Making Everything Easier!™

Compuware Special Edition

Web Load Testing

FOR DUMMIES®

Learn:

- **How web load testing helps enable better website performance**
- **When to load test**
- **Why web performance directly impacts your business**

Scott Barber
with Colin Mason

About the Authors

Scott Barber, Founder and Chief Technologist of PerfTestPlus, Inc., is viewed by many as the world's most prominent thought-leader in the area of software system performance testing and as a respected leader in the advancement of the understanding and practice of testing software systems in general. Scott earned his reputation by, among other things, contributing to three books:

- *Performance Testing Guidance for Web Applications,* Microsoft Press; 2007; coauthor
- *Beautiful Testing,* O'Reilly Media; 2009; contributing author
- *How to Reduce the Cost of Testing,* Auerbach Publications; 2011; contributing author

Scott has also composed over 100 articles and papers, delivered keynote addresses on five continents, served the testing community for four years as the Executive Director of the Association for Software Testing, and co-founded the Workshop on Performance and Reliability.

Today, Scott is applying and enhancing his thoughts on delivering world-class system performance in complex business and technical environments with a variety of clients and is actively building the foundation for his next project: driving the integration of testing commercial software systems with the core objectives of the businesses who fund that testing.

When he's not "being a geek," as he says, Scott enjoys spending time with his partner, Dawn, and his sons, Nicholas and Taylor, at home in central Florida and in other interesting places that his accumulated frequent flier miles enable them to explore.

For more about Scott, see his profile on about.me™ at `http://about.me/scott.barber`.

Colin Mason is the Product Manager for Gomez Web Load Testing at Compuware. Colin has over a decade of performance testing experience and has conducted hundreds of load tests for companies worldwide. He has presented at numerous industry conferences, including STAR, and has authored industry-acclaimed papers on performance testing web applications.

Web Load Testing FOR DUMMIES®

COMPUWARE SPECIAL EDITION

by Scott Barber
with Colin Mason

John Wiley & Sons, Inc.

Web Load Testing For Dummies®, Compuware Special Edition

Published by
John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030-5774

www.wiley.com

Published by John Wiley & Sons, Inc., Hoboken, New Jersey

ISBN: 978-1-118-16016-9

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents

Publisher's Acknowledgments

We're proud of this book and of the people who worked on it. For details on how to create a custom *For Dummies* book for your business or organization, contact `info@dummies.biz`. For details on licensing the *For Dummies* brand for products or services, contact `BrandedRights&Licenses@Wiley.com`.

Some of the people who helped bring this book to market include the following:

Acquisitions, Editorial, and Media Development

Project Editor: Carrie A. Burchfield

Editorial Manager: Rev Mengle

Custom Publishing Project Specialist: Michael Sullivan

Composition Services

Sr. Project Coordinator: Kristie Rees

Layout and Graphics: Samantha K. Cherolis, Lavonne Roberts

Proofreaders: Melissa Cossell, Jessica Kramer

Publishing and Editorial for Technology Dummies

Richard Swadley, Vice President and Executive Group Publisher

Andy Cummings, Vice President and Publisher

Mary Bednarek, Executive Director, Acquisitions

Mary C. Corder, Editorial Director

Publishing and Editorial for Consumer Dummies

Kathleen Nebenhaus, Vice President and Executive Publisher

Composition Services

Debbie Stailey, Director of Composition Services

Business Development

Lisa Coleman, Director, New Market and Brand Development

Introduction

Your company's website and web applications are critical to the success of your business initiatives. A business in today's digital culture needs to make sure that its website is working hard for the business and not against it. That's what web load testing is all about. Customer experience is the key to success on the web — web application performance is a priority. Think about the impact to your business (in other words, think about how angry the CEO and/or investors would be) if

- Your new application launch is delayed due to performance problems.
- Your site breaks under the load of your successful marketing promotion.
- High-traffic volume causes such poor web performance on your busiest online shopping day that abandonment skyrockets and conversions plummet.
- Your new infrastructure is configured improperly, grinding the website to a crawl.

Businesses need to focus on their web applications to protect revenue, reduce risk, and delight their customers, which makes web load testing a critical component to any risk management plan for web applications.

About This Book

In this book, we focus on load testing web applications because at the same time that the importance of web applications is rising and their usage is increasing, users' frustrations with poorly performing web apps are growing at nearly the same rate. Unfortunately, most web load testing approaches don't keep up with the evolution of web applications; we think we have some helpful things to say in that regard. We also briefly explain what a comprehensive testing program looks like while detailing web load testing as a part of the program.

Icons Used in This Book

The *For Dummies* series is notorious for its user-friendly look and feel, including the in-text icons that flag bits of information that are especially noteworthy. Here are the icons used in this book:

This icon targets practical and worthwhile hints that may save you time, effort, and stress.

This icon clues you in to information that you should stow away in your mental reserves — of course, you can always come back to this book for the reminders!

As you may have guessed, this icon points out potential dangers to avoid.

This icon points out helpful (but not crucial) bits of insight that you can skip over if you're pressed for time.

Where to Go From Here

You'll be happy to know that whatever your reading style, this book is sure to suit you. Sure, it's already concise — even so, you may not have the time to read the whole thing from cover to cover. You're free to do so, and we recommend that you do so you have a complete understanding of why we're here and how we can help you. If you're looking for specific information, however, feel free to use the Table of Contents to find the information you're looking for and head right to the pertinent section. We especially recommend that you check out Chapter 2 if you aren't familiar with outside-in load testing, which is what Compuware (and this book) are all about. We hope this book demonstrates to you the importance and value of web load testing and provides you with the information you need to enhance your current program or get your program off the ground. Happy load testing!

Chapter 1

Web Load Testing in a Nutshell

In This Chapter

- Figuring out what web load testing is all about
- Tying web load testing to business value
- Knowing what to expect from web load testing

Web load testing conducted properly is your best chance to ensure that your business critical web initiatives won't buckle under load or cause your customers to seek alternatives due to slowness. Whether you're launching a new website, preparing for holiday sales, deploying a new application for your employees in the field, or changing IT infrastructure, web load testing allows you to find and fix performance problems before going live — therefore reducing the risk of unwanted surprises. Web load testing may seem geeky, complicated, and time consuming, but in reality if done properly, it can save your company millions of dollars in lost revenue and brand equity, create customer loyalty, and lower the stress of deploying new initiatives. Web load testing is a crucial piece of your risk management plan.

In this chapter, we help you understand what web load testing is all about, what it's value proposition is to your business, and what different options you have for testing — as well as what we recommend.

What's Web Load Testing?

Web load testing is how you determine how much traffic your website or web application can accommodate without "breaking" or causing your customers to blog about how painfully slow it is. In general, the idea of web load testing is simple to understand. Web load testing is nothing more than exercising a website under a variety of production-like conditions to determine how it's going to work and to identify (and hopefully resolve) problems before your customers find them.

