Business white paper

# 10 Best Practices for Application Performance Testing

Leveraging Agile Performance Testing for Web and Mobile Applications

# 10 Best Practices for Application Performance Testing

**Table of contents**

## Introduction: What is Performance Testing?

Software application performance testing is a somewhat vague and subjective phrase that many people find difficult to define. For instance, what exactly is good performance? How do you determine if something is fast, and what makes an application slow? The problem is that these are subjective terms that vary between applications, users, and devices. If your goal is to build a fast web application, or you're dealing with users complaining that your mobile app is slow, testing for this may prove challenging.

The fact is that proper performance testing will help determine if a system meets certain acceptable criteria for both responsiveness and robustness while under reasonable load. Although responsiveness varies—it could be the reaction time to user input or the amount of latency between server request/response cycles—it's typically something that can be measured directly. Robustness also varies by system, but it typically translates into a measurement of stability, scalability, and overall system reliability.

Instead of dealing with the subjective, a successful approach to performance testing involves precise plans and well-thought-out goals. Start by defining test plans that include load testing, stress testing, endurance testing, availability testing, configuration testing, and isolation testing. Couple these plans with precise metrics in terms of goals, acceptable measurements, thresholds, and a plan to deal with performance issues for the best results. This should include measurements such as average response time over predefined timeframes, graphs of standard deviation, absolute timings, average latency, and request/response outliers, among others. Let's take a look at a more complete performance-testing blueprint for success.

## Ten Performance Testing Best Practices

Orasi and HP collaborated to develop these top ten best practices for application performance testing. They serve as guidelines for performance testing traditional software—i.e. web applications—but they apply to others, including mobile apps. For specific mobile performance testing best practices, see the sidebar "Additional Considerations for Mobile Apps." Let's look at the top performance testing best practices now.

**Take a DevOps approach to application performance testing, where developers, IT operations, and testers work as a team to build, deploy, tune and configure applicable systems, and execute performance tests.**

### 1. Test Early and Often

Performance testing is often an afterthought, performed in haste late in the development cycle, or only in response to user complaints. Instead, take an agile approach using iterative-based testing throughout the entire development lifecycle. The agile methodology has become popular due to the advantages of frequent iteration, the involvement of all stakeholders, and the continuous course of corrections it provides. The advantages compound when you take a continuous testing approach, involving more than development, which takes us to the second and related performance testing best practice.

### 2. Take a DevOps Approach

Soon after the lean movement inspired agile, IT organizations saw the need to unify development and IT operations activities. The result is the DevOps approach, where developers and IT work together to define, build, and deploy software as a team. Just as agile organizations often embrace a continuous, test-driven development process, DevOps should include developers, IT operations, and testers working together to build, deploy, tune and configure applicable systems, and execute performance tests against the end-product as a team.

### 3. Consider Users, Not Just Servers

Performance tests often focus on the results of servers and clusters running software. Don't forget that real people use software, and that performance tests should measure the human element as well. For instance, measuring the performance of clustered servers may return satisfactory results, but users on a single overloaded or troubled server may experience an unsatisfactory result. Instead, tests should include the per-user experience of performance, and user interface timings should be captured systematically in conjunction with server metrics.

To illustrate, if only one percent of one million request/response cycles are latent, ten thousand people—an alarming number—will have experienced poor performance with your application. Driving your performance testing from the single user point of view helps you understand what each user of your system will experience before it's an issue.

### 4. Understand Performance Test Definitions

It's crucial to have a common definition for the types of performance tests that should be executed against your applications, such as:

- **Single User Tests.** Testing with one active user yields the best possible performance, and response times can be used for baseline measurements.
- **Load Tests.** Understand the behavior of the system under average load, including the expected number of concurrent users performing a specific number of transactions within an average hour. Perform common tests with added system load, additional concurrent users, and so on. Measure system capacity, and know the true maximum load of the system while it still meets performance goals.
- **Peak Load Tests.** Understand system behavior under the heaviest usage anticipated for concurrent number of users and transaction rates.
- **Endurance (Soak) Tests.** Determine the longevity of components, and if the system can sustain average to peak load over a predefined duration. Memory utilization should be monitored to detect potential leaks. The testing will also ensure that throughput and response times after sustained activity continue to meet performance goals.
- **Stress Tests.** Understand the upper limits of capacity within the system by purposely pushing it to its breaking point. This helps determine the system's robustness in terms of extreme load and helps identify both potential scalability issues and breaking points.
- **High Availability Tests.** Validate how the system behaves during a failure condition while under load. There are many operational use cases that should be included, such as:
  - Seamless failover of network equipment, hardware, application servers, and databases
  - Rolling server restarts
  - Loss of a server within a cluster

- Seamless addition and removal of nodes
- Seamless patch or upgrade of software and firmware where applicable
- System functionality and performance during a server failover—before it happens in production
- The impact of network bandwidth variation, such as during a communications link failure

• **Configuration/Tuning Tests.** Assess the effect of changing system parameters or software versions. A reasonable goal is to maintain 80 percent to 85 percent optimal performance during, for example, an application server upgrade.

