accept tests using standard code-review processes.

Code review saves time, reduces bugs and helps keep the team on the same page. Those benefits translate to writing tests just as well as they do for writing code.

3. A historical log of test cases and how they change over time

Version control can also be a useful tool for gauging the progress of your test coverage and overall QA quality. For teams that want to strengthen their QA metrics measurement, version control creates a record of how tests have changed and evolved alongside the application.

Applying Version Control to Your QA Process

Applying the best practices for version control should be relatively straightforward if you're already using some form of it for your development team.

Use a Trackable Test Management System

Opt for test repositories and management systems that support editing and archiving. Even if tests are executed manually, make an effort to document how they're executed with every test run. Don't just overwrite or delete old test cases — archive them. This allows you to easily pull up older versions of tests if you need to roll back a feature version.

Test Early & Often

QA version control works best when it stays close to development. Shift left and test frequently, updating your tests as often as needed to stay in sync with the application's functionality. A modular test writing system can help you keep your test suite up-to-date without investing too much time in rewriting tests.

Keep Tests Short and Precise

Test cases that are focused on checking a single interaction are critical to successful version control. These granular tests allow you to pinpoint where the application or feature is breaking quickly and — in the case of outdated tests — update test flows easily.

7. Continuous Integration

The primary benefit of CI is not speed, but consistency. Continuous Integration provides a safety valve for quality as early as possible in the development process.

Good QA process is always part of a wider Continuous Integration (CI) process. Verifying your work regularly with an automated build helps catch bugs quickly, before they can snowball into larger issues. As a result, less time needs to be spent on testing overall. Each release, whether manual or automated, should be automatically run and reported on.

Maintaining an automated test suite is the best option for developing a scalable CI process. Human-based testing often can't scale to your needs, so integration may become a bottleneck. Further, if things aren't automated, they'll be missed.

Early Detection is the Best Prevention

The earlier you catch bugs, the faster you resolve them and the more cost-effective your QA process can be. To keep your time-to-fix as low as possible and releases running smoothly, shift testing "left" and aim to identify bugs as early as possible. A key part of "shifting left" is implementing (and optimizing) a continuous integration process. With a range of continuous integration server apps available, your team can stay up-to-date on the status of continuous integration builds right from their favorite Slack channel.

rainforest

8. Pluggable

Modern QA processes must integrate with your development tooling. In order to keep up with fast-moving development cycles, QA must be able to work within your coding process.

Your QA process should be "pluggable," or easily integrated into your larger development workflow. These three integration types allow for easy, deep integration into your process:

Bug tracker integration

Integrating your QA process with a bug tracking tool enables more efficient tracking of defects and prioritization. Also, it gets the right information to developers fast. Make sure your process includes things like state, screenshots and logs (HTTP, server, and console).

Integrate in to your CI

Even for manual processes you should block and wait for results. For automation, it must be runnable inside your CI system.

Have machine readable results

For example via an API. Ensure feature and or test status is available, as well as detailed results.

Why Quality Should Live Where Your Team Does

One way to remedy this is to bring QA notifications into the channels that your team already uses to discuss product and development issues. This ensures that bug alerts stay front-and-center for your team.

If your company communicates using a chat app like Slack, integrating bug alerts into your dev and QA channels keeps quality higher by preventing issues from slipping through the cracks. Some Rainforest users even have a dedicated QA channel to stay on top of functional testing results more effectively.

Learn More about Continuous Testing

Updating your QA strategy to keep up with the fast pace of development is challenging, and it can be a complex process. To apply the tenets of the Continuous Testing Manifesto and to develop a testing strategy that will work well for your organization, check out the resources below:

Continuous Testing: Striking a Balance Between Quality and Speed (featuring CircleCI)

Many teams strive to realize faster, more efficient development processes including continuous integration and continuous delivery. But moving faster often raises questions about quality tradeoffs. In this session, Rainforest CTO Russell Smith and CircleCI CTO Rob Zuber discuss what teams must do to achieve a balanced approach to QA that accelerates development goals.

Listen On-Demand

Continuous Testing Manifesto Deep Dive Webinar Series

In this webinar series, engineering leaders and developers from the Rainforest team dive deep into each tenet of the Continuous Testing Manifesto, discussing how teams can get the most out of their QA strategy and sharing tips for implementing continuous testing effectively.

Listen to this series and more on Rainforest's BrightTalk channel:

Listen On-Demand

About the Author

The Continuos Testing Manifesto was developed by Russell Smith. Russell is the co-founder and CTO of Rainforest QA. At Rainforest, Russ has helped hundreds of organizations implement faster, more effective QA processes using the key tenets behind continuous testing.

About Rainforest QA

Rainforest is changing the way QA is done in an era of continuous delivery. Our on-demand QA solution improves the customer experience by enabling development teams to discover significantly more problems before code hits production.

Hundreds of companies including Adobe, Oracle and Solarwinds use Rainforest to automate their QA testing process and easily integrate it with their development workflow via a simple API. Headquartered in San Francisco, Rainforest is a 2012 Y Combinator graduate funded by Bessemer Venture Partners and SVB Capital among others.

For more information, visit https://www.rainforestqa.com.