



RadView Software
White Paper

Test Metrics – Which Are Most Valuable?

It's no secret that today's web applications (websites, intranets, extranets) are critical to a company's reputation and business success. Visitors or users are immediately impacted by how well or how poorly an application runs. Inadequate performance or lack of functionality can adversely affect a company's reputation, and customer satisfaction. To ensure online business interactions are a success, thorough testing of web applications is essential.

What exactly needs to be tested and which test metrics will provide QA and test engineers with the most value? Today's test engineers are under more pressure than ever to ensure websites run as expected. And in many cases, they lack the time to complete comprehensive testing cycles. This white paper will discuss how to define a test plan, how to choose the necessary tests for an application, and how to determine what metrics are critical and what metrics are simply nice-to-have. It will cover analyzing the web application performance, traffic monitoring metrics, and end-user experience. It will also highlight the importance of communication and collaboration between QA and development and how that facilitates producing tests that simulate real-life application behavior.

Typically when testing web applications, test and QA folks start with functional testing - testing the pages, the links and all the buttons on the application. This is usually followed with load and stress testing to ensure the application can handle a certain volume of users. Then, depending on how critical the application is to the company's business, additional tests are run prior to launching the application into the production environment.

A well thought out test plan is necessary to coordinate the testing and ensure that the team is testing the right things and getting the most from its work. Otherwise QA and test will end up wasting precious time and effort. Building a test plan need not be a lengthy process, but it does need to reflect the application's purpose.

One of the goals of a good test plan is to include tests that mimic how the application is expected to perform. To do this effectively the QA and testing team needs information from the application's sponsor and the development team. QA and test need to meet as early in the development process as possible. At this initial meeting ask as many questions as possible to learn about the application's functionality and environment: For example:

- What's the application expected to do?
- What's a typical expected user experience? (positive and negative)
- What's the application architecture?
- What's the expected traffic volume in terms of unique, concurrent users?
- Will the traffic be dependent on time of day?
- What type of data will be downloaded from the site and from where? (text only, images, a web service?)
- How will the application's functions influence performance? (i.e., the viewing of images, interactive web service functions, etc.)
- What's the profile of the expected user? Company employee with high-speed access or a home user with an older OS, browser and dial-up connection?
- Will the application be hosted on a company intranet or accessed via the Internet?
- What and who in the test/QA and development group will handle the security issues and testing?

- What is the current development status of the application and the new builds schedule? This directly affects the test plan schedule.
- What's the application's release schedule?
- When will the development team train the QA and test team on the application's specifics? This is key to understanding the application and leads to better troubleshooting, faster problem solving and improves communication with the development team.

After this initial meeting, develop a description of the typical user(s). This is the first step to building true test case scenarios. Share this information with the application sponsor and the development team to check your suppositions. This will ensure that you are testing the 'right' things. At this step, good project management is needed.

Project management can be kept simple by developing an easy to access web-based communication forum. With this tool, a project management team and an internal user group can access all the team's documents; add comments and update schedules. With this process in place, you are ready to build your test plan. Here's a simple example.

Your company's HR department has sponsored a web-based online benefits enrollment application. Development has been building the application and the QA department expects to get it a month before the program starts for pre-deployment testing. It will be used by the company's entire workforce (10,000 employees) and will be accessed from work computers (state of the art, fast connection, updated browsers) and home computers (older models, browsers and dial-up connections.)

You set up a meeting with HR and development and ask them the basic questions to begin your plan. What you learn is that they expect half the folks to log on from work, with the other half logging on from home. You learn that the application traffic will be very heavy for the month of February (open enrollment time for benefits) and then drop to very little activity (mostly research from individuals and HR) for the rest of the year. You learn that a typical, positive user experience will have the user log on, browse through some static material, use a calculator function, and fill in forms and log off. You discover a negative user experience would be to log on and find the site unavailable. Or log on, get half way through filling out forms and be timed out or knocked off due to the volume of users or other glitch. Another bad experience would be if the application didn't save the benefit information form or if the application failed to 'send' the form to HR for approval. Then someone would think they had enrolled, while their paperwork was never processed.

There are three end users to this site: the company employee, the HR department and the benefits administrator at a third party company. All the links and processes between these groups must be tested. You need to identify the positive and negative experiences for each of these groups and prepare tests that will emulate each case.