• **Isolation Tests.** Measure the performance of typical silo components, such as service-oriented architecture (SOA) systems, user interfaces, and legacy systems. Include virtualization-based testing for web services, REST services, message-oriented middleware solutions, and others, where the actual systems need not be included in the tests.

### 5. Build a Complete Performance Model

Measuring your application's performance includes understanding your system's capacity. This includes planning what the steady state will be in terms of concurrent users, simultaneous requests, average user sessions, and server utilization during peak periods of the day. Additionally, you should define performance goals, such as maximum response times, system scalability, user satisfaction marks, acceptable performance metrics, and maximum capacity for all of these metrics.

It's crucial to define related thresholds that will alert you to potential performance issues as you pass those thresholds. Multiple thresholds need to be defined with increasing levels of risk. The associated levels of alerts can be used not only to define a response to a potentially escalating performance problem, but also help you work with DevOps to further tune the environment and refine your production system monitoring. True to the agile approach, capacity planning and measurement creates a feedback loop that improves both your application's performance *and* your monitoring and measurement of that performance.

An effective planning processing includes the definition of success criteria, such as:

• **Key Performance Indicators (KPI)**, including average latency, request/response times, and server utilization

• **Business Process Completion Rate**, including transactions per second, and system throughput load profiles for average, peak, and spike tests

• **Hardware metrics**, including CPU usage, memory usage, and network traffic

### Ten Additional Considerations for Mobile Applications

The following is a set of best practices specific to performance testing mobile applications:

**1. Consider network quality.** Latency tends to be higher on mobile networks, connection quality is more unpredictable, and connections come and go.

**2. Consider geography.** Users in certain regions of the country will experience different response times and affect your back-end infrastructure differently.

**3. Consider entire mobile product families.** Performance and available resources differ even within product families (i.e. Apple iOS). For instance, with iOS, iPhone 4, iPhone 4S, iPhone 5, and iPhone 5S are all in use, and Android is even more fragmented.

**4. Don't focus on specific devices.** Define device-agnostic tests.

**5. Expect more concurrent users.** People tend to use their mobile devices more often, and for short bursts of activity. Instant mobile access has the capability to create a load singularity that would not exist in normal desktop applications/sites. Special test cases should be developed to test these areas.

*continued*

## Ten Additional Considerations for Mobile Applications *continued*

**6. High user expectations.** Mobile users expect to get responses immediately, and if not they might move on to another application or site.

**7. Use emulation in performance tests.** It may not be feasible to wait for mobile software to be installed on actual devices for testing, and it may not be feasible to test the demands of multiple actual devices. Consider how device emulators fit into your manual and automated testing.

**8. Always test end-to-end.** A mobile app is only as good as its back-end server response time plus its own processing time.

**9. Isolate timings.** The performance break down is more complex. Measure the following:

a. Content request time
b. Content delivery time
c. Render time
d. User gesture response
e. General UI responsiveness

**10. Capacity testing.** Test low-memory and out-of-storage conditions.

### 6. Include Performance Testing in Development Unit Tests

Waiting until a system is mostly built or complete to run performance tests can make it harder to isolate where problems exist. It's often more costly to correct performance issues later in the development process, and more risky to make changes if functional testing has already been completed.

As a result, developers should include performance testing as part of their unit tests, in addition to dedicated performance testing. There's a significant difference in the testing approaches, as unit testing often focuses on sections of code, not just application functionality or the integrated system. This means developers will be intimately involved with the performance of their code throughout the development process, and everyone will have a leg up on knowing how to monitor individual components for issues in production.

### 7. Define Baselines for Important System Functions

In most cases, QA systems don't match production systems. Having baseline performance measurements of each system can give you reasonable goals for each environment used for testing. They specifically provide an important starting point for response time goals where there are no previous metrics, without having to guess or base them on other applications. Baseline performance tests and measurements, such as single-user login time, the request/response time for individual screens and so on, should occur with no system load.