In practice, many people confuse the terms *performance, load, stress,* and *endurance/duration testing* or use them interchangeably, thinking they're all fancy terms for the same activity — but they aren't. While the differences may seem subtle when it comes to testing web applications, the time, effort, and goodwill lost when a manager and tester use these terms to mean different things can't be overstated. Here's a quick list highlighting the major differences:

- **Performance testing:** Designed to determine or assess speed, scalability, and/or stability characteristics of an application; a parent category that includes load, stress, and endurance/duration testing
- **Load testing:** Designed to determine or assess performance characteristics of an application experiencing anticipated load conditions
- **Stress testing:** Designed to find failure conditions or break points of an application that (hopefully) only occur under stressful conditions (by stressful, we mean the ones that someone says, "That'll never happen in production" — but it *does*)
- **Endurance/duration testing:** Load tests that run for a long time (in the neighborhood of 10 times as long as a "normal" load test) to detect emergent behaviors — 24 to 72 hours isn't uncommon

Web load testing is one part of a comprehensive performance testing plan. Establishing resource budgets, performance tuning, and capacity planning are just some of the other elements that you would include.

Knowing Why Web Load Testing Matters

If you haven't been living under a rock, you know that customers' website performance expectations are high and getting higher. The scary truth is web load testing often doesn't get done or is done in a way that won't tell you if customers' expectations are met. Many people believe that if a website is a bit slow, or if people have a little trouble accessing it during a peak traffic period, such as during a new marketing promotion, implementing a new application, or during a holiday sale, it doesn't have much of an overall impact on the business. After all, people must expect that when it's crowded they have to wait a bit, just like they have to wait in long lines at stores on black Friday — right?

Wrong. First off, unlike going to a store where you know it's going to be crowded and time consuming before you ever get inside (because you have trouble finding a parking place), website customers have *no* indication that a site is experiencing a high-traffic period, so even if they're willing to accept that as a reason for slowdowns, they have no way to know why a site is slow — at least not until the media coverage tells them, which is usually after they've already found an alternative way to complete their tasks. What website users expect — no, what they *demand* — is good performance that's consistent every time they visit the site.

According to a 2008 Aberdeen Group study, a 1-second delay in page load time equals 11 percent fewer page views, a 16 percent decrease in customer satisfaction, and a 7 percent loss in conversions. In dollar terms, this means that if a site typically earns $100,000 a day, this year you could lose $2.5 million in sales for every second slower your site is compared to last year (or, for that matter, compared to your competitor's website).

Poor website performance impacts a business regardless of the type of transaction or conversion that you track. A retail site measuring shopping cart purchases sees a loss of revenue due to fewer purchases, a hotel reservation system sees a drop in the number of reservations, and a bank may see fewer bill payments made.

Looking at website efficiency from another perspective, Compuware analyzed page abandonment data across more than 150 websites and 150 million page views and found that an increase in page response time from 2 to 10 seconds increased page abandonment rates by 38 percent.

Website performance has a direct relationship to business goals. So if your business is in the business of making money and you have a website related to your business, you should at least consider doing some web load testing.

Just as business has changed dramatically over the last 15 years with the evolution of the Internet, so have performance and load testing. Many people think that the answer to performance or scalability of a website is to add more hardware. Unfortunately, while that may have been true in the late 90s — with today's technology, adding hardware is at least as likely to make website performance *worse* as it is to make it better. Even when adding hardware is the answer, you almost never get the most out of your hardware investment without taking the time to test and tune your web application.

Figuring Out How Web Load Testing Works

If you decide to load test your company's website or web application more formally than just having all the company's employees click at it over lunch from a user's perspective, you have three options.

Testing user experience under load

Testing user experience under load, in which you execute your load testing 1.0 or 1.5 tests (described in Chapter 2) while real humans use the system to assess performance, is a good practice that's not used often enough and has only recently been named by Philip Nguyen of Citrix Systems. Expert load testers agree that implementing this practice serves to

- Link the numbers generated through load tests to the degree of satisfaction of actual human users

- Identify items that human users classify as performance issues that don't appear to be issues based on the numbers alone
- Convince stakeholders that the only metric to collect that can be conclusively linked to user satisfaction with production performance is the percent of users satisfied with performance during production conditions

User experience under load, however, is most effective when combined with full-scale web load testing delivered through self-service or hiring an expert.

Self-service

Self-service, in which you pay for the use of remotely hosted load-testing software and set up and run the tests yourself, is a great option for companies that already have a fairly comprehensive performance testing and tuning program in place but are looking to add the end-user perspective to their programs. If that doesn't sound like your company, you may want to hire a professional (see the next section on "Hiring a professional").

If you choose self-service, you're going to need either a bona fide web load testing geek on staff or someone familiar with a variety of testing who's available and willing to learn a new discipline — perhaps with help the first few times.

Features included in web load testing software provision cloud services, record test scripts, run the tests, and provide reports. When looking for self-service software, consider ease of provisioning, flexibility of scripting and setting up tests, and accuracy and relevance of the data/reports returned.

The organizations that provide self-service options are typically very good at completely taking the infrastructure aspect of web load testing off your plate, but the remaining tasks of designing, executing, and analyzing tests are tasks you may not want to tackle as a first timer.

Hiring a professional

You may decide to hire a professional to set up and run tests for you. If you choose to hire a pro, get your team together to

answer a bunch of questions about your needs, write a check, and then participate in looking at the results analysis data to determine what your business needs to fix or work around.

We recommend shopping around and checking references before writing the check. Web load testing pros, like most everything else, aren't all created equal.

What You Can (and Can't) Expect from Web Load Testing

Regardless of whether you choose to run the tests on your own or hire an outside provider to do it for you, you should know what you'll get from the testing. Table 1-1 provides a brief rundown of what you can (and can't) expect.

Table 1-1 What to Expect (and not) from Web Load Testing

Web Load Testing Does . . .	*. . . But Does Not . . .*
Implement usage models based on information at hand	Know how accurate that information is
Try to simulate realistic production conditions	Simulate everything that can or will happen in production; stress testing, for example, is a better choice if your goal is to preempt Murphy's Law (see the "What's Web Load Testing?" section earlier in this chapter)
Collect and compile a lot of data	Interpret that data for you — at least not if you want maximum value from that data; data analysis is a team sport!
Provide valuable information to business decision makers	Make pass/fail or go/no go decisions. That's what project managers and executives are for.
Focus on achieving a positive experience for end-users	Tell you what a positive experience for an end-user is without actually interacting with end users

Chapter 2

The Importance of Outside-In Load Testing

In This Chapter

- Looking at web application delivery
- Seeing the effects of load testing from the three main perspectives

At the end of the day, nothing has a larger business impact than the quality of the user experience delivered by your application. Remember: Users don't care if the application is busy; they want the same performance whether they're the only user on the application or one of thousands. The bottom line is that the overriding goal of any web load test is ensuring a quality user experience even under peak load. For this reason, outside-in load testing, or what we refer to as load testing 2.0, is crucial for success. This chapter explains why.

Understanding the Challenge of Delivering Web Applications

To really understand the different aspects of performance and load testing and why outside-in testing is such a big deal, you need some idea of how web applications are delivered. Web applications are delivered through an often complicated interaction of many semi-independent parts. These parts must come together to bring that web application through the application delivery chain to the end-user's screen.

A very simple (and outdated for all but the simplest websites) model is what's known as a 3-tier architecture, where the

web application has three significant and mostly separate components:

- The presentation tier or web server
- The business logic tier or application server
- The data tier or database server

In other words, the "pretty" parts of the web pages are one component, the thinking and decision making is another component, and all the information is the third component. Generally speaking, these components all live "next to" one another in what's called a *datacenter*.

