Analyzing, Interpreting, and Reporting Load Test Results

A Neotys Whitepaper

Table of Contents

Why Analysis of Test Results Matters

In today's App Economy, the speed at which businesses secure revenue directly correlates to application quality and customer perception, expectation, and delight. Does the business application deliver the anticipated user experience and desired value? Or, is it riddled with slow response times, timeouts, broken links, redirects, and other issues that degrade the corporate brand, erode customer loyalty, and irritate users so much that they flee to the competition?

The objective of performance testing is to identify and expose the most likely sources of application risk and deliver the actionable insight required to mitigate the code and performance bottlenecks that lead to application downtime and outages. When done correctly, load tests reveal system errors and limitations so that they can be remediated quickly. The accurate interpretation of performance test data is an incredibly critical role. Analysis results inform the tactical and strategic decisions that drive the business. A premature decision to go live with a risky website or application can have profound, and unwelcome, ramifications that impact revenue, brand, market perception, and the user experience.

When the application either doesn't meet user expectations or deliver the value users expect, customers will take their business elsewhere, which is why the analysis of performance test data is so essential. Results analysis is a predictive tool that describes the expected behavior of the application under production load stressors and provides the foundation for smart decision-making. The mandate here is that data needs to become actionable information—information that will drive strategic decisions and influence business outcomes.

Because business success can depend on accurate load test analysis and conclusions, testers will benefit from a logical framework to approach load test analysis. Neotys framework components include these steps:

- Define Criteria for Success and Failure
- Establish Objectives
- Execute Realistic Tests
- Analyze Test Results
- Report Based on Stakeholder Preference

A Logical Framework for Approaching Load Test Analysis

Load testing results analysis can be a thankless and stressful job. It not only requires a comprehensive knowledge of load testing design, an understanding of the technical layers involved in the application, and familiarity with the application's architecture, but it also demands the ability to make accurate conclusions about the data and communicate them to stakeholders—all within a short time window. Skill and experience enable testers to ask the right questions, conduct the proper tests, make the

correct conclusions, and win the trust of stakeholders who own business decisions based on those conclusions.

Although most testers agree that breaking an application under heavy load is pretty straightforward, finding the root problem based on automatically generated load testing reports is surprisingly challenging. For this reason, it's best to follow a rigorous paradigm of a performance test.

Define Criteria for Success and Failure

Defining criteria for success and failure is a prerequisite to any test strategy. Before testing an application, establish acceptable thresholds for robustness and performance. In most cases, these criteria are defined regarding average and maximum response times per page, maximum error rate, or the number of requests per second.

As testers proceed through this process, they should remain mindful of the user experience. The people that will ultimately use the application may reside in different locations (geography) which can impact bandwidth, data transfer efficacy (packets dropped), and latency. Because user behavior describes how users interact with an application, understanding these behaviors and the paths users take to navigate through workflows is essential. Further, user preference concerning device types will be varied. As a result, devices connecting to the application will possess different hardware, firmware, and operating systems. Realistic performance test design needs to take all these factors into account.

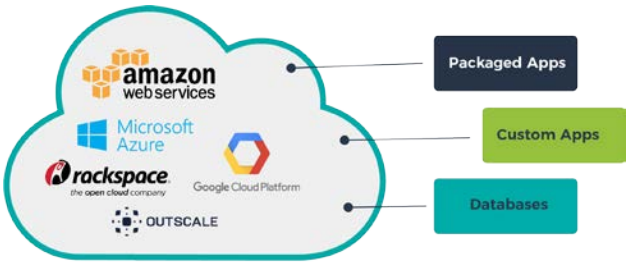


Figure 1 – Always keep good User Experience as the central success criteria, regardless of type of app or cloud provider used

Establish Test Objectives

Performance tests can evaluate application robustness and performance, hardware, and bandwidth capacities. Well-defined requirements set precise test objectives that assess application stability. When the simulated load remains constant over an extended period, load tests reveal whether the application supports the anticipated number of simultaneous users and the desired response time for critical pages. The server is considered overloaded if these usage figures regularly exceed 90%.

