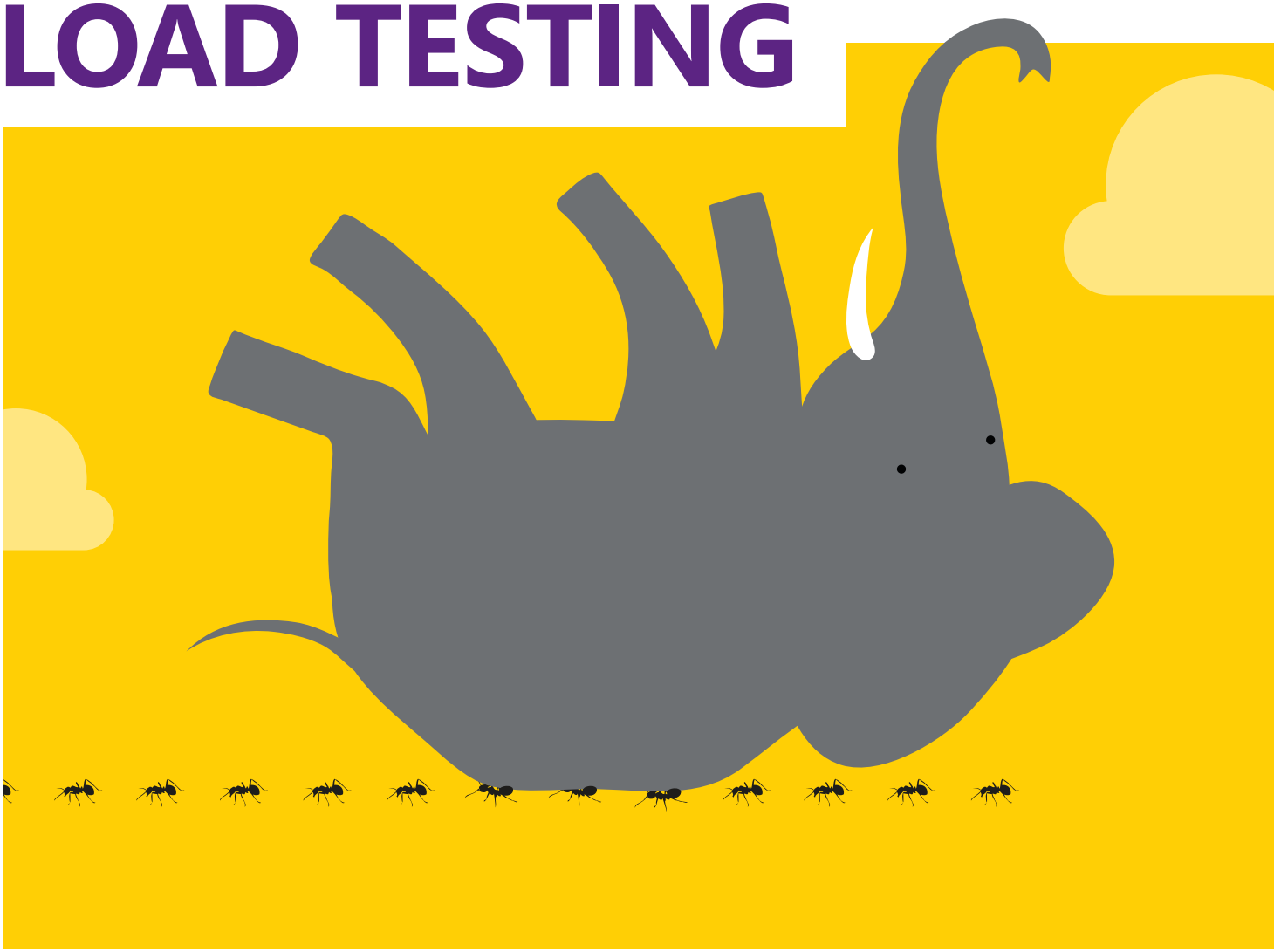
Taking the First Steps in

WEB LOAD TESTING



Telerik

TEST STUDIO

An Introduction

Software load testing is generally understood to consist of exercising an application with multiple users to determine its behavior characteristics. As testers add users, they measure such characteristics as application response time for different pages and various measures of system resource utilization and performance.