Seems simple, but not anymore. In the past, only a few components existed together in the same data center that was likely to have a (more or less) direct connection to the Internet through a single Internet Service Provider (ISP). Today, web and mobile applications are composite and delivered to users via a complex web application delivery chain. Applications effectively treat the entire Internet as their datacenter and comprise content and web services delivered from the cloud, third party sources, and content delivery networks (CDNs), connected through different ISPs and mobile carriers from locations all over the globe. In fact, an average web transaction contains content delivered from over ten distinct servers. Figure 2-1 shows a typical and current web application architecture, known as "Web 2.0."

Notice in this figure that there are at least three components that the simple 3-tier architecture ignores — the components that are outside the datacenter:

- The stuff in your network between the Internet and the datacenter
- The Internet between the user and your network (for example, CDNs, Ad servers, third party ecommerce platforms, web analytics, newsfeeds, and so on)
- The user's machine and browser

These components all add variables that can have a significant impact to the potential success (or failure) of your website's ability to handle major fluctuations in load.

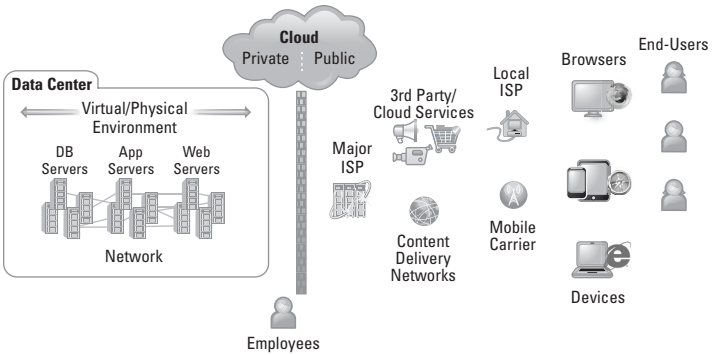

Figure 2-1: A typical Web 2.0 architecture.

Of course, architecture isn't the only thing that's evolved. Several years ago, web browsers were lightweight displays of information that were assembled on the fly as that information was received from a single source. Today, the average number of sources is far greater. Web browsers have advanced quickly to handle the implications of this evolution. With so many moving parts, the only viable way to determine and assess actual user experience is to test from the outside in.

Comparing the Different Testing Methods

Load testing, although one part of a comprehensive performance testing plan, contains its own set of variables. (Nothing involving technology is *ever* very simple, is it?) Essentially, you can load test your website from three different perspectives:

- **Inside out:** This method assesses code and individual components but not the web application as a whole.
- **Middle in:** This method assesses all the components in your data center but not the "last mile" to the user.
- **Outside in:** This method assesses the web application as a whole, accounting for all components.

The majority of application performance testing today is done either at the component level (inside out) or from within the corporate network (middle in). These methods are appropriate

for goals, such as tuning hardware and software components and don't answer the question, "What will the user experience be like?" Instead, you need to load test from the end-user perspective (outside in). A comprehensive *performance* testing program (which includes load testing as one part of the program) includes a "goals appropriate mix" of inside-out, outside-in, and middle-in performance testing.

While there is much value to be gained from performance testing your web application from each perspective, there's some value that you can realize *only* by load testing from the outside in. In order to understand that value, you need to step into your users' shoes.

In simplest terms, a quality user experience comes down to three components:

- **Success rate:** The percent of the time the application was available to the user and she was able to accomplish what she went there to do.
- **Response time:** The amount of time it takes for the page to render completely in the end-user's browser.
- **Quality:** The accuracy of the page as compared to its intended design and function.

Any time users experience low success rates, frustrating response times, or noticeably low quality in the presentation of content, we refer to that as the *user experience break point.* And while users used to tolerate a fair amount of failures, slowness, and poor quality, today's users are far less forgiving. Each year, they expect increasingly faster and flawless experiences, not just during application off peak hours, but every time they use the application.

In order to meet these expectations, you need to be able to measure performance under load from the customer's point of view. Your performance testing program needs a component to effectively test the entire web application delivery chain from the customer end in. We call that component *web load testing.*

In the same way that web application architecture has evolved, web load testing has also evolved. It is becoming common to refer to this evolution as load testing 1.0, 1.5, and 2.0.

Load testing 1.0 (from inside the firewall)

Load testing 1.0 adheres to the philosophy of generating load and measuring performance behind the firewall. (If you refer to Figure 2-1 earlier in the chapter, you see that this testing includes the datacenter alone.) This approach is the correct way to load test when your goal is to tune the components that *you* control to handle more load, respond faster, or simply make better use of existing hardware or resources. However, this is *not* a complete approach if your goal is to determine or assess the experience of your users.

Unfortunately, as many organizations have learned, most of the overall user perceived response time is eaten up by processes between the firewall and the datacenter within the company hosting the web application. When you test from inside the same space behind the firewall, you can't measure these delays. Moving the simulated users outside the firewall, allows you to see the delays inside the firewall — hence the move to load testing 1.5, testing from the cloud.

Load testing 1.5 (from within the cloud)

Load testing 1.5 is just a fancy name for "do what you were doing before, but from outside the firewall at least some of the time." In Figure 2-1 in this chapter, this means testing using the cloud, you can see that there are still a lots of parts in the application delivery chain that remain outside of your testing scope in this method.

There have been a variety of methods for doing this, including generating load from separate datacenters, generating load from the cloud, or even configuring computers as "load generation agents" in remote field offices. This approach leads to significant improvements in web application performance.

As it turns out, this approach has been so successful that most teams that are developing web applications today do a "good enough" job optimizing the components inside their firewalls, but that's still not the whole story.

Load testing 2.0 (from the user perspective, or outside-in)

So what's load testing 2.0? It's the form of load testing that provides the greatest insight into the end-user experience of real customers prior to releasing an application. It uses loads generated in high volume from the cloud, combined with load from actual desktops where your end-users are located. Load testing 2.0 lets you detect the kind of problems that lead to excessive user perceived delays in "the last mile," or between the cloud and your user's computer screen. In other words, load testing 2.0 enables you to determine and assess the entire user-perceived response time for a web application, not just some or most of it. Figure 2-1 covers the entire application delivery chain — no components are left out of the testing scope. It also allows you to

- Accurately measure user perceived response times under the load or loads of your choice
- Ensure that web applications scale under load
- Identify performance problems caused by third-party content and services

In other words, the value you can gain only from outside-in load testing includes

- Assessing actual application response times from a user's perspective
- Assessing differences in response times based on the user's browser type/version
- Assessing differences in response times based on the user's physical location in the world

Believe it or not, the items in the list most commonly result in lost revenue, brand damage, low customer satisfaction, and increased cost after release. Specific problems that can be found most easily by using load testing 2.0 include misconfigured CDNs, load balancing problems related to the number of IP addresses, and availability in a specific location. Poor or broken functionality is far less frequently the cause of a web application failing to meet the expectations of the business. Most major issues related to functionality aren't surprises to businesses, but most major performance issues are. In our experience, businesses don't like those kinds of surprises.

Chapter 3

To Test or Not To Test

In This Chapter

- Establishing application performance goals and web load testing goals
- Determining measures and metrics to assess those goals
- Using events to trigger critical web load tests

To get the most value out of your web load testing, you need to start by setting both application performance goals and business goals for the web load testing effort. With your goals established, gather the team and collaboratively agree on what you'll measure and what metrics you'll use to determine whether you've succeeded in achieving each goal. Testing early is obviously ideal, but some definite triggers still exist that indicate particularly valuable times to conduct a web load test. In this chapter, we guide you through all these tasks and help you find your ideal load-test triggers.

