# 7 Load Testing Mistakes

YOU DON'T WANT TO MAKE

RADVIEW

# Introduction

Load testing is one of the last pre-production links in the development chain. This makes it tempting to cut corners, rush results, and just get testing underway so you can move forward to launch.

However, there's no argument that load testing is a highly-complex undertaking. Plan incorrectly, build scenarios that don't simulate a real production environment accurately, or overload your load generators – and at best you've lost the time, money and resources involved in re-running the test. At worst, you haven't realistically load tested your system – and your long-anticipated launch can turn into a disaster.

Leveraging thousands of hours of load testing experience in organizations across the globe, we put together the following list of seven common load testing mistakes. They could happen to anyone, but they won't happen to you once you've read this eBook.

Mistake #1

# Not Defining Clear Goals

Without clearly-defined, quantifiable testing goals, load testing is a guessing game, at best. Load testing goals need to be clearly defined based on business requirements prior to running and measuring your test scenarios. Here are some common testing goals you might want to consider:

- **Capacity Per Scenario**
  How many users should the system handle per specific scenario?

- **Throughput**
  What is the volume of data you expect your system to handle? For example, if users need to download 1Gb files, how much data should the system handle under peak load?

- **Response Time Per Activity**
  What is an acceptable response time for each specific transaction? What's the maximum response time, averages, and acceptable level of outliers?

- **Acceptable Percentage of Errors**
  For example, is a login error rate of 0.1% acceptable for a banking application?

- **Goals per Type of User, Location, Browser, and more**
  Do you have different types of users (gold, VIP) for whom you need to provide a different level of service? Should a specific user location be defined with different goals?

- **Standart vs. Peak time Goals**
  Have you defined different response time goals depending on time of day or peak user activity?

- **Uptime Duration**
  Most production systems don't run for only 2-4 hours like a load test. What is your system's uptime and how do you translate it to a runtime load testing goal?

Consider which of these goals are relevant for your system and environment. Only by defining specific load testing goals like these will you be able to clearly evaluate and report the results of your performance testing.

Mistake **#2**

# Not Creating a Realistic Test Environment

A real-life production environment has nearly endless components – servers, databases, hardware, 3rd party tools, integrations, background processes that run periodically and much more. Because of this, a key load testing challenge is simply building a test environment that simulates the actual production environment.
Without investing time and thought in creating a realistic environment, you can waste massive efforts testing something that is not real. Make sure your testing environment is:

- **Similar to Production**
  Your test environment should mimic as closely as possible the real production environment in terms of hardware, configuration, memory, databases, load balancers and so on. You can start small with testing using a reduced environment, but keep in mind that you must simulate a wide-scale production system.

- **Isolated**
  Your load testing environment should be completely isolated from other activities. If additional users store data on your testing system or run processes you're unaware of, your load testing results will be skewed.

- **Populated with Data**
  Load testing a database system with 100 records is not the same as a real production system with 20 million records. You need to invest the time to generate a quantity of data that resembles the real production system.

- **Integrated with 3rd party apps**
  Testing should validate your system with any 3rd party applications (credit cards, order fulfillment, etc.) used in production. The bottleneck you're searching for may be hiding there! At the same time, be sure not to create any real 'business' activity in your tests such as cash transfers, purchases, etc.

- **Running periodic or background processes**
  'Hidden' processes such as cleanup, backup, reporting, aggregation, or data exports are often part of a production environment., but are often forgotten during performance testing.. These tasks may consume significant machine resources and dramatically affect system behavior.

Mistake #3

# Cutting Corners

It's very tempting to compromise when building load test scenarios. Even when facing budget, resource or time constraints – be careful where you cut corners, so as not compromise the results of your load testing. Here are two common hazards you should avoid.

## Number of Users

Load testing a system with 100,000 users requires several machines, which may not be available. So why not use 10,000 users instead and have each perform a transaction every second, as opposed to a 10 second think time for the 'real' 100,000 user test?

The problem is that the two scenarios differ significantly. The server doesn't need to maintain the same number of connections for 10,000 users, memory requirements for 100,000 versus 10,000 users are vastly different, and so are database queries. You can't cut corners with the number of users and expect to realistically test performance.

## Data randomazation

It's much easier to generate 100 user profiles than 10,000 profiles. However, if the same 100 users repeatedly access the system and their credentials data is cached, you'll never have the effect of 10,000 different users accessing the system.

