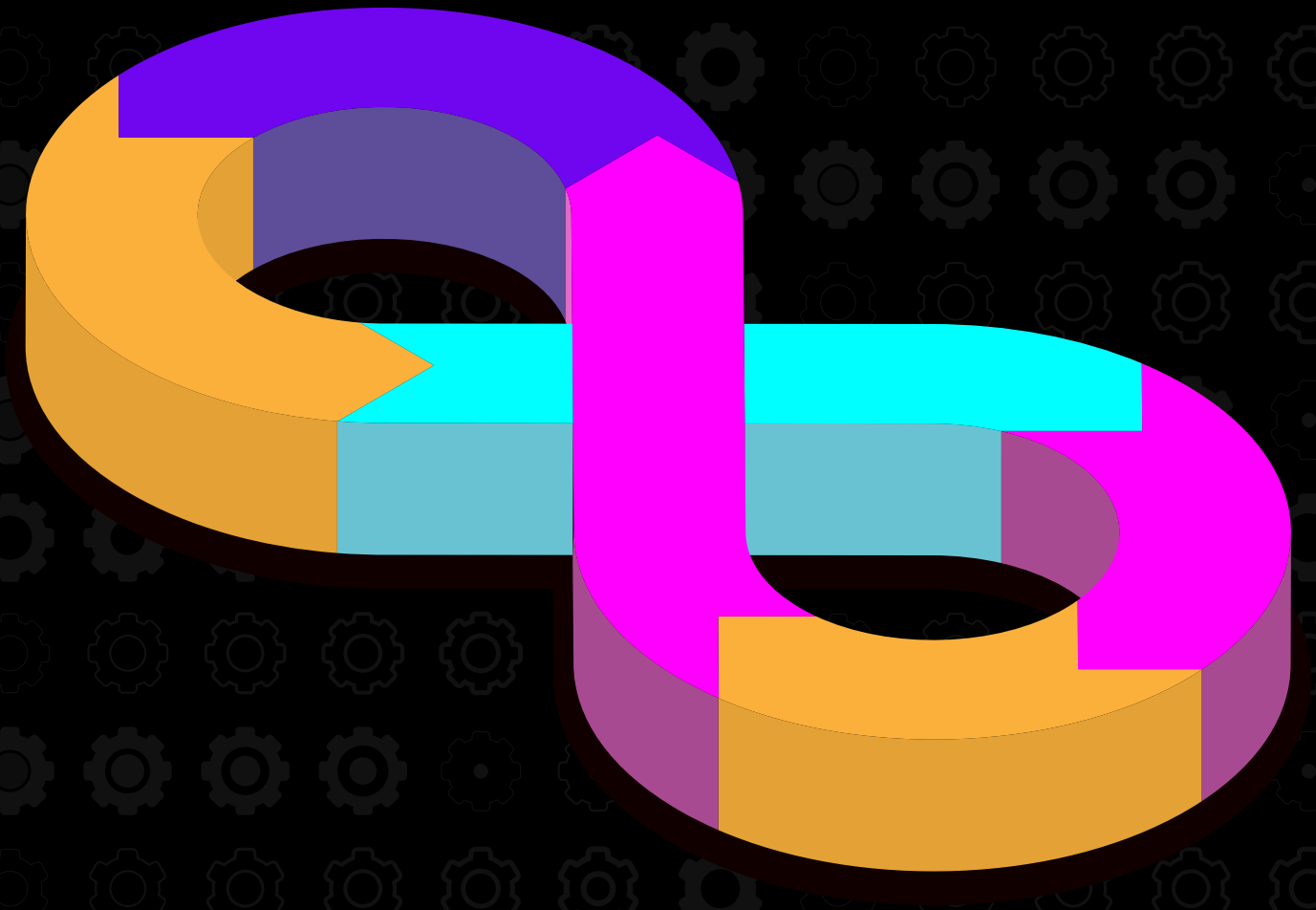
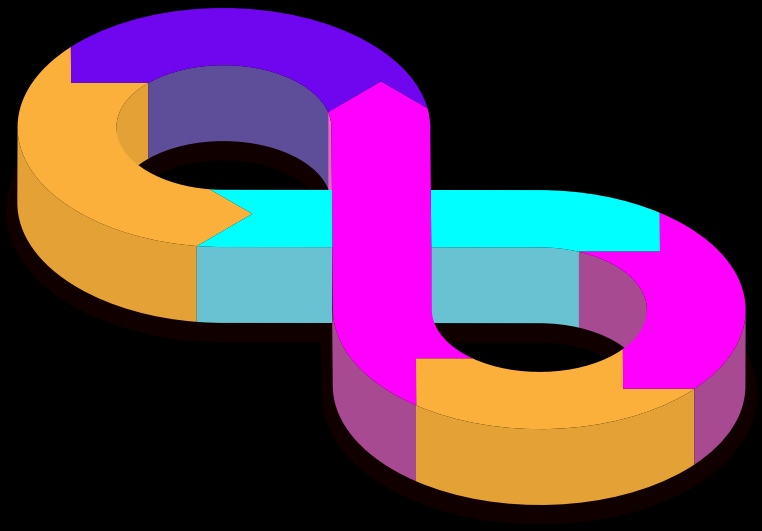
BlazeMeter
by Perforce