### 8. Perform Modular and System Performance Tests

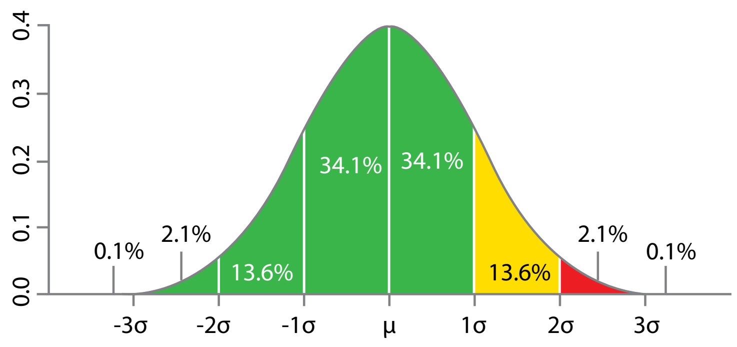
Modern applications are composed of many individual complex systems, including databases, application servers, web services, legacy systems, and so on. All of these systems need to be performance tested individually and together. This helps expose the weak links, learn which systems adversely affect others, and understand which systems to isolate for further performance tuning.

### 9. Measure Averages, but Include Outliers

When testing performance, you need to know average response time, but this measurement can be misleading by itself. For example, if you strictly measure the average response time for your application at less than five seconds, and this is within target, your related performance tests will pass. However, some of those responses may have been at or well beyond five seconds, leading to a poor experience for those affected users. Be sure to include other metrics such as 90th percentile or standard deviation to get a better view of system performance.

KPIs can be measured by looking at the average and standard deviations. For example, set a performance goal at the average response time plus one standard deviation beyond it (see Figure 1). In many systems, this improved measurement affects the pass/fail criteria of the test, matching the actual user experience more accurately. Transactions with a high standard deviation can be tuned to reduce system response time variability and improve overall user experience.

**Figure 1. Measuring standard deviation shows averages and outliers.**



**10. Consistently Report and Analyze the Results**
Performance test design and execution are important, but test reports are important as well. Reports communicate the results of your application's behavior to everyone within your organization, and can even serve as bragging rights for project owners and developers. Analyzing and reporting results consistently also helps to define attack plans for fixes.

Remember to consider your audience, since reports for developers should differ from reports sent to project owners, managers, corporate executives, and even customers if applicable. With each report, note obvious software changes made (enhancements, bug fixes, and so on) as well as any other changes tested (third-party software upgrades, changes to environment, hardware, and so on).

## Five Common Mistakes to Avoid

In addition to following a set of best practices for performance testing, there are common mistakes to avoid. These include:

- Not allowing enough time. Build time into the schedule up front.
- Throwing software "over the wall." Keep developers involved by including performance in unit testing efforts.
- Using a QA system that differs greatly from production in terms of size, configuration, and/or underlying hardware.
- Insufficient software tuning.
- Failing to define a troubleshooting plan. This includes having known responses to performance issues, and production run books for error or failover situations.

## Conclusion

The best practices for performance testing include proper definition and planning, scheduling time upfront, remaining agile in your approach, testing early and often, getting as much of the organization involved as possible, testing all components together and separately end-to-end, considering failure scenarios, measuring thoroughly, and reporting results properly. Additionally, remember to avoid throwing software "over the wall" to a separate testing organization, and ensure that your QA platforms match production as closely as possible.

As with any profession, your efforts are only as good as the tools you use. Be sure to include a mix of manual and automated testing across all systems. Orasi and HP offer performance testing tools and services to help you get started and continually refine your testing process as you move forward.

**For more information, contact Orasi today**
**www.orasi.com**
**678-819-5300**

## About Orasi

Orasi is a leading provider of software, support, training, and consulting services using market-leading test management, test automation, enterprise testing, environment hosting, and mobile testing technology. For over 12 years, Orasi has consistently helped customers successfully implement and integrate software testing environments to reduce the cost and risk of software failures. Orasi offers proven solutions for the installation, implementation, configuration, and use of performance testing technologies and best practices.

An HP Software Platinum Partner and authorized Support partner, Orasi resells HP's test management and automation solutions, is a leading provider of software testing services, and offers mobile testing, application security, and cloud-based testing and monitoring solutions. Orasi was HP Software's US Solution Partner of the Year in 2011 and 2013, and was Support Partner of the Year for 2009, 2011, and 2012.

**Sign up for updates**
**www.orasi.com/news/Pages/Events.aspx**

Share with colleagues