Mistake #4

# Starting too Large

The objective of load testing is to simulate a large number of users in a realistic environment. However, experienced load testers understand that to begin testing with a final load goal inevitably leads to failure.
Why? Well, let's assume that your final testing scenario is 10,000 users from five locations, over three types of devices, and with 10 different usage scenarios. If you run this up front, it will be next to impossible to isolate errors when they arise.
Instead, start with one user, one location, and one device. Create a testing scenario that grows gradually, and closely monitor for errors at each stage. Here are some more tips for "starting small":

## No load at all

Start testing web pages without any load test tool at all. There are several tools such as WebPageTest.org that you can use. This will allow you to verify that there are no issues on your site. For example, if a single page loads in 20 seconds, there's no point of running a load test, and it should first be fixed.

## Single scenario + single user

Before combining scenarios or generating load, run each scenario on its own for a full hour. This will help you identify issues such as memory leaks, errors, test data, and your actual test script.

## Single scenario, minimal load

Test each scenario with a small number of threads (e.g. 5 virtual users) before you jump into generating load. This will help you ensure that there are no deadlocks (on the simple situations) or other issues that affect your application just because a few users are working in parallel.

## Increase loads and/or mix scenarios

After verifying that the basic aspects of your scenario are working properly, you can start increasing the load and also consider creating a simple mix of scenarios. You can either increase the load for each scenario until you reach its maximum requirements, or start mixing scenarios first and only then increase the load.

Mistake # 5

# Overloading Load Generators

Load generator machines, while serving your testing goals, can also skew your test results. An overloaded load generator machine can create a situation where no load is generated at all, or load is generated but with skewed results. To detect whether load generators are overloaded, check:

## Load generator machine resources

- CPU utilization and memory usage
- Context switches per second - a high context switching number indicates that the
- CPU is less efficient, spending more time on itself than performing its task
- Page faults - when this number is high, the system is spending resources on writing data to disk, indicating inefficiencies and potentially harming performance

Disk queue length - while a queue for writing to the disk will always exist, if this number constantly increases it can indicate load generator overload. If queue length increases while load size remains constant, the indication is even stronger.

## Transactions per second

Compare the load generator machine with a 'probing client' machine – a separate machine that executes a single virtual user. Assume you increase the load size and see that TX/sec value is not growing linearly. By examining the TX/sec created by the probing client and comparing it to the load generator machine, you can determine whether your load generator is having trouble.

Mistake **# 6**

# Ignoring System Errors

Performance metrics and response time are understandably the key focus in load testing. But some system faults manifest themselves through system errors that are not so obvious - rather than a crash or a drop in response time.
Consider what happens, for example, if integration with an external tool or a background process has stopped working. Since this does not affect the user, it will not be evident in the response time. Another example would be an HTTP 500 error, which occurs in only a small percentage of cases, affecting only a small number of users.
To identify all system vulnerabilities related to load, pay attention to errors and suspicious behavior even when response time seems perfect. For example:

## User errors

Errors sent by the server, which would be viewed by users (such as HTTP 500), should be viewable in your load testing tool.

## Server-side errors

Check server log files to find errors on the server side, like exceptions or crashes of server-side components or services.

## Wrong data

In some cases, request-response flow and response time will be fine, but the data arriving from the server will be wrong. Make sure you have data validation embedded in your scripts to identify such cases.

Mistake #7

# Undocumented Testing / Load Scenarios

Re-running scenarios and comparing results between runs are an integral part of load testing. But when done multiple times, while tweaking and adjusting different parameters, application versions and test settings - it can become a nightmare to track the changes made in each test execution.
After several runs, it’s easy to become confused and forget which changes and settings belong to which test run, and whether your interpretation of results is correct. Here are some key items to be documented in each executed scenario, which will help you keep track of your test progress.

## Scenario rational/objective

Document the purpose of this specific execution – what does it aim to check or validate in relation to previous runs? What are the expected outcomes? What is the load size, and other key parameters that were changed relative to other runs?

## System under test settings

Which version/build of the software was tested? Which specific fixes does it include? In addition, take note of important configuration and settings changes made in the environment infrastructure or configuration.

## Test environment settings

Note important or changed settings in load generator machines and load testing tool settings, such as changes in caching or gzip.

## RESULTS/CONCLUSIONS

Record your own conclusions regarding each run, noting assumptions regarding the cause of issues and changes to be made in order to isolate the problem.

## THE BOTTOM LINE

Professional load testing tools are crucial to the success of your web site or application – and can dramatically impact your organization’s business as a whole.

But even after you choose your tool of choice, don’t forget that load testing is only as reliable and accurate as the testing environment and scenario.

Results from poorly-planned and poorly-conceived testing are by definition poor.

Take the time to create a detailed plan, ask the difficult questions – and you’ll already be well on the road to avoiding application load testing mistakes.

## About Radview

RadView Software provides WebLOAD, the world´s best value commercial-grade load and performance testing solution for internet applications. Deployed at over 3,500 customers, WebLOAD helps launching internet applications and with confidence.