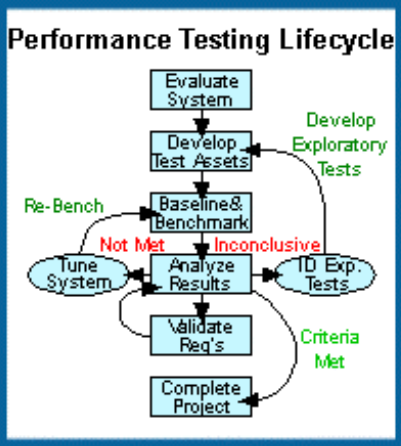
Effective Performance Testing



Part 11: Collaborative Tuning

(This article has been adapted from the made for Rational Developers Network article series *"Beyond Performance Testing"*. The currently completed articles in this series are available in their original form at www.rational.net)

In the course of the previous six articles in this series you've learned how to identify, confirm, exploit, and in some cases even resolve failures, slow spots, and bottlenecks. Traditionally, this is as far as performance testing ever goes, but the title of this series is "Beyond Performance Testing." The step beyond performance testing is performance tuning. Tuning is the process by which all the remaining unresolved failures, slow spots, and bottlenecks are improved, fixed, or mitigated. The performance tester isn't typically involved in this process, but it's my belief that this is a mistake. For this reason, I'm starting another four-article theme intended to help you, the performance tester, become an integral part of what should now be the performance testing and tuning team. This article begins the theme by discussing the advantages of this approach, the roles and contributions of the members of this team during the tuning process, and the ways you can get more involved in a team approach.

This is the second article in the "Beyond Performance Testing" series, which focuses on isolating performance bottlenecks and working collaboratively with the development team to resolve them. If you're new to this series, you may want to begin by reading Part 1the series introduction. This article is intended for all levels of users of the Rational Suite® TestStudio® system testing tool, as well as managers and other members of the development team.

So far, this is what we've covered in this series:

Part 1: Introduction

Part 2: A Performance Engineering Strategy

Part 3: How Fast Is Fast Enough?

Part 4: Accounting for User Abandonment

Part 5: Determining the Root Cause of Script Failures

Part 6: Interpreting Scatter Charts

Part 7: Identifying the Critical Failure or Bottleneck

Part 8: Modifying Tests to Focus on Failure or Bottleneck Resolution

Part 9: Pinpointing the Architectural Tier of the Failure or Bottleneck

Part 10: Creating a Test to Exploit the Failure or Bottleneck

This article is intended for mid- to senior-level performance testers and members of the development team who work closely with performance test engineers. If you haven't read BPT Parts 5, 6, 7, 8, 9, and 10, I suggest you do so before reading this article.

Why Tune Collaboratively?

If you're new to performance testing, you may be surprised to hear that there's often a lot of resistance to the idea of collaborative tuning. A common argument is that there should be a clear division of tasks between performance testing and performance tuning. This stems from the QA-versus-development mindset that prevails in the software development industry as a whole. While I believe this is a problem in general, I think it's an even bigger problem for performance testing than it is for other types of testing because division of tasks decreases efficiency and keeps so many potential gains from being realized. In the following sections, I'll discuss the potential advantages of tuning collaboratively and the disadvantages of not doing so.

Bringing Tester and Developer Mindsets Together

We've known and documented for years that testers and developers think differently. Most of the time this is a good thing, but sometimes it results in an "us versus them" attitude. In general, the difference in mindsets can be summarized as follows:

- Testers tend to look for ways to make the application perform incorrectly in every possible situation.
- Developers tend to try to make the application perform correctly in the situations in which they envision the application being used.

This difference in thought is what leads to developers responding to reported defects with the statement "But no one would ever *really* do that!" Testers know that no matter how unlikely a user action is, *someone* will do it, and that risk needs to be addressed. Often during functional or system testing, this difference leads to confrontations about what should or shouldn't be fixed, with hard feelings resulting on one side or the other based on a manager's decision about whether the defect gets fixed. However, when it comes time to start tuning for performance, you want these two thought processes working together collaboratively, not clashing as adversaries.