Gathering Your Team

In Chapters 1 and 2, we establish why web load testing is valuable to everyone who wants the business to succeed. Unfortunately, simply declaring its importance isn't enough. To maximize the value of web load testing, the entire team needs to contribute to the establishment of goals, and by entire team we mean the following:

- All the developers
- All the testers

- All the analysts
- All the managers (project, development, test, and so on)
- The project's executive sponsor
- And at least one representative from each of the following areas:
    - Product management
    - Marketing
    - IT/Support Operations

After the team is gathered, you can start determining your collective and departmental goals.

Establishing Goals

As you prepare for your web load testing, start by establishing the goals for the effort. Remember, you want to establish both application performance goals and goals for the web load testing effort.

As you begin, ask yourselves three fundamental questions to see where you currently stand:

- Will we be testing all of our internal and external components?
- Will we be incorporating a geographical load testing element into our testing plans to verify performance for our key regions and markets under load?
- Will we focus on validating end-user experience or simply stress testing our internal systems?

Department-specific goals

As a starting point to determining your goals, consider what marketing, operations, developers, and project managers believe the performance goals for the application are and what they believe the goals of the web load testing effort are. Take a moment right now to consider those areas yourself.

Done? We suspect you imagined different goals by area, but each goal and objective was in some way related to the overall success of the application.

Now consider the table of actual questions, concerns, and goals, in Table 3-1, from previous clients, organized by business area. How did your goals compare to these?

Table 3-1 Client Questions, Concerns, and Goals

Business Area	*Question/Concern*
Public Relations/Line of Business	"Will this web application strengthen or weaken our brand image?"
	"Will our customers use it happily or grudgingly?"
Marketing	"We just invested $XX in the marketing campaign, can we support the volume we need for a positive ROI?"
IT Operations	"We are doing all this extra work to enable this web load testing, what value will we get out of it?"
	"Can we use your test results to help us improve our hardware needs estimates?"
Development	"Did the changes we made last week improve or degrade performance?"
QA/Test Team	"Do all of the features and functions work correctly under load?"

Align your goals with what everyone on your development team is thinking because ultimately, everyone is invested in having an excellent end-user experience for all your customers.

General business goals

In addition to thinking about goals from the perspective of different areas of the business, it's a good idea to also consider the following questions, or very similar ones:

- Will the site break under the anticipated peak load?
- Can we handle unanticipated surges in traffic?

- Will our most important transactions and pages perform properly — regardless of load?
- Are both the web pages and the back-end systems optimized for fast performance?
- Will our customers have a positive experience no matter where they are located geographically?
- Are our site's third-party providers such as ratings and reviews, ads, news feeds, ecommerce engines and content delivery networks hurting our performance?

How valuable would you find a service that provided the answers to these questions before your application was released into production? How much more valuable would you find that service if it also provided an assessment of the accomplishment of those goals you established at your team meeting? That is the potential value of your web load testing efforts. The next step is to determine how to deliver that value.

The following list comprises web load testing goals from one of Compuware's recent clients (lightly sanitized to respect their privacy):

- Identify all pages/transactions with greater than 4 second (user-perceived) response times at least 10 percent of the time.
- Identify geographies/regions with greater than 4 second response times at least 25 percent of the time.
- Determine degradation point(s) (load level equating to the user experience break point) under top 3 usage models (models are discussed in Chapter 4).
- Identify cause and early indicators of degradation points found.
- Collect resource usage data requested by capacity planners and/or support operations.
- Determine user satisfaction levels by running 75 percent estimated peak load during UAT (User Acceptance Test) sessions.

- Confirm functionality under load by executing automated functional tests during web load tests.
- Conduct "specialty" tests to assist in tuning efforts as requested by project leadership.

These may not be your goals, but achieving these goals is valuable. If you or your organization is having trouble with goal setting, you may find it useful to use this example to seed a brainstorming session with your team.

Test Planning

Web load testing is just one part of a comprehensive performance testing program. Generally speaking, you should start with the first delivered components and build the simplest possible tests; then build in complexity as more components are delivered. The simplest possible tests aren't always web load tests, but by following this progression, when a bottleneck is uncovered during web load testing, you already know that all the simpler possibilities have been removed, and the newly introduced component almost certainly contributed to the appearance of the bottleneck.

The same concept applies to the web load testing part of the program. Start as early as there is something reasonable to test from the user's perspective, build the simplest test possible, and build in complexity as more of the application is available. Software geeks often call this an *iterative,* or *agile,* approach.

As it turns out, running a load test that achieves the overall goals for the application in the first iteration is rare, so structure the overall web load test plan to incorporate what you discover from every test into subsequent tests. In other words, design a plan that isolates the components of the system and application so they can be validated one by one against the goals. Then run the planned tests to identify which of those areas needs improvement. Make the necessary changes and then retest until the goals are achieved.

Figuring Out What to Measure

When it comes to web load testing, there's no standard list of what's valuable to measure. However, some measurements are common:

- User-perceived response time
- Response time variations by geography
- # of Virtual Users — measured at many points in time
- Resource utilization (for example, CPU, Memory, Disk I/O, Network Bandwidth) — measured at many points in time, at many locations
- # of Errors — definition of error may vary

We encourage you to ask "Is there a reason we aren't measuring X?" for any item on the preceding list that you aren't currently measuring. Some very good reasons exist for *not* measuring one or more of these things; sometimes, you have reasons to measure things not on this list. These measures are simply the default when we have no additional information to guide us. The easiest place to go for this additional information is simply to ask the development team what data it would be helpful to them for you to measure.

Devising Metrics to Add Meaning to Your Data

If you've been involved in software development for a while, you know that metrics is a controversial topic in the industry. If that is news to you, we'll explain briefly.

A *metric* is simply a measure of something that has meaning to someone(s) who cares. Some great examples of metrics include a baseball player's batting average or a marketing department's return on investment for a direct mail marketing campaign. These things can be measured and calculated to tell a story.

When it comes to software, especially when it comes to testing software, metrics get confusing. For example, some organizations report on the "test pass rate." Sounds good, right?

What percent of the tests that have been run passed? Here's the problem. What does a 96 percent pass rate actually mean? Does it mean that you did all the tests that you expected to pass first and haven't done the ones that you expect to fail? Were the 4 percent that failed the "are the graphics pretty" tests or the "can the customer buy our product" tests? As you can imagine, we could keep going, but the point is that for a metric to have meaning, it needs to be understood for what it is by everyone who will be exposed to the metric.

We wish we could give you a magic list of metrics to collect during your web load testing, but we can't (not in good faith, anyway). What we can do is *strongly* recommend that for each of your goals, work with one or more representatives from each group that is interested in that goal to establish one or more metrics that you collectively believe will answer, or help to answer, the questions:

- Has this goal been achieved?
- To what degree?
- What needs to be done to achieve this goal?

Metrics aren't the same as measures. *Measures* are just data; they may or may not tell a story. A baseball player's batting average (a *metric*) is calculated from two *measures:* base hits and at bats. You can't have metrics without measures, and you're likely to have many, many more measures than metrics.

Determining When to Test (and Re-test)

Clearly you cannot do web load testing against your application constantly. Well, I guess you could, but we've never encountered a situation where doing so would be cost effective.