The 11 Principles of Sustainable Continuous Testing



As release cadences continue to accelerate, more organizations are looking to shift their testing left and embrace continuous testing. By testing at each stage of the Software Development Lifecycle (SDLC), DevOps teams can meet the need for speed without sacrificing quality.

Yet there are additional issues to consider aside from testing the application functionality:

- **56% of critical dependencies** are unavailable.
- **50% of time** is spent looking for test data.
- **64% of defects** occur in the requirements phase.

To truly adopt continuous testing, teams must develop a continuous testing program that looks at the whole picture and take these other considerations into account.

This eBook will delve into the 11 principles of sustainable continuous testing. These principles will focus both on testing application functionality and application performance & security.

Contents

Background: The Challenges to Continuous Testing

Continuous testing is the practice of testing across every activity in the SDLC to uncover and fix unexpected behaviors. Yet continuous testing can also happen before development even starts.

Let us take a closer look at two areas where continuous testing makes a large impact: throughout the testing pyramid and in the requirements phase of the lifecycle.

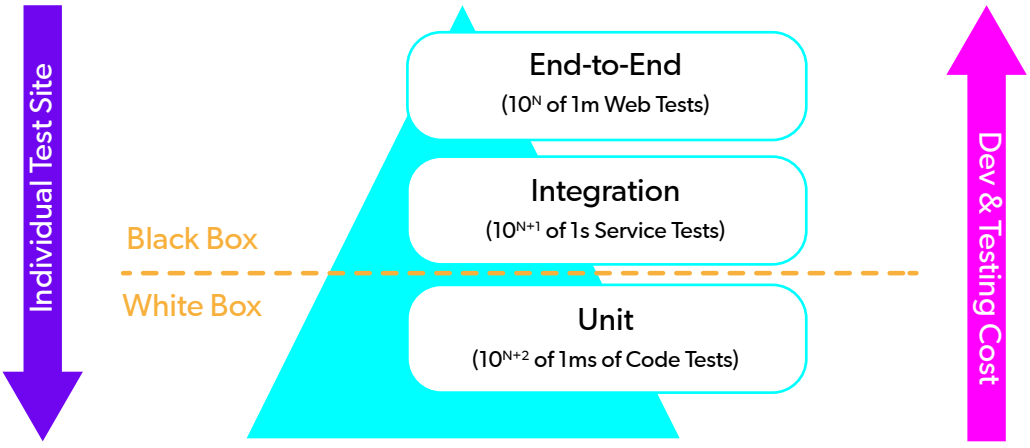
THE TESTING PYRAMID

One of the largest challenges to achieving continuous testing is creating the right testing strategy. To ensure quality at speed, testers need to cover scenarios across the entire test pyramid and in the right proportions.

API and unit testing are at the base of the test pyramid. These tests are smaller and shorter, and typically created and run by the developer. Teams need proper environments and fast feedback to find and fix problems within unit and API testing without interrupting the workflow.

Additionally, developers depend on the availability of environments that are stable, reliable, and always on. Therefore, dev teams require mock services and test data on demand to seamlessly progress their software development and testing.