Stress tests can validate hardware configurations or the number of simultaneous users that the application can handle while maintaining acceptable response times. They also provide insight into server behavior under high load (i.e., does it crash?) and the load threshold above which the server begins to generate errors and refuse connections.

Lastly, performance tests validate performance variations after an application or infrastructure update and assess whether implemented upgrades resulted in real performance gains and which (if any) pages experienced performance degradation.

Execute Realistic Tests

How the word "realistic" gets defined depends on with whom you speak and where they work. Testing strategies vary by organization, depending on the application under test (AUT), resource and tool

availability, and desired business goals. The reasons for this are many. The business needs to define its quality benchmarks and designate what pillars of quality are most critical to the brand. Further, team dynamics influence the flexibility and speed of the testing process. For example, agile environments demand that load tests occur at the beginning of the development process and that the application proceeds through a process of continuous testing. Other development environments may not stipulate this requirement.

Testers need to understand how software applications should respond to real-world scenarios; this insight provides the basis for successful performance test design and helps teams prioritize what areas of the application pose the most risk. As part of this process, testers must consider the types of devices, environments, anticipated load, and data types supported within the application ecosystem. They then need to align this understanding with their preproduction environments to assess what scenarios can be tested in preproduction versus production.

Certain facets of the load test require particular attention to obtain useful results. For example, simulating users requires more than merely playing back requests to the server. Testers also need to:

- **Define the number of Virtual Users:** Ensure that the number of Virtual Users approximates the number of real users once the application is in production, with a realistic think time (Think Time) applied between pages.

- **Define several types of Virtual Users:** Not all users use a web application in the same way. Define a Virtual User for each user profile: simple browsing, browsing with modifications, system administrator, etc.

- **Use different user accounts and values:** Use variables to dynamically modify critical values such as user account logins or other form parameters (such as **productID** in an e-business application). The main idea of this is to bypass the use of the various server caches.

- **Test load balancers:** Load balancers can use the client IP address to balance the load over several servers. If all Virtual Users are using the same IP address (default setting), then the total load will be sent to a single server.

- **Simulate actual user bandwidth:** A user connecting at 100 Mbps through a local network and a user connecting with a dial-up modem at 56kbps do not load the server and bandwidth in the same way.

Performance Testing Best Practice Tips

We recommend that testers apply some useful and proven tips to manage their time most efficiently during test execution.

Prioritize user and test scenarios to test critical functionality first. By applying appropriate risk assessment techniques, testers can prioritize their tests and determine where to direct the most intense test efforts and what areas warrant lighter testing, to conserve resources for more intense scenarios. Such risk-based testing helps identify significant problems more quickly and earlier in the process by focusing test on the riskiest application aspects.

For many systems, performance and robustness problems result from:

- Resource-intensive features
- Timing-critical/sensitive uses
- Probable bottlenecks (based on internal architecture/implementation)
- Customer/user impacts including visibility
- Previous defect history (observations noted across other similar systems during live operation)
- New/modified features and functionality
- Heavily used features
- Feature set complexity
- Exceptions
- Troublesome system components that are poorly built or badly maintained
- Platform maintenance

Before recording traffic, delete cookies and clear the browser cache. If testers neglect to clear the browser cache while recording user scenarios, then they may obtain skewed results; the web browser may rely on cached data to process client requests rather than sending data to the server.

Begin recording new scenarios from the web browser launch. If testers begin recording a scenario after connecting to the web server under test, then any web pages they may have already opened will be excluded from scenario playback. This would pose issues if authentication steps were executed before the start of the scenario recording.

Parameterize scenarios to simulate more realistic server load. By replacing recorded parameters with variable values in test requests, testers can augment scenarios with dynamic behaviors that mimic groups of unique real users and send user-specific data to the server. We recommend that before testers parameterize scenarios, they dig into the types of request and response data that are transferred to and from the server.

Verify user scenarios. When you run a test simulating many virtual users, it may be difficult to find problems that are not caused by the simulated load. That is why we recommend that you verify a scenario using one virtual user. After you make sure the scenario works fine for one virtual user, you can check how it works under real-life conditions.