Instead, organizations need to decide when it makes the most sense to engage in web load testing activities. Remember, as part of a comprehensive performance testing strategy, web load testing is not the only thing being done to prepare your application to perform well. Rather, it's the part of the

strategy that puts all the pieces together to finally answer the question "Will our users have a satisfactory experience, regardless of load?"

Several business, development lifecycle, and business external events can trigger thoughts of web load testing. You may certainly identify other times when you want to conduct web load testing, and you may certainly decide not to conduct web load testing around a trigger event identified below. We encourage you to stop for a moment and make an active decision about whether it's in your best interest to conduct web load testing around the following trigger events:

- **Business triggers**
  - Significant marketing campaigns
  - New product releases
  - Upcoming press coverage
  - Merger, acquisition, and IPO announcements
  - Litigation
- **Development lifecycle triggers**
  - Release candidate identified
  - New feature added
  - Back-end technology change/upgrade
  - Front-end website redesign
  - User acceptance testing
  - Beta releases
- **External triggers**
  - Holiday seasons
  - Weekly/Monthly/Quarterly patterns identified
  - Unanticipated press

These triggers really boil down to it being a good idea to do at least some web load testing before an expected increase in traffic, or before releasing new features/technologies into production.

Chapter 4

Designing Web Load Tests

In This Chapter

- Determining how much load to test
- Building your workload model(s)
- Avoiding common web load test design errors

It's time to start designing your web load tests. This chapter helps you ensure that the right people are discussing the right topics to end up with an appropriately designed test. The results of your web load test can never be any more accurate than the design of your test, so make sure you review this chapter carefully.

How Much Load?

While you want to try out a variety of load levels during test execution, both lower and higher than your target load to understand how performance characteristics change as load changes, you can't make an assessment about whether those changes are acceptable unless you know your target load. Determine what level of load to target for your tests.

When determining your target load, start with historic usage data. Look at the application's peak traffic in the past year — in terms of concurrent users, peak sessions in an hour, or in page views. If you don't have access to this data, this may be a good time to schmooze your favorite IT pal and ask what data exists and discuss implementing a web tracking analytics solution.

If this application is first generation or historical data doesn't exist, what's your next best step? Research — internally and externally. Collect historical data for similar applications in

your organization or competing applications from other organizations. You may be surprised how easy it is to find information about your competitors. Companies love to brag about load numbers when they're high, and the press loves to publish load numbers after noteworthy performance problems.

If neither of those approaches work, you may have to collect marketing projections and do some fancy math to extrapolate those numbers. After you establish a peak load number, set your goal beyond that level. Whether 25 percent beyond or ten times beyond, this decision is one for your team to make based on your expected growth and how "safe" you want to be.

When you're researching data on which to base your peak load number, check out the following four statistics:

- Peak number of user sessions seen on the application
- Peak number of page views seen on the application
- Average length of time an individual session lasts for each business process
- The average number of pages the typical user views in that session

Whether you're planning to conduct a web load test soon or not, find out whether your company is collecting this data for your production web applications . . . and if not, encourage them to start.

Selecting and Organizing Transactions

After you have a target load number, it's time to ask "What will all those people be doing during our simulation?" *Transaction* is a web load tester's fancy way of saying "a thing that people do on our site." As you probably guessed, the best way to determine what people do on your site is to mine historical data. Unfortunately, even if you have historical data, you can't

reasonably include every transaction in your web load test. If you have historical data, use it as a kind of a sanity check after you've built a transaction model from scratch.

Building from scratch

To build a web load test transaction model from scratch, we recommend that you hold a team meeting, in which you hand everyone seven index cards, each labeled with one of the following seven words that complete the sentence "Ensure your web load test includes transactions that are . . . ":

- **Frequent:** Common user activities.
- **Intensive:** For example, resource hogs.
- **Business critical:** Even if these activities are both rare and not risky.
- **Legally enforceable:** SLA's, contracts, and other stuff that will get you sued.
- **Obvious:** What the users will see and are most likely to complain about. What's likely to earn you bad press?
- **Technically risky:** New technologies, old technologies, places where it's failed before, under-tested areas.
- **Stakeholder mandated:** Don't argue with the boss (too much).

Ask each member to list the top five transactions for this web application in each category (less than five is okay, more than five isn't, and order doesn't matter). While the meeting continues, tally up the transactions (ignoring the categories, so "Login" could be listed by one person in several categories . . . that counts as several tally marks). Near the end of the meeting, reveal the tally by making a list of the transactions in order of frequency; draw a line somewhere on the list indicating that the transactions above the line will be part of the web load test and the ones below won't.

If no one complains, you have your list. If someone complains, tell them that's the maximum number of transactions that you have time to create, and see whether the team is willing

to remove any of the transactions above the line on the list so that person's transaction will have a place in the simulation. Don't hesitate to modify this approach to work within your corporate environment.

Now that you know how many users you're targeting and what transactions you want to simulate, your next step is to determine how many users are performing which transactions in what sequence.

Looking at historical data

Historical data comes in handy if you have it, but whether you have it or not, the approach is pretty much the same:

1. **Draw a picture of how all users navigate the site to accomplish the transactions you identified.**

   See Figure 4-1.

2. **Imagine three situations where the number of people doing each task is significantly different, for example:**

   - Non-holiday — search heavy
   - Holiday special — add to cart and order heavy
   - Week before holiday — check order status heavy

3. **Print out three copies of the picture for one or more representatives from each group on the team.**
4. **Label each copy with one of your three situations.**
5. **Ask each representative to fill out the percentage of people they believe will be doing each transaction during each situation.**
6. **Collect the answers, average them, adjust them based on historical data, possibly exaggerate the highs and lows for a more inclusive range of test results, and call the job done.**

   You can consider this part done at least until you start getting results from your tests that suggest other distributions may be more valuable.

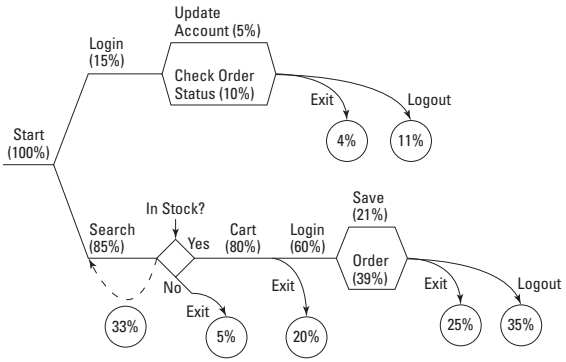

Figure 4-1: An example of navigating the site.

Additional Considerations

A huge number of possible additional considerations exist when it comes to designing as-realistic-as-reasonably-possible web load tests. The key is to quickly figure out which ones matter, account for them as accurately as you can without detrimentally delaying the testing effort, and run some tests. Remember, it's more important to find major issues than to make perfect predictions.

At least account for the four items in the following sections before conducting any web load tests approaching your target load, then enhance your design with additional realism between tests when the rest of the team is scrambling to resolve the major issue uncovered by the previous web load test.

Navigation

One of the most frustrating truths about testing web applications is that there seem to be 12 ways to accomplish every task . . . and that's before you account for things like the back button, people getting called away in the middle of a task, or human error. This human variance in navigation paths is no less relevant during web load testing than it is during functional testing. However, the emphasis with web load testing is to test the most likely and critical transactions. The number of possible combinations is, for all practical purposes, infinite for a typical website, so don't go down the "complete coverage"

road. Ensure that some navigational variance is built into the design of your web load test.