As the SDLC progresses, teams start building more integration tests which consist of functional and non-functional testing types. When the release gets closer to production, the developers and test engineers perform end-to-end tests which include the highest scale testing types across most relevant platform combinations.



Software Testing Pyramid

Having the ability to drive the complete continuous testing plan from unit through API, integration, and E2E, is imperative to the success and velocity of the agile teams.

THE TESTING PREQUEL: THE REQUIREMENTS PHASE

Yet it is not just the testing itself that poses a challenge. Currently, 64% of defects originate in the requirements phase of the SDLC. Ineffective communication at this stage can introduce defects into the lifecycle before a single line of code is even written.

Very often requirements are specified at a very high level that is good enough for developers and testers, but no one else. Without having the domain subject matter expertise to fill in the gaps in the requirements phase, team members are at risk of interpreting the same instruction differently.

On the flip side, requirements can be specified in an overly detailed fashion but take too long to decipher and maintain over time. Not to mention developers and testers find it hard to grasp so much information bundled together in a giant requirement.

The requirements definition process must improve to be unambiguous, complete, and testable.

11 Principles of Continuous Testing

To successfully accelerate and improve a code release process through continuous testing, teams must adhere to certain principles to ensure that quality keeps pace. To embed quality in applications, teams need to add more modern testing practices in the form of small quality checks performed throughout the application pipeline. This would enable teams to test small sections of code constantly.

Achieving continuous testing does not just mean focusing on test automation. Even after your API tests and GUI tests are automated and integrated into a continuous integration agent, those tests have dependencies that need consideration. Teams must satisfy the dependencies on test data, interfaces, and environments before they can be executed. You must:

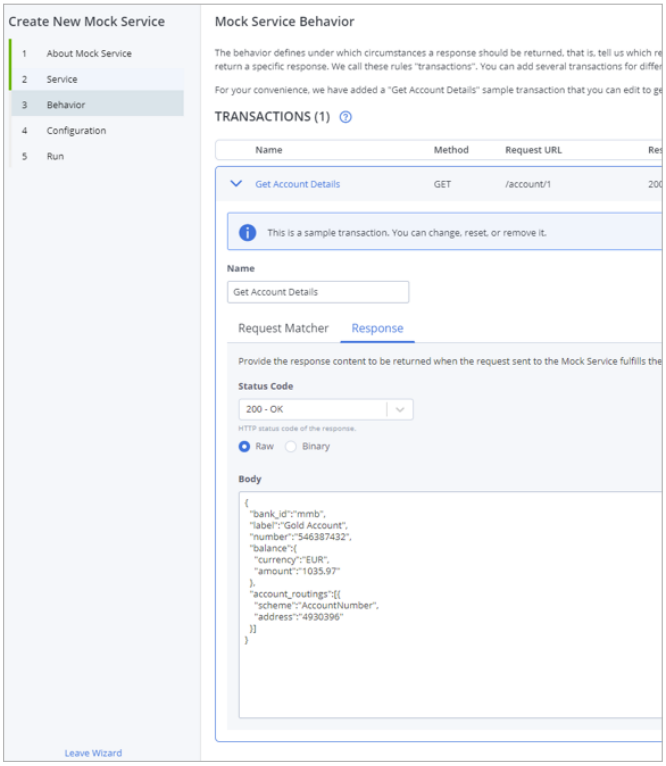
- Remove environmental constraints.
- Virtualize any interfaces you do not own or control, or do not want to test.
- Utilize ephemeral testing environments.
- Automate test data provisioning and management to feed your tests on-demand.
- Automate your tests so they can run against your virtualized interfaces using the appropriate test data.
- Have your pipeline orchestration engine set up in such a way that there is no need for manual intervention during the above steps.
- Focus on complementary disciplines.

Let us explore these 11 principles of continuous testing in more depth.

VIRTUALIZED ENVIRONMENTS

Continuous testing means both testing more frequently and hitting multiple environments more frequently. If these environments are not constantly available for testing, teams will encounter bottlenecks.

Some environments are accessible through APIs, while teams can access others through various messaging interfaces. Those environments could be built with modern architectures, while others are monolithic legacy client/server or mainframe systems.