The best way to build a collaborative relationship in performance testing is to introduce the performance tester to the development team early in the project as a "resource." Often long before the first build, the developers will have parts of the application built and will want to know how they perform — essentially, they'll want to conduct performance unit testing. This is where the performance tester comes in. The developers typically don't have access to the tools or the tool expertise to conduct these performance unit tests. The performance tester can build these tests on the spot and help resolve many performance issues even before the first official test. In this way, the tester becomes a collaborative part of what will later be the performance-tuning team.

Looking at the Big-Picture View Alongside the Detail View

Developers are focused on the details during performance tuning. This is a good thing, since the performance issues generally live in the details of the application. In contrast, performance testers focus on the big-picture, end-user experience. This is also a good thing, as the end user should always have an advocate in the development process. These differences only become a problem when the two views are considered one at a time and not side by side. For example, a developer may fix a particular symptom (such as slow login) but damage the overall performance of the system (by making every page authenticate instead of downloading the access list during login, for example). With a separation of tasks, the developer may not know that the fix had a significant side effect (making every page except login slower) until the next build is released to the performance tester. Worse, if they're not working together when the performance tester gets the build, the tester will have no idea where to look for potential side effects (that is, may only retest the login page and not know to test all the other pages). If they're working together, the performance tester can build specific tests on the fly to help understand the effects of the changes as they happen. That way the developer can keep the detail focus, the tester can keep the end-user focus, and both can be confident that they're getting quality feedback.

Streamlining the Tuning Cycle

Usually, here's how the tuning cycle goes: the performance tester identifies an issue, the developers run off and fix it, the developers send a new build to the performance tester, the tester finds more side effects than solutions and sends the build back to the developers, and so on. This is all well and good, except that when the performance tester and the developers aren't seen as part of a collaborative testing and tuning team, each step in this cycle often takes days. As a result, it's common for a week to go by between issue identification and side-effect identification.

This might not sound bad to you until I mention that most issues take less than a day to resolve. So where do the other four days go? The following timeline typifies what happens in my experience when tuning isn't a collaborative process:

Day 1 AM — Tester identifies issue.

Day 1 PM — Tester schedules meeting to demonstrate issue to development team.

Day 2 PM — Tester has meeting to demonstrate issue to development team.

Day 3 AM — Developer assigned to resolve the issue, who inevitably wasn't in the meeting, comes to tester's desk for personal demo.

Day 4 AM — Developer believes issue is resolved and requests that a special build be promoted into the test environment.

Day 5 AM — Tester retests application and finds side effects.

Day 5 PM — Tester schedules meeting to discuss side effects.

. . .

Day X — Developer or configuration manager promotes performance-driven modifications to test environment with next scheduled build for testing.

It's obvious how inefficient this process is. When the tester works together with the developers as a collaborative team, the timeline looks more like this:

Day 1 AM — Tester identifies issue and calls developer.

Day 1 AM — Developer comes to tester's desk for personal demo.

Day 2 AM — Developer believes issue is resolved and asks tester to validate this in the development environment.

Day 2 PM — Tester retests application, finds side effects, and calls developer.

Day 2 PM — Developer comes to tester's desk for personal demo.

. . .

Day X — Developer or configuration manager promotes validated performance-driven modifications to test environment with next scheduled build for final validation.

As you can see, this team approach leads to a drastically shortened tuning cycle. It eliminates several intermediate steps that cause unnecessary additional work for individuals who would prefer brief progress updates. The feedback loop is now roughly one day instead of one week long. The value of this time savings is incalculable when you consider that most projects execute their first collective performance test two weeks before "go live" day. With the typical approach, the team gets a chance to fix only one performance issue and one set of side effects. With the collaborative approach, there's the opportunity to address five performance issues and their side effects at no additional cost, all as a result of team building.