Analyze Performance Test Results

Result analysis begins with the testing tool. The primary goal of any performance testing tool is to provide a clear status on application performance and to help testers derive insight from the data. To simplify this task, results, and reports should be easy to customize, intuitive, and focus on three central themes:

- Response times
- Availability
- Scalability

Ultimately, the reports that the tool generates need to demonstrate whether the performance requirements identified during the performance strategy phase are validated. Test summaries should be clear and offer graphs and statistical tables that make the job of performance test result analysis easier.

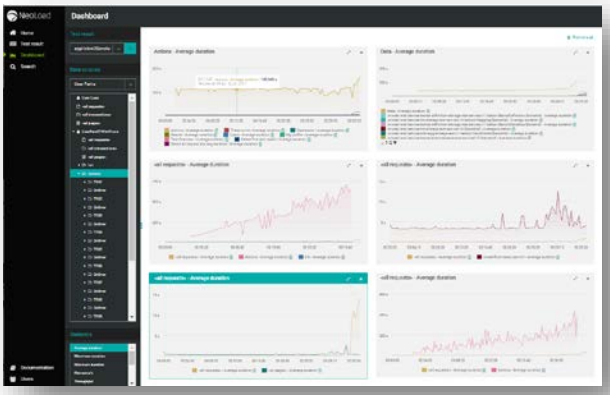


Figure 2 - NeoLoad Web offers customizable shared dashboards for visualizing test results

Measuring Web Application Performance Levels

As testers assess application performance, they will want to consider a variety of performance metrics:

- Expected response time (time required to send a request and receive a response)
- Average latency time
- Average load time
- Anticipated error rates
- Peak activity (users) at specified points in time
- Peak number of requests processed per second
- CPU and memory utilization requirements to process each request

To check the application's proper behavior under load, testers might want to begin by first reviewing the response times for all pages and making sure they meet performance requirements. It's also helpful to validate that the number of errors is acceptable at the target loading. From there, testers can move on to assess other metrics that relate to their desired quality thresholds.

Comparing Test Results

NeoLoad's graphical interface includes a Compare mode which makes it possible to generate comparison reports that illustrate the results of two tests. NeoLoad can generate comparison reports with the same structure and in the same formats (Word, HTML, PDF, and XML) as standard reports. The more alike the designs of the Virtual Users in both tests, the more relevant the results comparison is. Neotys advises to compare two test results that use the same User Paths, but different scenarios (different load policies, different durations, etc.). Color coding assists with the interpretation of comparisons.

Understanding Test Results Context

To identify potential performance bottlenecks, testers should have access to granular statistics for different application pages. Then, through comparison, testers can analyze results from different runs of the same (or different) scenario(s).

Typical steps involved in a testing phase include:

- Running a particular scenario
- Analyzing results and identifying "slow" pages
- Changing and improving pages or the code called by those pages
- Re-running the scenario
- Comparing results before and after improvements

Producing Informative Results

Some handy tips to make results more informative include:

- Validate critical pages: By checking that the server response is valid under load, testers not only ensure that the scenario works as expected, but also that the load imposed does not cause errors in the application. Once the test is complete, select the requests containing errors, or those where the validation failed. The content of the corresponding server response may then be analyzed to determine the cause of the problem. Note that validation uses up system resources, such as CPU and memory, on the Load Generator. It may be best to restrict its use to testing key pages only (for example, those that provide access to a database, or those more likely to produce errors).

- Begin with low volumes: Begin with a short test and a light load. Correct any application errors occurring under the light load before carrying out tests using heavier loads.

- Stop Virtual Users that contain errors: When a Virtual User receives an error, it should typically stop running. If this does not happen, it could continue playing requests that have no meaning. For example, if the user login fails, there is little point sending further browsing or search requests to the application as it will only distort the response time statistics for those pages.

- Use Containers: Each transaction (registration, online purchase, etc.) may be composed of several web pages. Statistics may be obtained by transaction by grouping these pages in a Container.

Load Tests Reveal Trends

Trends are essential indicators of the progress (or regression) of daily performance testing of newly-released versions of application components. Spotting trends in performance regression more quickly and pinpointing which changes introduced the regression leads to easier and less expensive resolution. Trends also give testers an immediate, clear idea of the general quality trend of overall performance. Users can graph trends in critical statistics covering several tests to quickly identify performance regression.