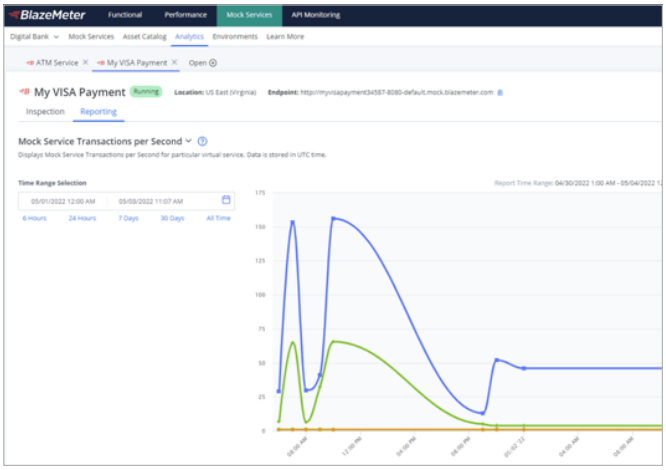


Testing using mock services

So how do you coordinate testing through multiple environments when they are either not always available or too expensive to test against? Virtualizing those environments allows you to test your code without having to worry about other systems and environments that you are not changing.

Making those environments ephemeral, available on-demand through service virtualization, removes that constraint from your development lifecycle. You control these environments; you can spin them up as needed and they will always be available.

Unlike basic mocks, true virtualized services or "mock services" can provide sophisticated testing scenarios by returning unexpected or negative responses, which allow chaos testing by emulating real-world conditions. With virtualized services, your application is truly go-live tested.

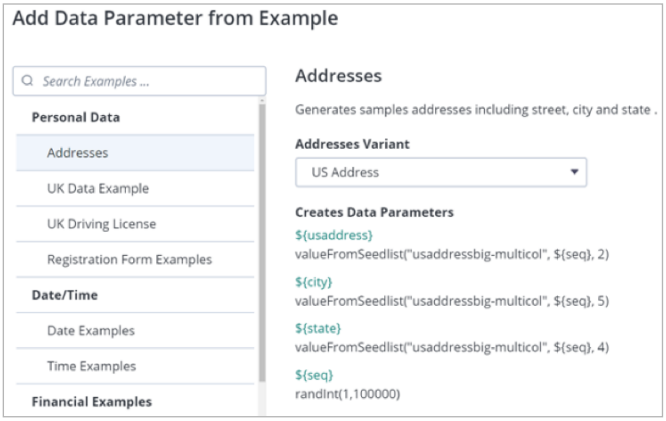


Sample mock services testing report using BlazeMeter.

TEST DATA MANAGEMENT

Finding the appropriate test data is an ongoing struggle for testers. If testing is the biggest bottleneck in the SDLC, then test data is arguably the biggest bottleneck in testing! You must have the right data and the appropriate diversity of data for positive and negative testing scenarios. You won't find all that diversity in production. And production data comes with inherent compliance concerns.

Increasingly, teams are turning to synthetic data generation to streamline their continuous testing. Synthetic data enables teams to be confident that no personally identifiable information is at risk in the data they are testing.



BlazeMeter Test Data Generation

Plus, gathering data from production, and condition or provision it properly before sending it to your tests is a cumbersome process. Only with synthetic data can you keep up with the speed your application pipeline requires.

TEST AUTOMATION

In a fully orchestrated application pipeline, today's scripts can still fail due to factors not related to the application code. So, no one will trust the results.

While it may seem obvious, it is important to note that you need reliable test automation to achieve continuous testing. Organizations that achieve continuous testing release on a daily or bi-weekly basis and run tests that cover a variety of platforms, test cases, and user conditions.

With such robust test coverage, organizations who automate their tests minimize defects while the delivery pipeline focuses more on innovation. They spend no more than 20% of their time focusing on any backlog.

Related Reading:

Perforce Test Coverage Guide >>

When organizations can automate 90% or more of their tests, they can rest assured that their tests are running reliably in the background. This reassurance gives the space they need to focus their efforts elsewhere, such as manual testing in the areas that need it or coming up with different ways to test new application features.

PIPELINE ORCHESTRATION

Successful pipeline orchestration is critical to achieving continuous testing. As part of any DevOps adoption initiative, it is unrealistic to expect to get to continuous testing without the reliability and speed of a standardized and automated pipeline.