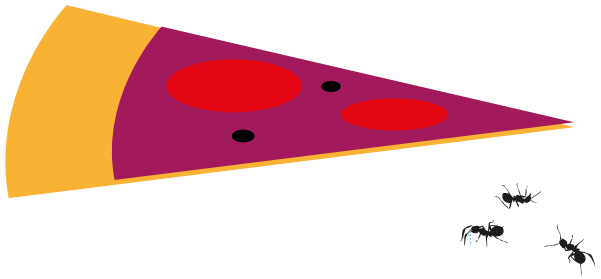
Load testing may also consist of increasing the number of users until the application breaks or otherwise becomes unusable. This provides testers and other stakeholders with an indication of the upper limit of users possible, as well as potential areas of weaknesses in individual components.

The importance of load testing, especially for **customer-facing applications**, can't be overstated. Rarely a week passes without a story of websites becoming nonresponsive or crashing under an unexpectedly high number of users. Organizations regularly lose millions of sales or lost business opportunities, or suffer embarrassment or bad publicity due to poorly scaling websites.

However, load testing is also important for many **internal applications**. Without good design and implementation practices, critical applications used by salespeople, customer support professionals, and others may slow down or fail when in heavy use.

Other business questions can be just as important. Organizations may be planning for a new marketing campaign, sales promotion, or product launch, and must have confidence that their customer-facing applications won't fold under heavier or different traffic. Load testing is often done in preparation for upcoming changes in existing application use. Some of the most publicized and costly failures have occurred to organizations doing high-profile and expensive advertising campaigns, such as during the Super Bowl.

In the past, load testing used to be a manual effort. Colloquially, it was often called a pizza test, as it was typically conducted by groups of staff members working at their computers during non-business hours, who would be offered pizza in return for volunteering their time.



The pizza test was a poor substitute for more comprehensive automated load testing. The manual testers worked from a set task script or scripts, and variation in how they executed those scripts and the timing of the individual steps often produced inconsistent or incorrect results. It was also very time-consuming and usually didn't provide testers with detailed information about specific application behaviors.

Today, most organizations prefer load testing by using simulated users running automated scripts. This approach provides greater consistency in the methodology, so that results are more repeatable. It is also much faster and more reliable in producing good data describing the application and system characteristics.

The response of many organizations to poorly scaling web applications is to use more servers, simply because servers are inexpensive and easy to add. However, depending on the characteristics of the application, additional servers often don't address the underlying problem. Good upfront load testing

can save money throughout the application lifecycle in fewer servers and less electricity. It can also provide an indication of the integrity of the design and implementation of the application.

But most important, it can give an organization the ability to deploy a mission-critical application with the confidence that its behavior is well understood and that it will stand up over the expected user load. That is more than worth the relatively small amount of time and effort needed to do effective load testing.

Before Load Testing Starts

Load testing starts with **requirements**. In many cases business owners, business analysts, or other stakeholders specify how many users it has to support. That may be based on a known number of employees or registered users, or it may represent an estimate of the expected number of simultaneous customers or users for the application.

Teams may also have **goals** for load on the application that may go above and beyond stated requirements. The goals may be derived from requirements, but can also be based on knowledge or experience of the stakeholders in the application architecture or problem domain. Senior testers or developers may believe that a certain load should be achievable, and will look for ways to achieve that goal.

In at least some cases, the stated or implied requirements represent a guess as to the number of simultaneous users that the application must support. This is especially true for customer-facing web applications, which can grow its user base quickly if the company or application is successful in its market. In other cases, the number of possible simultaneous users is completely unknown, because the organization or stakeholders have no reasonable basis for an estimate.

That's why it's important to not just load test with a specific number of users, but to **test with different numbers of users performing different tasks**. Even if the application meets its initial load requirements, it's essential that testers go beyond simply saying that the requirements were met. If the application use grows over its lifecycle, the data collected during test is important to determine whether or not it needs further development.

In addition to looking at the application over multiple load characteristics, testers also want to look at its **maximum possible number of simultaneous users**. This data provides an estimated upper bound of use, and will also show which application components or system resource is fully committed first.

The next thing testers must consider is **how to profile the application under load**. This consists of determining what type of data should be collected to both demonstrate whether or not the application meets requirements, and to determine the performance and reliability characteristics of the application under an increasing load. While response time is the most important metric here, there are a variety of other options that provide testers with valuable additional information on the robustness of the application.

In all cases, testers have to **look for real or potential bottlenecks as the number of simulated users increases**. The obvious system bottlenecks are CPU, disk traffic, and network traffic, and collecting this data should be at the top of any load testing list.

But testers need more information in order to characterize the impact of increasing users on an application. In a distributed application, testers will want to look at **utilization and network traffic across specific machines**, and also examine how specific parts of the application process data and respond to requests.