Pacing

Almost any tool you use to create web load tests captures the amount of time *you* spent on a particular page while recording a transaction and later uses that time as how long to stay on that page before moving to the next during test execution. That is far from realism. Most of your users almost certainly spend more time on each page than you — even when they're on task. Your users take phone calls, get drinks for their kids, answer the door, or even go take a nap in the middle of trying to accomplish a task on your website. Their sessions time out, Internet connections drop, and so on. So take those recorded delays and vary them. Make every page delay a bit different, make use of those "pick a random number between x and y" features, and make it so that some of the virtual users randomly stop what they're doing and never come back.

Geography

Because the focus is web load testing, which focuses on actual user experience, you need to incorporate the geographic location of your users into your test design. Where are most of the application users located? Is a marketing campaign targeting a new geography? What are the top revenue generating regions?

Test from precisely where your customers are. Ensure that you test from the correct locations and also from the last mile. Your customers don't live in the cloud and their experiences on their browsers can be vastly different by location.

Browsers/devices

Your users will access your web applications from different devices, on different browsers, in a variety of combinations that are likely to shock you. You can't account for them all, but you need to account for a healthy sampling in your tests.

Don't limit your tests to the most modern or the minimal recommended hardware/software versions. Your users come from the entire spectrum, and your application can be tuned to provide a better user experience on either old or new.

Chapter 5

Executing Web Load Tests

In This Chapter

- Determining the tools needed to execute web load tests
- Evaluating web load test environments
- Validating data before you begin
- Running through the execution cycle

As you may imagine, the preparation and execution of web load tests can be particularly technical, but just because it generally takes the techies in your company to do this part of the job doesn't mean you should ignore it and simply assume they can handle it entirely on their own. Preparation and execution includes a wide range of practical, business, and resource allocation considerations that are almost always above the responsibility of the web load tester.

Before you even begin running the tests, you need to coordinate test execution and monitoring with the team and then validate tests, configurations, and the state of the environments and data. This chapter covers the tools and conditions needed to run the tests as well as pre-execution and execution tasks and common mistakes to avoid.

Selecting a Tool (or Tools)

The key to tool selection is first determining what you need from the tool, then finding a tool that does what you need. Generally speaking, to conduct highly valuable web load testing, a tool, or collection of tools, should be able to do the following:

- Simulate transactions realistically and accurately
- Generate the volume of load you desire
- Collect the measures and metrics you've established
- Perform appropriate diagnostic capabilities

If you have previously evaluated a tool for load testing, you already know that a lot of tools are available that claim to be able to do the things in this list. Some are more expensive than others, some are easier to use than others, and some have more bells and whistles than others. This shouldn't surprise you.

If your task is to drive a nail, you can grab a rock, you can buy a hammer, or you can buy a pneumatic air gun. Which one you choose depends on how many nails you want to drive, how much money you have, how important the face of the wood is, and whether you need it just this once or will be driving nails for years to come. Of course, if you choose to buy a hammer, you still have to decide which one, how heavy you want it to be, and so on. This is exactly what selecting a load testing tool is like.

There is one consideration that's particularly relevant when it comes to a web load testing tool: A tool for web load testing 2.0 *must* simulate and measure client-side activity. The response time measurements must be the same as if you used a stopwatch to measure the time it takes from "click" to "display complete." The tool needs to simulate actual end-users as closely as possible by using commercial browsers, replicating device combinations, and most importantly testing from where the users are geographically.

Discussing diagnostic tools

Many organizations are finding diagnostic tools to be invaluable additions to their web load testing toolkit. Diagnostics tools greatly simplify the task of identifying the root cause of the performance issues your load testing uncovers — often right down to the offending line of code. This discovery is particularly valuable for shortening and simplifying the analysis cycle discussed in Chapter 6.

If you've previously conducted load testing of the 1.0 or 1.5 variety, you may believe that you already have the tool you need for web load testing 2.0. That isn't necessarily true. Tools for load testing 1.0 and 1.5 tasks need to record and replay traffic "on the wire" — which means that these tools do not simulate or measure any activity on your user's device such as client-side scripting, or browser activity. In other words, they can't directly collect user-perceived response times, for example.

Don't ask your web load tester to force the tool you have to do web load testing 2.0 tasks with a load testing 1.0 or 1.5 tool. That's like asking someone to use a hammer to drive a drywall screw — it *might* be possible, but it most certainly isn't worth the effort.

Creating the Right Test Environment

The goal is for the test environment to mirror your production environment as closely as possible, and that test environment needs to be explicitly designed before you start creating your web load tests. If the test environment is even slightly different from the environment you designed your tests to be run against, there's a high probability that your tests may not work at all, or worse, that they work but provide misleading data. Test environments that differ from production can be exceptionally valuable for other aspects of a comprehensive performance program, but this situation is one you should avoid if at all possible when executing web load tests.

The following are some additional items to consider related to test environments:

- Ensure that the test environment is correctly configured for metrics collection.
- Consider simulating background activity, when necessary. For example, many servers run batch processing during predetermined time periods, while servicing users' requests. Not accounting for such activities in those periods may result in overly optimistic performance results.

- Start by running simple web load tests to ensure the environment is configured correctly and that your tests are functioning as designed.
- Increase volume and complexity incrementally as you validate.
- Don't expect to run a full usage scenario web load test at the target volume on your first try successfully.

Validating Test Data

Load testing consumes a lot of data. A realistic web load test must use a variety of test data sufficient for each simulated user to be as unique as a real user. This volume of test data brings its own challenges.

After you're satisfied that your tests are running properly, the last critical validation step is to validate your test data. Run a test that loops through all your data to check for unexpected errors. If appropriate, validate that you can reset test and/or application data following a test run. Report performance issues uncovered during these test runs. Consider cleaning the database entries between error trials to eliminate data that may be causing test failures; for example — partially completed order entries that you can't reference in subsequent test executions.

Don't use results from validation test runs to make business decisions. Tests run for validation are deliberately unrealistic compared to the expected usage of your website. This is done to make it easier to test the tests. However, any performance issues you detect are worth looking into more deeply — if for no other reason than validation is normally conducted when there's still time to fix performance issues without delaying the release date.

Running the Tests

Executing tests is what most people envision when they think about load testing, and most people think it's pretty simple. In fact, even many people who have attended training on load testing treat test execution as little more than starting a test

and monitoring it to ensure that the test appears to be running as expected. In reality, this activity is significantly more complex than just clicking a button and monitoring machines.

Test execution can be viewed as a combination of the following subtasks:

- Use the system manually during test execution so you can compare your observations with the results data at a later time.
- Run other tests during your performance test to ensure that the simulation isn't impacting other parts of the system. These other tests may be either automated or manual.
- Observe your test during execution and pay close attention to any behavior you feel is unusual. Your instincts are usually right, or at least valuable.

As we discuss in Chapter 3, web load testing is an iterative process. Even after you've run simple tests to validate your environment and your tests, you should expect that after every test or two, there will be results that lead you to want to explore something more deeply, or make changes to the application with the intent of improving it. Deeper exploration and improvements each tend to lead to creating new tests, with new goals, measures, and metrics that will be executed several times before returning to the web load tests you have planned.