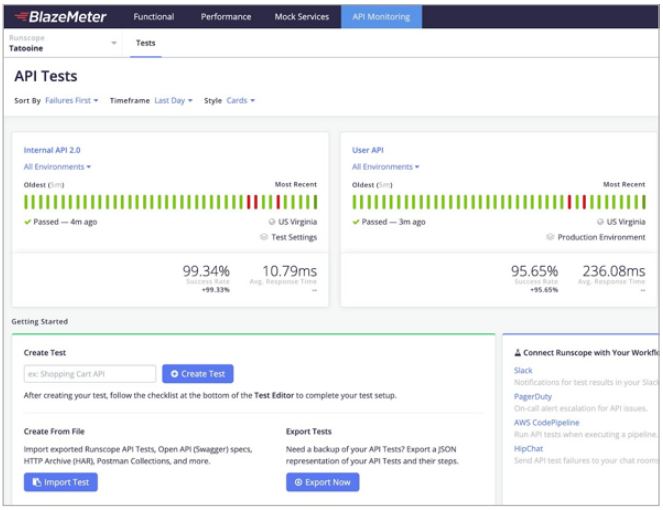
A pipeline orchestrator is an automated workflow tool that will run all your automated tests and fully integrate with code deployment activities as it moves through the pipeline. Understanding your pipeline in more depth enables your team to integrate continuous testing attributes within each stage.

Your pipeline orchestration must be integrated with your automation suite, and your team must understand how it works, how to interpret results, and how to make it scalable. Everyone involved in testing should also have full visibility into what is being run through the pipeline with full transparency.

API TESTING

API testing enables teams to properly align their testing efforts with the testing pyramid. To achieve continuous testing, teams must strengthen their unit and API testing while reducing their reliance on UI testing, especially for business logic testing.

By testing at the unit and API levels as much as possible, teams can find defects at earlier stages of development and reduce the cost of fixing these defects in the future.



API Testing with BlazeMeter

PERFORMANCE AND LOAD TESTING

Even when applications are functionally running properly, you must radically change the way you think about performance testing to truly test continuously.

As you shift testing to the left, you are testing earlier in the lifecycle with less application and infrastructure components. While these tests are smaller by nature, they are also much greater in volume.

Make performance and load testing available to everyone across all Agile teams. Developers and testers alike must be able to create a

performance test and run it by themselves, without having to send requests to anyone else.

SECURITY TESTING

Security testing must play a central role when strategizing a transition to continuous testing. While often treated as an end-of-the-line concern, security deserves to be placed in the continuous testing stream at the very start.

Developers often unknowingly create security vulnerabilities. Continuously testing security is not just about policing those vulnerabilities; it is also about helping developers better understand the security of their code to prevent them from creation those vulnerabilities in the future. Security must become a partner, a trusted advisor, and expert, giving developers the training and tooling that can be measured downstream.

Currently, security and testing are sitting on the wrong side of the wall from the developers. Security should not be seen as a conflicting with quality. Rather, your security, testing, and development teams should work in partnership with each other.

ACCEPTANCE OF TEST-DRIVEN DEVELOPMENT AND BEHAVIOR-DRIVEN DEVELOPMENT

Test-driven development (TDD) and behavior-driven development are great complements to continuous testing.

By creating tests to ensure that the various acceptance criteria for each user story are met, teams can build more focused tests within the sprints and developers can focus more on meeting business objectives. Over time, teams will define much more detailed acceptance criteria, which will require more comprehensive testing.

AUTOMATED TEST GENERATION

The activity of manually designing and writing manual tests or automated test scripts is a major bottleneck.

Teams should automatically generate the new acceptance-level tests for manual and automated testing in the beginning of the sprint. Then, when developers start checking in and merging the first working pieces of their code, teams can run automated tests immediately. This automation allows developers to get faster feedback on the quality of their code as they check it into source control.

REQUIREMENTS ENGINEERING

You must include all SDLC stakeholders in the continuous testing journey. That means finding a better way to communicate and collaborate on the requirements. The earlier you include requirements engineering into your continuous testing initiative, the smoother the continuous testing journey will be in the long run.