Finally, the application will pull information in real-time from two of the benefits provider's sites. This material will be mostly text and requires another set of tests to ensure that the link is working and the data being requested and sent is correct, arriving at an acceptable speed and available to your company's employees during the heavily trafficked months. Ask the benefits provider to prove their load capacity. If the benefit provider gives your company access to their system, you can be sure

they are supplying it to other customers and you want to ensure your employees have the same timely access as their other clients.

From this meeting you derive at least three customer scenarios, plus a list of features to be tested. These may include testing:

- all hardware and software connections between users
- all buttons, pages and links
- the scalability (up to 10K users) of all customer scenarios
- that users are not dropped or timed out while filling out forms
- the speed of the link between HR and the 3rd party for submitting forms under stress
- the save and send functions under stress
- that the information link from the 3rd party benefits provider to the company site works under stress
- that completed forms are sent to HR for approval and incomplete forms sent back to the employee to be finished

Share what you plan to test with the development team and the sponsor of the application. Ask for feedback. There may be things your team has overlooked and things that are unnecessary to test.

After consolidating the feedback you are ready to line up the appropriate tests for these functions. The next step is to understand what the metrics from your testing software can tell you. Then you will be in a position to merge your customer information with metrics to create an efficient test plan.

Understanding Metrics

During a test session, virtual clients generate result data (metrics) as they run scenarios against an application. These metrics determine the application's performance, and provide specific information on system errors and individual functions. Understanding these different metrics will enable you to match them to the application function and build a more streamlined test plan. (The names may differ, but the following metrics are provided by most of the popular testing software.)

I. Measuring Scalability and Performance

Depending on your application, scalability and performance testing could be a priority. For example, if the website is open to the public during a sporting event, i.e. the Olympics, the application will need to scale effortlessly as millions of people log on. Conversely if you are testing an in-house benefits enrollment application, the number of users is predictable as is the timeframe for the application's use and ensuring availability is a simpler test process.

So, how do you measure the scalability or performance of an application? How do you compare two sessions and know which session created more stress on the application? Which statistic should you use to compare two load machines testing the same application? To start with, it's important to understand what you are testing when you are looking at site usage.

Hits Per Second

A Hit is a request of any kind made from the virtual client to the application being tested. The higher the Hits Per Second, the more requests the application is handling per second.

A virtual client can request an HTML page, image, file, etc. Testing the application for Hits Per Second will tell you if there is a possible scalability issue with the application. For example, if the stress on an application increases but the Hits Per Second does not, there may be a scalability problem in the application.

One issue with this metric is that Hits Per Second relates to all requests equally. Thus a request for a small image and complex HTML generated on the fly will both be considered as hits. It is possible that out of a hundred hits on the application, the application server actually answered only one and all the rest were either cached on the web server or other caching mechanism.

So, it is very important when looking at this metric to consider what and how the application is intended to work. Will your users be looking for the same piece of information over and over again (a static benefit form) or will the same number of users be engaging the application in a variety of tasks – such as pulling up images, purchasing items, bringing in data from another site? To create the proper test, it is important to understand this metric in the context of the application. If you're testing an application function that requires the site to 'work,' as opposed to present static data, use the pages per second measurement.

Pages Per Second

Pages Per Second measures the number of pages requested from the application per second. The higher the Page Per Second the more work the application is doing per second. Measuring an explicit request in the script or a frame in a frameset provides a metric on how the application responds to actual work requests. Thus if a script contains a Navigate command to a URL, this request is considered a page. If the HTML that returns includes frames they will also be considered pages, but any other elements retrieved such as images or JS Files, will be considered hits, not pages. This measurement is key to the end-user's experience of application performance.

There is a correlation between the Page Per Second and the stress inflicted on an application. If the stress increases, but the Page Per Second count doesn't, there may be a scalability issue. For example, if you begin with 75 virtual users requesting 25 different pages concurrently and then scale the users to 150, the Page Per Second count should increase. If it doesn't, some of the virtual users aren't getting their pages. This could be caused by a number of issues and one likely suspect is throughput.

Throughput

This is an important baseline metric and is often used to check that the application and its server connection is working. Throughput measures the average number of bytes per second transmitted from the application being tested to the virtual clients running the test agenda during a specific reporting interval. This metric is the response data size (sum) divided by the number of seconds in the reporting interval.

Generally, the more stress on an application, the more Throughput. There should be a strong, predictable correlation between these two. If the stress increases, but the Throughput does not, there may be a scalability issue or an application issue.

Another note about Throughput as a measurement – it generally doesn't provide any information about the content of the data being retrieved. Thus it can be misleading especially in regression testing. When building regression tests, leave time in the testing plan for comparing returned data quality.