The following are some additional items to consider related to the load test cycle:

- If you'll be repeating the test, consider establishing a test data restore point before you begin testing.
- No matter how far in advance a test is scheduled, give the team 30-minute and 5-minute warnings before launching the test (or starting the day's testing). Inform the team whenever you're going to be executing for more than one hour in succession.
- Maintain a test execution log that captures notes and observations for each run.
- Run test tasks in one- to two-day batches. See the tasks through to completion, but be willing to take important

detours along the way if an opportunity to add additional value presents itself.

- Remember to simulate ramp-up and cool-down periods appropriately.
- Plan to spend some time fixing application errors, or debugging the test.
- Analyze results immediately so that you can modify your test plan accordingly.
- At appropriate points during test execution, stress the application to the target load, as this can provide extremely valuable information.
- If at all possible, execute every test twice, adjusting variables such as user names and think times to see if the test continues to behave as anticipated. If the results produced aren't very similar, execute the test again. Try to determine what factors account for the difference.
- Test execution is never really finished, but eventually you'll reach a point of diminishing returns on a particular test. When you stop obtaining valuable information, change your test.
- Communicate test results frequently and openly across the team.
- Treat your load test design as a moving target. Adjust as you learn. Ask yourself:
    - Have recent test results or project updates made this task more or less valuable compared to other tests we could be conducting right now?
    - What additional team members should be involved with this task?
    - Do the preliminary results make sense?

Web load test execution is a team sport. Your web load tester won't be doing deep exploration into results, at least not alone. Your web load tester will neither be deciding what measures and metrics must be improved, nor making the changes to improve them. In fact, if you want to maximize the value of your web load testing program, you don't want your web load tester analyzing results alone either, but that discussion is coming in Chapter 6.

Chapter 6

Analyzing and Reporting Results

In This Chapter

- Getting the right people involved in results analysis
- Identifying and handling performance issues
- Building reports that make sense

For your web load testing effort to add significant value to your project, the team needs analysis, comparisons, details behind how the results were obtained, conclusions, and consolidated data that supports those conclusions. Results analysis needs to be an ongoing team activity, and reports need to tell clear stories. This chapter helps you do just that.

Analyzing Data

Data analysis needs to be a collaborative, real-time effort that begins while the test is executed. Have the analysis team use the application and monitor its components while the test is running because this speeds up and smoothes out data analysis of web load test data. The entire team gets the context it needs to make sense of the data that can only be viewed or consolidated after the test completes. To configure your analysis team, gather the following:

- All involved web load testers
- The system architect or lead developer
- Datacenter/system administrator(s) (including technical representatives for outsourced components)

- Database personnel (designer *and* administrator/maintainer if not the same person)
- A senior member of the functional QA/Test team
- A business analyst or user representative
- At least one non-technical manager or executive
- At least one person with the following skills (can be any of the preceding people):
  - Spreadsheet
  - Statistics
  - Data visualization

Looking for Problems

Analysis of web load test results is more than finding slow pages and how many users can cram into the site before it gets painfully slow. Lookout for four classes of issues:

- **Bottlenecks:** Systemic issues that cause user slowdown
- **Slow spots:** Isolated pages, objects, or activities that take a long time but don't impact other users' activities
- **Limits:** Points at which performance changes noticeably
- **Defects/errors:** Things that are broken or don't work

After an issue is detected, the classification process starts:

1. **Determine the complete impact of the issue.**

   Was it a bottleneck? If so, you likely have a systemic problem — something in the system, network, or application that's shared by all users. Was the problem a slow spot? Potentially the problem was limited to users in certain geographies, or maybe a graphic is unacceptably large.

2. **Determine whether the issue was related to a limit of some kind.**

   Compare the results with the utilization of the system, application, or network. Commonly, you may see activities, such as a CPU utilization spike shortly before or after the moment of the observed issue. This kind of info provides the first clues to the cause of the issue.

Defects or errors are usually easier to classify. Either there was or wasn't an issue. Much of this information comes from using the application manually during your test and reviewing log files of information collected during the test.

Log file parsing is a tedious exercise, but we strongly recommend that you don't leave it out of your analysis because when defects or errors show up in log files, they have a nasty habit of being ones that you want to fix — quickly.

Identifying causes

Finding causes often involves additional testing. Use your data and observations to rule out features or systems that weren't involved and not the cause. For the best chance of quickly determining the root cause of bottlenecks, check out the following three key data sets:

- **User experience:** The metrics to review in this area are those identified in the test planning phase: response times, success rates, and quality. The user experience metrics are the most visible indicators of overall performance. If response times and error rates are the same at one user as at 1,000 users, there wasn't a bottleneck. But if errors start to increase or response times degrade, it's an indicator of a problem and one that real users will quickly experience under peak conditions.
- **Amount of traffic being generated and amount of load on the system before problems emerge:** For load-related bottlenecks, the key metrics are all around the levels of traffic on the system. Focus on the number of users, the throughput rates in terms of page requests per second, object hit rates, and bandwidth (megabits per second). Ideally, as the number of users increases, the throughput numbers increase, meaning that as demand increases, there's sufficient capacity to meet that demand. The point at which users or demand increases but the throughput doesn't increase is the capacity limit or bottleneck.
- **System level utilization of the hardware that is supporting the application:** These metrics correlate to how hard the base system is working in response to the increased traffic. It would be expected that as load increases so too would utilization.

Consider the overall performance from the end-user perspective against the system utilization. A significant difference exists between acceptable response times at the peak traffic levels with 25 percent system utilization versus 95 percent system utilization. In the former case, the outlook is quite good. In the latter, the peak traffic isn't far away from causing problems, meaning there's little margin of error and no room for growth.

One caveat to the increasing load not producing increased utilization is in configuration issues. For example, maybe a web server or firewall is configured to allow a certain number of simultaneous connections; after that limit is exceeded, new users aren't allowed access. Response times increase, errors may be encountered, but system utilization won't increase.

An appropriate diagnostics tool can greatly simplify cause identification and resolution. Diagnostics tools are useful for pinpointing the root cause of issues, and they foster the collaboration common in agile teams that's necessary to support quick issue resolution by providing a common set of tools for developers, web load testers, and support operations teams to work from. The ability to collectively look inside the datacenter after a web load test often means that teams can resolve problems without running additional tests to conclusively identify and reproduce them.

Dealing with Issues

Issues are going to come up, but identifying those issues is important. The following list helps you identify the issues. Listed below in order of frequency, based on Compuware's experiences, coworkers, and friends, are the main ways:

- Using the system under load and thinking "I wouldn't want to use this!"
- Technical team members monitoring their components during load tests thinking "Oooo, that's not good," or "Why is that measure doing *that*?"
- Finding error messages. Usually in logs after the test completes, but sometimes during the test.

- Identifying metrics that aren't meeting stated goals.
- Identifying "not good patterns" (sometimes called "anti-patterns") in results while analyzing data.
- Identifying "not good patterns" of results when comparing multiple test executions.

The data analysis team looks for the following bad patterns:

- Does it take longer to complete transactions as the load test progresses?
- Do page response times change as the load test progresses (all or just some)?
- Do errors increase as the load test progresses?
- At what point does user experience fail to meet the stated goals?

Due to the nature of performance testing in general and web load testing in specific, it's typically impossible to prove the difference between an outlier and an issue or between an application issue and a testing issue through testing alone. This is another reason to have a data analysis team.