Interpreting Test Results

Performance test result interpretation requires context. Context begins by reviewing the success and failure criteria set by the organization when it established acceptable benchmark thresholds for robustness and performance. Those criteria derive from key performance targets related to service level and capacity.

Service Level Targets:

Availability or "up-time": Refers to the amount of time an application is accessible to the end-user.

Response Time: How long it takes for the application to respond to user requests, typically measured as system response time.

Throughput: Measures the rate of application events (i.e., the number of web page views

within a specified period.)

Capacity Targets:

Utilization: Capacity of an application resource. This has many parameters that relate to the network and servers, such as network bandwidth, system memory, etc.

Performance engineers have a responsibility to report reliable information about the systems they test within the context of the software quality thresholds and benchmarks set by their employer. Deep QA skill and experience are necessary for successful test result interpretation. Testers are always at the peril of asking the wrong questions and concluding information that is not predictive of the production load the system will face.

A key strategy to overcome the peril of asking the wrong questions is hypothesis testing. Based on test result data, testers should formulate hypotheses, make tentative conclusions, determine what information is required to confirm or disprove each theory and package their findings supplemented with pertinent visualizations. Test reporting should provide system performance insights, explanations of bottlenecks, and a report narrative that offers actionable direction on how to overcome the performance challenges identified during testing.

Confirming that application performance deteriorates under certain loads is of some value. Understanding the reason for the performance degradation, however, and identifying the limiting resource that causes it, is actionable information. This data and the prescriptive steps that QA plans to take to remediate the underlying issues is what stakeholders need to see in test reporting.

Report Based on Stakeholder Preference

Almost all load testing solutions allow for the creation of complex and attractive graphs that correlate data. The Performance Engineer's first inclination may be to include all available graphs in their reporting. However, before creating the report, it's important to understand the role and the technical skills of the person reading it.

In NeoLoad, there is a selection of report types that testers can create. Technical reports show key data and graphs for developers and operations. Executive reports provide concise application performance status and graphical presentations (e.g., pie charts) that make results easier to understand.

Customize Reports for the Intended Audience—Remember that Data Needs to Become Information

Because different stakeholders need specific performance test analysis information, each role may prefer different presentation and data representation methods. Using a performance test tool that offers a variety of data representation makes sharing information more accessible. Testers need only augment visuals with brief commentary and share this information cross-functionally.

Based on respective stakeholder roles and interests, expectations around reporting vary. QA and development stakeholders care about the technical implications of test result conclusions, such as overall code quality and service level thresholds. Other stakeholders may focus on the business implications of test analyses, such as the impact on revenue attainment and customer retention. While this can make sharing performance test results challenging, understanding the information requirements of different stakeholders and what reporting they value makes providing appropriate, on-demand information easier.

All Stakeholders

All stakeholders have an interest in performance-related questions such as:

- Is performance improving or not?
- Are we meeting service level agreements (SLAs) thresholds and internal requirements?
- What reports are available? Are they customizable so I can request more/less detail?
- How frequently can I view reports?

However, the level of detail and the manner and frequency of information presentation varies significantly by role.

Executive Stakeholders

Executive stakeholders have specific reporting needs and expectations that differ from other team members because they are responsible for meeting revenue targets and justifying why they are (or are not) achieved. Test analyses provide clarity and insight into the technical performance of the organization in delivering value to the customer. Due to time constraints, senior management prefers to consume information concisely that highlights key points and success/risk factors. Executives expect visual representations of data to be intuitive and digestible at a glance and performance test result summaries to address these questions:

- Are we on schedule?
- How will performance and load test results impact production?
- How accurate are these results? What are the risks?
- What tasks remain?

Technical and QA Management

Project managers, development leads, and QA managers require the same insights as executive stakeholders, yet they need more frequent and detailed information about test coverage and whether performance tests identify and resolve issues efficiently. Their questions focus on:

- Are performance tests identifying and addressing problems efficiently?
- Is the current performance test coverage sufficient? Do we need to apply more test?

- Are there currently any show-stoppers? If so, how do we mitigate them?