Memory use is also an important characteristic in application load, in that using more memory than necessary can prevent an application from effectively responding to more users. An increasing memory profile as users are added and removed could also be an indication of a memory leak, which can happen with either unmanaged or managed code.

Testers will also want to look at **individual application components** and their interactions with other components to determine if there are bottlenecks or inappropriate wait states that will cause the entire application to slow significantly or fail as the number of users increase. Other metrics, such as database calls or cache misses, may make sense depending on the application.

Development teams need this level of data to help diagnose and correct defects associated with an inability to meet requirements, or if there are undesirable characteristics in the application that should be corrected at some point.

Setting Up the Load Tests

Once testers have determined what application and system characteristics they need to record and measure, they must consider **what tasks the simulated users perform**, and from what browsers and browser versions. To start this process, they must work with the user community, business analysts, and product owner to determine common use cases for the application.

In many cases, these use cases already exist in automated functional tests, or can be created by combining multiple tests. If the load testing tool can accept and execute existing functional tests, that can save a great deal of time and technical effort over creating load tests from scratch. Either the tests themselves or their generated network traffic can drive the load testing tool. Functional tests will have to be assembled according to the use cases, and additional tests may need to be recorded to help tie them together to build use case scripts.

The question of browsers and browser versions can be answered externally by examining browser use across the broader Internet, or internally by looking at traffic to current customer-facing websites. In the case of internal applications, organizations can dictate which browser or browsers can be used.

If multiple browser use is required, additional planning is necessary to make sure the load tests replicate real use as closely as possible. This is typically done in a simulated fashion from the load testing tool, rather than physically using different browsers and browse versions. Load tests don't require actual client computers or browsers; instead, the load testing tool also simulates the selected browser mix.

Ideally, testers want to replicate the actual user environment and use cases as accurately as possible. Doing so provides a level playing field to not only determine if requirements are being met, but also to gain important information about the behavior of the application under load.

Load testing is an experimental process, and should be planned and executed methodically. Testers should **change only a single parameter for each load test**, whether it is the type or mix of scripts, server configuration, test dataset, or other factor. This will enable testers and developers to more effectively identify weak spots and diagnose poor load conditions.

Interpreting the Load Testing Results

Analyzing and interpreting load testing results can benefit from a good knowledge of distributed application architecture, programming language characteristics, system resource utilization, and the interactions between them. It also helps to have experience in understanding how an application interacts with its host computer systems with multiple users.

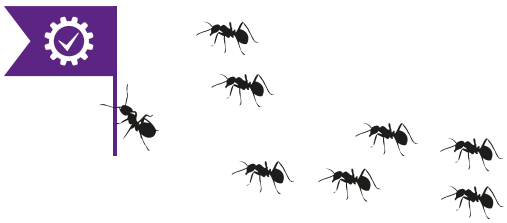
But even without a high level of load testing skills and experience, good testers can **ask the right questions** in order to determine if the application meets requirements, and whether it needs more investigation before deployment.

1. **Does the application meet requirements for simultaneous users? How many users does it take before the application become unresponsive or fails altogether?** These are the most immediate questions testers are trying to answer, and go a long way toward determining if the application is ready to be deployed.
2. **What does the response curve look like as the number of simulated users increases?** Sometimes requirements also specify application response time parameters; if not, there is usually a common sense level of response time that should be tested. Testers can determine if the application responses are prompt enough for real use, and if not investigate further.
3. **What do the curves for memory, disk, and network utilization look like as the number of simultaneous users increases?** These curves can provide information on potential application defects that may not be detectable through other means, and also help testers determine any limitations that should be addressed prior to deployment.
4. **What other measured characteristics look unusual or out of balance?** Depending on what characteristics are being measured, testers may notice spikes, lost transactions or other potential problems that should be investigated further.

Summary

Thanks to advances in the features and usability of load testing tools, testers can replicate real life use more closely than ever to determine whether a web application meets user capacity and response time requirements, system resource utilization, and other application characteristics. Because they don't have to write code to define use cases and scripts, multiple tests using different configurations can be run in less time than it used to take to run a single test. Understanding how the application is likely to perform under real life conditions, and identify potential problems under user load, can go a long ways toward improving application quality.

Ultimately, load testing is about making sure business goals are met, and unpleasant surprises minimized or eliminated entirely. By identifying and addressing problems prior to deployment, organizations can have confidence that their applications will effectively serve the business.



Learn about load testing with Telerik Test Studio

Peter Varhol

Telerik Test Studio Evangelist

@pvarhol on Twitter.

Telerik
TEST STUDIO