FEEDBACK LOOPS

Feedback loops are critical to successful continuous testing. Dashboards in real time. Must be automatic and entire team must have access. Feedback loops across the entire SDLC, not just production, should be used as a compass to help you navigate your continuous testing transformation.

How to Measure the Impact of Continuous Testing

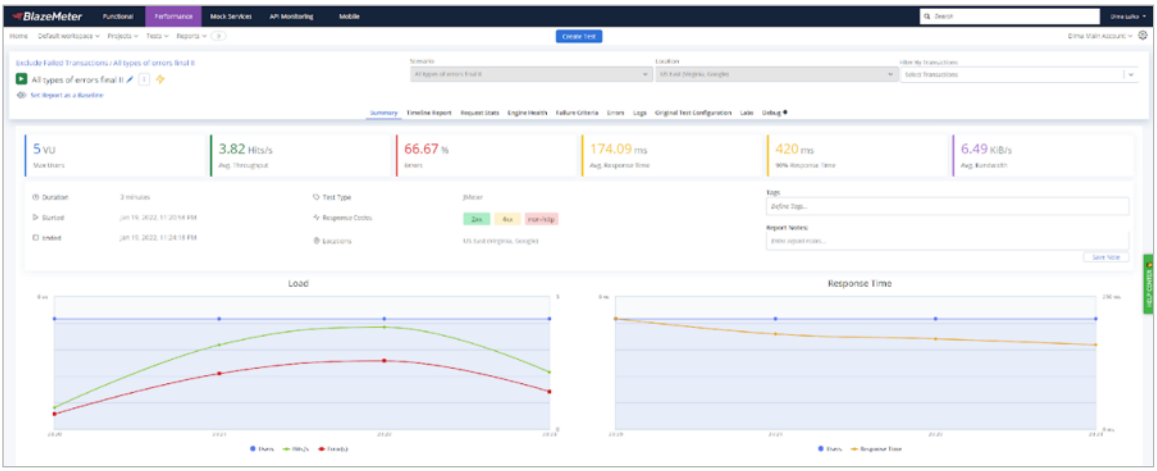
Every project requires metrics to determine progress and quality, and to identify business value. This is no different for continuous testing.

Some of the overarching benchmarks of determining progress and quality include customer adoption, engagement, and revenue. Teams should seek to understand and quantify the sentiment of end users and add them to a continuous feedback loop back to developers and the business.

You must also be able to track how many defects you have in pre-production and production, and pair this knowledge against the speed at which you can deploy. As people write increasingly better code, teams should see the number of defects drop against a constant release cadence. Or better yet, against an increasing release cadence.

In assessing and measuring quality versus quantity, factor in the following:

- Time to market.
- Value stream analysis.
- Timelines and costs of manual test execution.
- Regression testing.



Sample Performance Testing Report

Start shifting left to developers by democratizing testing tools and evolving testers into engineers. Every leader could and should have their own view of what constitutes the most valuable continuous testing metrics. Ideal metrics to track include:

- Lead times.
- Releases.
- Production incidents.
- Business value.

A continuous testing plan must plan for future aspects of development in an era of unabated change and speed. From the increased role of AI to the Internet of Things, teams must use predictive analytics to ensure their applications are ready for the new technologies that are on the horizon.

Summary

Achieving continuous testing is a journey. But the principles of continuous testing discussed in this eBook will help you make the journey a successful one.

As teams become more comfortable with these continuous testing principles, your entire organization will begin to benefit from them but in the short and long term.

RELATED RESOURCES

- Continuous Testing Meets Contract Testing—How to Shift Left in the Cloud-Native Era
- BlazeMeter University: Become A Continuous Testing Ninja
- What Can Perfecto & BlazeMeter Do For You?

ABOUT BLAZEMETER

The BlazeMeter Continuous Testing Platform is a complete solution for shift-left continuous testing. The platform includes UI functional testing, user experience testing, API testing and monitoring, performance testing, test data, and virtual services.

All capabilities are deeply integrated in an intuitive workflow designed for agile teams and provide robust support for popular open source tools. Delivered in software as a service with support for multiple clouds or private cloud, it is simple to get started, intuitive to use, and a powerful tool for delivering innovation with quality and speed.

Visit www.blazemeter.com for a free account, to request a demo, or learn more.