Technical Team Members

Technical team members want information relating to test results, monitored data, and general observations that are actionable, relevant, and tailored to answer only their questions. Since much of their interest focuses on opportunities for analysis and improvement, they want information that addresses:

- How do results impact my area of focus?
- How can I access the most recent test results?
- Where can I get the raw data?
- Can you capture additional metrics during the next test run?

Collaboration Delivers Success

Everyone wins when the application meets or exceeds business and user expectations. When testers have access to powerful testing tools, they can easily inform stakeholders and provide the actionable insights that fuel discussion, collaboration, and decision making.

Using NeoLoad tools, testers and stakeholders can improve project collaboration and:

- Create customizable, personalized, and shareable graphical dashboards that allow users and stakeholders to mix several elements and KPIs in the same graph (e.g., average transaction times, average request response time, and total transaction failed for each element)
- Share test analysis for running or terminated tests with all stakeholders: developers, QA managers, business stakeholders and product owners
- View test results during runtime in Continuous Integration (CI) testing environments
- Extract test data and metrics through NeoLoad Web's API. Analyze and correlate test data with other parameters from 3rd-party data providers, and build your custom reports from 3rd-party data analysis tools

Conclusion

The ubiquity of apps presents a significant revenue channel for most companies. Because consumers demand responsive and seamless user experiences, they enforce a high bar for application quality. This ubiquity makes the job of the performance engineer that much more strategic to the business. When applications fail to meet user expectations, companies lose revenue and customers.

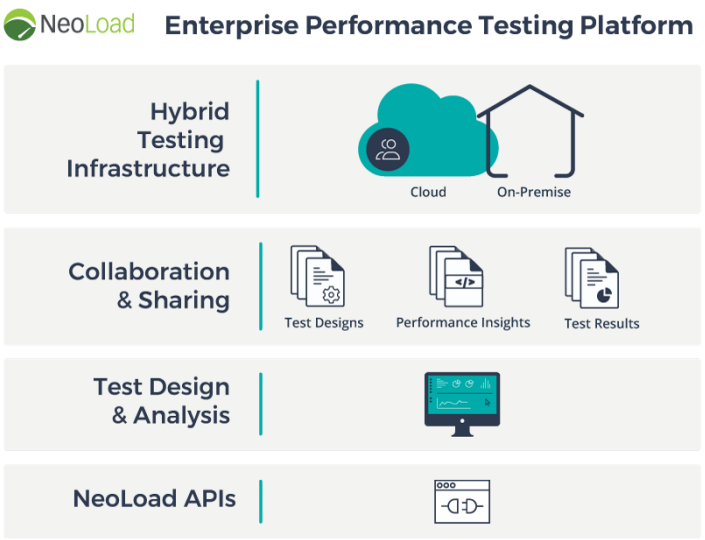
Performance test analysis provides insight into the availability, scalability, and response time of the application under test. This is why interpreting test results within the context of quality thresholds established by the business are such a critical task. Result interpretation is a crucial skill that yields actionable information about the overall quality of the application, its anticipated behavior under load, and required additional testing.

Timely and accurate reporting and sharing of these insights with extended team members and stakeholders support vital technical and business decisions. By creating compelling reports that respond to stakeholder concerns and interests, testers can arm team members with actionable information and

promote the collaboration that drives business decisions about application release and deployment risk factors.

About Neotys

The success of your digital strategy relies on your ability to deliver fast and reliable software, regularly. Creating great software quickly, using an optimized performance testing process is your competitive advantage – Agile and DevOps are part of the solution.



Neotys has nearly 15 years of development investment into NeoLoad – the performance testing platform designed to accelerate Agile and DevOps processes. It's built by engineers who recognized that to achieve their Agile adoption objective; they needed to create a product that could facilitate superior load and performance testing continuously.

The result – up to 10x faster test creation and maintenance with NeoLoad.

We genuinely believe that the Performance Engineer can become the critical application performance partner providing the best testing coverage while respecting the cadence of the Continuous Delivery process. As performance becomes the responsibility of the wider team, continued delivery of an optimized performance testing platform is what drives our work every day.

For more information about Neotys and NeoLoad visit: www.neotys.com or contact sales@neotys.com