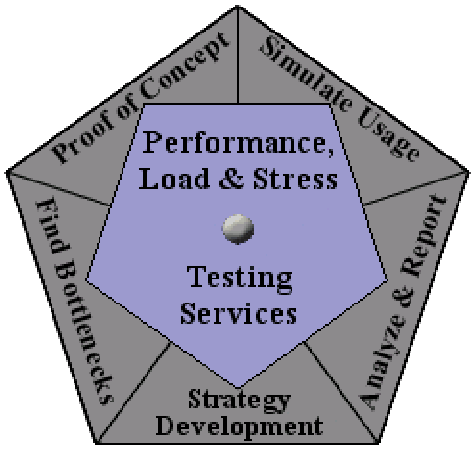
PerfTestPlus
Better Testing... Better Results

Beyond Performance Testing

by:

R. Scott Barber



BPT Part 1: Introduction

"Computers are useless. They can only give you answers."

– Pablo Picasso (1881—1973)

Performance testing is the discipline concerned with determining and reporting the current performance of a software application under various parameters. My series "User Experience, Not Metrics" discusses and demonstrates how to do it, with the help of the Rational Suite® TestStudio® system testing tool. Computers excel at this stuff, crunching numbers and displaying answers in neat ways. But there comes a time after the tests are run when someone who's reviewing the results asks the deceptively simple question, "So what, exactly, does all this *mean*?!?" This point beyond performance testing is where the capabilities of the human brain come in handy.

"Computers are good at swift, accurate computation and at storing great masses of information. The brain, on the other hand, is not as efficient a number cruncher and its memory is often highly fallible; a basic inexactness is built into its design. The brain's strong point is its flexibility. It is unsurpassed at making shrewd guesses and at grasping the total meaning of information presented to it," writes British journalist Jeremy Campbell in Chapter 16 of *Grammatical Man: Information, Entropy, Language, and Life* (1982).

"Beyond Performance Testing" will address what happens after initial test results are collected, the part it takes a human brain to accomplish. We'll explore what performance test results mean and what can be done to improve them. We'll focus on performance tuning, the subset of performance engineering that complements performance testing. We'll examine the process by which software is iteratively tested using Rational Suite TestStudio and tuned with the intent of achieving desired performance, by following an industry-leading performance engineering methodology that complements the Rational Unified Process® approach.

This first article is intended to introduce you to the concept of performance engineering that underlies the series and to give you an overview of the articles that follow. "Beyond Performance Testing" builds on and is a companion to the "User Experience, Not Metrics" series, so you should be familiar with the topics presented in that series before you begin reading this one.

About Performance Engineering

Performance engineering is the process by which software is tested and tuned with the intent of realizing the required performance. This process aims to optimize the most important application performance trait, user experience.

Historically, testing and tuning have been distinctly separate and often competing realms. In the last few years, however, several pockets of testers and developers have collaborated independently to create tuning teams. Because these teams have met with significant success, the concept of coupling performance testing with performance tuning has caught on, and now we call it performance engineering.

Let’s begin by exploring this concept at a high level.

The performance testing part of performance engineering encompasses what’s commonly referred to as load, spike, and stress testing, as well as validating system performance. You may also have heard other terms used to describe aspects of what I’m calling performance testing.

Regardless of the terms you may use to describe testing it, performance can be classified into three main categories:

- **Speed** – Does the application respond quickly enough for the intended users?
- **Scalability** – Will the application handle the expected user load and beyond?
- **Stability** – Is the application stable under expected and unexpected user loads?

The engineering part comes in as soon as the first measurement is taken and you start trying to figure out how to get from that measurement to the desired performance. That’s really what engineering is all about: solving a problem to achieve a desired and beneficial outcome. As a civil engineering student at Virginia Tech in the early 1990s, I was taught an approach to solving engineering problems that can be summarized as follows:

- **Given** –What are the known quantities?
- **Find** – What are the requirements for the desired item/outcome?
- **Solution** – How can we build or acquire the desired item/outcome, meeting the requirements, within the parameters of the known quantities?

Or as my father used to say, "What have you got? What do you want? How do you get there?" In civil engineering, the implied problem was always "build or modify a structure to satisfy the given need." In performance engineering the implied problem may seem different, but it’s fundamentally the same – that is, "build or modify a computer system to satisfy the given performance need."

With that said, I suspect you would agree that performance is the trait of the system that we wish to engineer, thus performance engineering.

Overview of This Series

"Beyond Performance Testing" is a fourteen-part series, with parts to be published monthly. Each article will take a practical approach; it will discuss how to immediately apply the methodology or technique being introduced and will give examples and case studies taken from real performance engineering projects. Where feasible, articles will also include "Now You Try It" exercises so that you can practice applying the methodology or technique on your own. Articles that don't contain specific exercises with solutions will contain approaches, checklists, and warnings that will help you apply what you learn directly to your projects. In many cases, I won't be able to address all of the technical issues specific to your system under test, but you'll find out what questions to ask.

Each article will be identified as beginner, intermediate, or expert level, depending on the technical complexity of the code or systems analysis required to accomplish the technique being discussed. The concepts presented in each article will be valuable to all members of the performance engineering team, from the project manager to the most junior developer.

Following is a brief description of the articles.

Performance Engineering Housekeeping

Parts 2, 3, and 4 start us out with what I think of as "performance engineering housekeeping" topics. These three articles aren't particularly technical but will give us the common footing we need for the remaining articles.