Rounds

Another useful scalability and performance metric is the testing of Rounds. Rounds tells you the total number of times the test agenda was executed versus the total number of times the virtual clients attempted to execute the Agenda. The more times the agenda is executed, the more work is done by the test and the application. There is a correlation between Rounds and the stress executed on an application. The test scenario the agenda represents influences the rounds measurement.

This metric can provide all kinds of useful information from the benchmarking of an application to the end-user availability of a more complex application. It is not recommended for regression testing because each test agenda may have a different scenario and/or length of scenario.

II. Application Responses and Availability

Now that we have the scalability and performance issues outlined, which metrics should be used to measure the user's experience?

Hit Time

Hit time is the average time in seconds it took to successfully retrieve an element of any kind (image, HTML, etc). The time of a hit is the sum of the Connect Time, Send Time, Response Time and Process Time. It represents the responsiveness or performance of the application to the end user. The more stressed the application, the longer it should take to retrieve an average element. But, like Hits Per Second, caching technologies can influence this metric. Getting the most from this metric requires knowledge of how the application will respond to the end user.

This is also an excellent metric for application monitoring after deployment. Using baseline data, a test insert 'probe' can alert test or QA when the application slows to a certain response time.

Time to First Byte

This measurement is important because end users often consider a site malfunctioning if it does not respond fast enough. Time to First Byte measures the number of seconds it takes a request to return its first byte of data to the test software's Load Generator. For example, Time to First Byte represents the time it took after the user pushes the "enter" button in the browser until the user starts receiving results. Generally, more concurrent user connections will slow the response time of a request. But there are also other possible causes for a slowed response. For example, there could be issues with the hardware, system software or memory issues as well as problems with database structures or slow-responding components within the application.

Page Time

Page Time calculates the average time in seconds it takes to successfully retrieve a page with all of its content. This statistic is similar to Hit Time but relates only to pages. In most cases this is a better statistic to work with because it deals with the

true dynamics of the application. Since not all hits can be cached, this data is more helpful in terms of tracking a user's experience (positive or frustrated). It's important to note that in many test software application tools you can turn caching on or off depending on your application needs.

Generally, the more stress on the site the slower its response. But since stress is a combination of the number of concurrent users and their activity, greater stress may or may not impact the user experience. It all depends upon the application's functions and users. A site with 150 concurrent users looking up benefit information will differ from a news site during a national emergency. As always, metrics must be examined within context.

III. Integrity and Failures

Failed Rounds/Failed Rounds Per Second

During a load test it's important to know that the application requests perform as expected. The Failed Rounds and Failed Rounds Per Second tests the number of rounds that fail.

This metric is an "indicator metric" that provides QA and test with clues to the application performance and failure status. If you start to see Failed Rounds or Failed Rounds Per Second, then you would typically look into the logs to see what types of failures correspond to this metric report. Also, with some software test packages, you can set what the definition of a failed round in an application. Sometimes, basic image or page missing errors (HTTP 404 error codes) could be set to fail a round, which would stop the execution of the test agenda at that point and start at the top of the agenda again, thus not completing that particular round.

Failed Hits/Failed Hits Per Second

This test offers insight into the application's integrity during the load test. An example of a request that might fail during execution is a broken link or a missing image from the server. The number of errors should grow with the load size. If there are no errors with a low load, the number of errors with a high load should remain zero. If the percentage of errors only increases during high loads, the application may have a scalability issue.

Failed Connections

This test is simply the number of connections that were refused by the application during the test. This test leads to other tests. A failed connection could mean the server was too busy to handle all the requests, so it started refusing them. It could be a memory issue. It could also mean that the user sent bogus or malformed data to which the server couldn't respond so it refused the connection.

Conclusion

When a testing team pairs exactly what can be learned from each test to the user experience, testing time can be spent wisely. Working with the application's sponsor and the development team early in this process is key, as it will impact the types of metrics needed to examine and ensure performance. Also, this provides the testing team with the time needed to draw up an efficient test plan, gather any additional hardware and software tools needed, and schedule the testing. While this isn't always possible, it should be a team goal, as it will absolutely ensure better results – through comprehensive and thorough testing that is performed early and often.

RadView Software, Inc.

7 New England Executive Park
Burlington, MA 01803
Phone: 781-238-1111
Fax: 781-238-8875
Toll Free: 1-888-RADVIEW

RadView Software, Ltd.

14 Hamelacha Street
Park Afek
Rosh Haayin, 48091 Israel
Phone: +972-3-9157060
Fax: +972-3-9157683