When someone on the data analysis team identifies something that makes them think "hmmm . . . that doesn't seem right/good," try executing the test again — possibly modifying the test with the intent of exploiting what's wrong. If the problem is still evident, tell the rest of the team. If someone thinks he knows what's causing it and can do something to fix or prove it quickly, do it, and then rerun the test. If you think a special skill will help make sense of the data/observation, call in someone with that skill to help.

If none of that works (or you've already spent more than double the test execution time trying to figure out the issue), and you've run the test at least twice with the same result, log it in your defect tracking system. After it's in the system, either move on to the next test or call an immediate team meeting to figure out how to handle the issue. The urgency depends on how major the issue is and whether it jeopardizes the value of continued web load testing. If you find issues that aren't directly related to your goals, do one or more of the following:

- Revisit your measures and metrics to find a better way to assess your goals.
- Revisit your goals to see what you forgot, assumed, or misinterpreted.
- Forget the goals and metrics for a while, figure out what it takes to give the users a positive experience; do what it takes to give it to them, and figure out how to articulate goals and what measures and metrics to use to assess those goals later.

Reporting Sensibly

Reporting the results of one or more web load tests is difficult. The key to effective reporting is to present information of interest to the intended audience in a manner that is quick, simple, and intuitive. Additionally, *always* report against established goals and targets, and only report on things that aren't goals or targets when they're causing end-user dissatisfaction or violating an unstated requirement of the system (for instance, orders appearing to be taken but not appearing in your order tracking report).

Technical reports should include a test description, easily digestible data, access to the data set and test conditions, short analysis statements, and currently unmet or untested goals/targets. Stakeholder reports should include current results for all goals/targets; visual representations of the most relevant data, workload model, and test environment with corresponding summaries; access to associated technical reports, data sets, and test conditions; and summaries of observations, concerns, and recommendations.

Master web load test reporters are able to present all that information on a single page or slide with the viewers of the report drawing the correct conclusions without confusion or having to ask questions. We haven't met many master web load test reporters, but we encourage all web load test reporters to strive for this level of mastery.

Chapter 7

Ten Common Myths and Misconceptions of Web Load Testing

In This Chapter

- Recognizing critical terminology differences
- Identifying faulty assumptions

Web load testing isn't the easiest thing in the world to accomplish successfully, but it doesn't have to be grueling, either. By noting and heeding the common snares highlighted in this chapter, the process of optimizing your web applications can be simpler (and less stressful).

Three Terms, Same Meaning

Performance engineering, performance testing, and *capacity planning* are three distinct activities:

- Performance *engineering* refers to the activities associated with designing and developing an application in a particular manner to achieve desired performance characteristics.
- *Performance testing* entails determining or assessing the current performance characteristics of an already developed application or application component.
- *Capacity planning* is an activity driven by IT Operations that uses mathematical models to estimate the hardware required to support certain application loads and plan for future growth.

Using the correct term at the correct time not only makes you look really smart in front of your peers and superiors but also saves a lot of time and confusion.

Industry Standard Response Time = Satisfied Users

When it comes to user satisfaction, it's not a question of numbers; it's a question of human psychology. There's no response number that makes all users happy, and some users report the site as being "slow" even if it had infinitely fast response times (because the design of your site causes it to take them longer than they find acceptable to accomplish their task).

Don't waste your time searching for some number published by some reputable organization to be your response time goal. Benchmarking is useful as a place to start, but spend the effort figuring out what *your* users find acceptable.

Page Load Times Easily Correlate to User Perceived Response Times

Users regularly don't even *notice* how long it takes a page to load completely; frequently they stop paying attention after the item they're looking for appears. As far as they're concerned, the application response time is from the time they click until the time they find what they're looking for or are no longer waiting on the page to continue trying to accomplish their tasks.

Sometimes, the total page load time and the user-perceived response time are very similar, and sometimes they're not. Your job is to help your company deliver an application that satisfies the user, so optimize the page load times as much as you can. If that's not enough, optimize the way the page loads so the user can get to what she's looking for before becoming annoyed by waiting.

High Traffic Means Slow Performance Is OK to Users

Users don't care how many people are on your site, where others are connecting from, or what device they're using; they have (or form) expectations for the performance of your site, and they aren't happy when those expectations aren't met — no matter what good reason there is for performance differences.

"Takes Too Long" Always Refers to Page Load Times

Many times users say things like "It takes too long to do *x* on your site." Sometimes that means that the site isn't performing well. Sometimes it means that the workflow of your site isn't designed well. If a user is used to completing a form on a single page, but your site breaks the form up into four pages and changes the order of data entry, the user will tell you that it takes too long no matter how fast those four pages load. What he's saying is that it takes too long to complete the task. Make sure you do your homework when you hear phrases like "takes too long" before deciding whether the issue is performance related or design related.

Reducing Think Time to Zero Is a Good Way to Simulate More Users

Reducing the think time between page requests so you can generate more virtual users in a shorter period of time doesn't generate a load signature that's even vaguely similar to adding more realistically modeled virtual users.

To put it another way, condensing the activity of a typical 15 minute user session (the time a user spends interacting with a website in an uninterrupted manner) down to 1 minute by eliminating all the time that user actually spent reading, thinking, typing, and/or staring off into space (called "think time") and looping the condensed version 15 times back to back using a single virtual user (a computer generated user designed to simulate a real user of the system — many tools license by number of virtual users available to the web load tester) is absolutely nothing like using 15 uncondensed virtual users each performing the user session once during that same time.

Concurrent, Simultaneous, and Per-Hour Are Equivalent Load Measurements

Concurrent users refers to the number of active sessions at a particular moment in time, which changes from moment to moment. *Simultaneous users* are all executing the same set of actions at the same time, which is a practical impossibility for systems used by humans over the Internet. The number of users per hour (you can choose a different period of time, but hour tends to be appropriate for a web application) is the most correct and most easily understandable way to articulate load.

Functional Test Use Cases Are a Good Place to Start

A web load tester can't just script the functional use cases and call that a load test. Well, we guess they could, but it would be a very bad load test. This doesn't work for several reasons:

- Functional use cases tend to be short, overlapping, and numerous (20 usage scenarios is considered a lot for a load test).
- Functional use cases rarely represent a sequence of events that real users would deliberately follow.
- Functional use cases aren't designed or prioritized to find or highlight performance issues.

Using functional use cases as a "sanity check" can be extremely valuable, but other than that, put them away and design your load tests from scratch.

A Fractional Production Environment Allows Accurate Performance Predictions

Technically it's possible to extrapolate the results from a fractional production environment for load testing, but there are very few people who are good at it, it is very hard, and even the slightest difference between "expected" production and "actual" production can completely invalidate the prediction. In fact, those few people who can do this well *always* check their predictions against actual production measures and metrics and then revise their predictions to account for whatever differences are in "actual" production versus "expected" production. In load and performance testing lingo, this is called extrapolation, and it is usually followed by words such as "is evil."

Bottlenecks, Errors, and Slow Spots Mean the Same Thing

Bottlenecks, errors, and slow spots are different in important ways. Bottlenecks impact *all* users. Errors are items that *should* have been resolved before web load testing ever began (but weren't for any number of possible good or bad reasons). Slow spots are isolated items or activities that are slower than users prefer (or maybe slower than they're willing to tolerate) but are isolated to individual users, objects, or activities.