- **Part 2: A Performance Engineering Strategy** – The performance engineering methodology that serves as a basis for applying the concepts in later articles is described here. This article also outlines a performance engineering strategy document (equivalent to a functional test plan) that can be used to help organize and document performance engineering activities, plus a performance engineering results document.
- **Part 3: How Fast Is Fast Enough?** – One of the biggest challenges we face in performance engineering is collecting performance-related requirements. This topic was touched on in the "User Experience, Not Metrics" series and is discussed here in more detail.
- **Part 4: Accounting for User Abandonment** – If an application is too slow, users will eventually abandon it. This must be accounted for in both modeling and analysis, because not accounting for abandonment can drastically change your results and tuning strategies.

Detecting and Tuning Bottlenecks

Parts 5 through 10 are technical articles focused on detecting, exploiting, and preparing to tune bottlenecks. These articles follow the steps of the approach we discuss in Part 2 and continually relate back to the performance requirements we discuss in Part 3.

- **Part 5: Determining the Root Cause of Script Failures** – The obvious symptom is very rarely the actual cause of a script failure. It's imperative to determine if the failure is due to the script or the application and be able to quantify the root cause, not just the symptoms.

- **Part 6: Interpreting Scatter Charts** – Scatter charts are the most powerful evaluation tool at a performance engineer's disposal and must be used wisely to pinpoint performance issues. This article expands on discussions of the scatter chart in the "User Experience, Not Metrics" series.
- **Part 7: Identifying the Critical Bottleneck** – One part of the system is always slowest, and until you remedy that bottleneck, no other tuning will actually improve the performance of the application along that usage path. Before you tune it, you must first conclusively identify it.
- **Part 8: Modifying Tests to Focus on Failure/Bottleneck Resolution** – Once a failure or bottleneck is found from a functional perspective, resolution can be reached more quickly if you modify your existing tests to eliminate distraction from ancillary issues.
- **Part 9: Pinpointing the Architectural Tier of the Failure/Bottleneck** – Just because you've identified a bottleneck or failure from a functional perspective and can reproduce it, doesn't mean you know where it is. Pinpointing exactly where the bottleneck is can be an art all its own.
- **Part 10: Creating a Test to Exploit the Failure/Bottleneck** – Now that you know what the bottleneck is functionally and where the bottleneck is architecturally, you will likely need to create a test to exploit the failure/bottleneck in order to help the developer/architect with tuning. This test needn't bear any resemblance to real user activity but rather needs to exploit the issue, and that issue alone. In fact, these scripts often don't even interact with the system in ways that users would and may include direct interaction with back-end tiers. TestStudio can save you significant time and effort in developing such a test.

Advanced Topics

Parts 11 through 14 are "clean-up" topics. These are areas that are referred to in one or more of the previous articles but not done justice due to topic and length constraints. These articles expand on topics that are applicable throughout the engineering process.

- **Part 11: Collaborative Tuning** – Once you can exploit the failure or bottleneck, you need to know how to work with the rest of the team to resolve it.
- **Part 12: Testing and Tuning on Common Tiers** – Web, application, and database servers are common tiers related to testing and tuning. Here we discuss specific issues, methods, and resolutions for these areas.
- **Part 13: Testing and Tuning Load Balancers and Networks** – These components require a significantly different approach from that called for by common tiers. This article discusses issues specific to these areas and approaches for handling them.
- **Part 14: Testing and Tuning Security** – Security is the most performance-intensive activity of an application. Tuning security issues often involves taking a risk-based approach to balancing necessary security with desired performance.

Summing It Up

Our job isn't done when we accurately report the current performance of an application; in fact, it's just begun. The "Beyond Performance Engineering" series is designed to discuss and demonstrate how to take the step from simply reporting results to achieving desired performance with Rational TestStudio and a proven performance engineering methodology. The articles will focus on the topics that are hardest to find information about and point you to where you can find more information. I hope you're looking forward to reading this series as much as I'm looking forward to writing it!

References

- *Grammatical Man: Information, Entropy, Language, and Life* by Jeremy Campbell (Simon & Schuster, 1982)

Acknowledgments

- The original version of this article was written on commission for IBM Rational and can be found on the IBM DeveloperWorks web site

About the Author

Scott Barber is the CTO of PerfTestPlus (www.PerfTestPlus.com) and Co-Founder of the Workshop on Performance and Reliability (WOPR – www.performance-workshop.org). Scott's particular specialties are testing and analyzing performance for complex systems, developing customized testing methodologies, testing embedded systems, testing biometric identification and security systems, group facilitation and authoring instructional or educational materials. In recognition of his standing as a thought leading performance tester, Scott was invited to be a monthly columnist for Software Test and Performance Magazine in addition to his regular contributions to this and other top software testing print and on-line publications, is regularly invited to participate in industry advancing professional workshops and to present at a wide variety of software development and testing venues. His presentations are well received by industry and academic conferences, college classes, local user groups and individual corporations. Scott is active in his personal mission of improving the state of performance testing across the industry by collaborating with other industry authors, thought leaders and expert practitioners as well as volunteering his time to establish and grow industry organizations. His tireless dedication to the advancement of software testing in general and specifically performance testing is often referred to as a hobby in addition to a job due to the enjoyment he gains from his efforts.

About PerfTestPlus

PerfTestPlus was founded on the concept of making software testing industry expertise and thought-leadership available to organizations, large and small, who want to push their testing beyond "state-of-the-practice" to "state-of-the-art." Our founders are dedicated to delivering expert level software-testing-related services in a manner that is both ethical and cost-effective. PerfTestPlus enables individual experts to deliver expert-level services to clients who value true expertise. Rather than trying to find individuals to fit some pre-determined expertise or service offering, PerfTestPlus builds its services around the expertise of its employees. What this means to you is that when you hire an analyst, trainer, mentor or consultant through PerfTestPlus, what you get is someone who is passionate about what you have hired them to do, someone who considers that task to be their specialty, someone who is willing to stake their personal reputation on the quality of their work - not just the reputation of a distant and "faceless" company.