The Testing and Tuning Team

I've referred thus far to the performance tester and the developer, but there are several other key players on the performance testing and tuning team. Your team may include slightly different members/titles, but the roles I outline here are common in my experience.

Project Manager

The project manager has overall responsibility for the project, to include development, testing, schedule, budget, and personnel.

Performance-Related Focus:

The project manager is ultimately responsible for the performance of the application as delivered. Sometimes the project manager is the direct supervisor of the performance test engineer, and other times a subordinate of the project manager is the direct supervisor of the performance test engineer. In either case, the project manager should have a direct relationship and establish continuing one-to-one communication with the performance test engineer.

Contribution to Collaborative Tuning:

The project manager has to believe in and promote a collaborative testing and tuning team or it generally won't happen. This person sets the overall tone for the testing and tuning exercise. If the project manager doesn't actively support the collaborative effort, the typical tuning cycle outlined earlier is sure to result.

Lead Developer/Architect

The leader developer/architect is generally the top technical person on the project, with overall responsibility for the development effort. In most cases, this individual reports to the project manager. The other developers, system administrators, database administrators (DBAs), and so on report to this person. Occasionally the performance test engineer will also report to the lead developer.

Performance-Related Focus:

The lead developer is ultimately responsible for delivering the application with the required performance. This person is also responsible for ensuring that the development team and the performance test engineer communicate effectively.

Contribution to Collaborative Tuning:
The lead developer/architect typically assigns tasks to the other developers, so it's up to this person to make performance testing and tuning a priority. If this person doesn't give the appropriate developers the time and freedom to participate in the testing and tuning activity, this activity will usually end up getting overlooked. The lead developer/architect is also generally the individual who must be convinced to bring in outside experts when needed.

Other Developers/Administrators/DBAs

Other developers, system administrators, and DBAs are generally responsible for developing, creating, and/or maintaining one specific component of the application. They generally report to the lead developer/architect, at least for their tasks related to the project at hand. I'll use the term *developers* to refer to members of this group as a whole. The developers generally begin a project assuming that the performance test engineer is "just a tester." Until you establish yourself as a member of their team for this part of the project, they probably won't work with you very much.

Performance-Related Focus:

Each member of this group is responsible for the performance of her or his particular focus area, and ideally, the performance of the integration with other developers' focus areas.

Contribution to Collaborative Tuning:

These individuals are the ones who actually tune the application to meet the performance goals. They're also the ones who are best served by having personal relationships and tight feedback loops with the performance test engineer. They're at the core of the collaborative testing and tuning team.

Test Manager

The test manager is responsible for managing the testing effort as a whole and reports to the project manager. In many cases, the testers, business analysts, and configuration managers report to the test manager, who reports to the project manager.

Performance-Related Focus:

In most cases, the test manager serves as the supervisor of the performance test engineer and is therefore responsible for the overall performance-testing effort. In my opinion, this often causes a conflict of interest. Many of the test managers I've worked for over the years have been very focused on the typical tuning approach outlined earlier, demanding that all performance results go through them to the project manager, who then passes the information to the lead architect to assign a developer to the reported issue. In this scenario, the reverse is often true as well. When the developer requests a specific test to validate performance, the request has to go through the lead architect to the project manager and then to the test manager before it reaches the performance test engineer.

In my experience, many test managers find it difficult to adopt the collaborative tuning approach. For this reason, I recommend sitting with the test manager, the project manager, and the lead developer/architect long before performance testing actually starts, in order to define roles.

Contribution to Collaborative Tuning:

In the best case, the test manager is the first-level manager who helps get support and resources for the performance test engineer. In most cases, this person serves as the process, schedule, and documentation enforcer.

Performance Test Engineer

By this point in the series, I think you've got a pretty clear picture of the performance test engineer's role. In summary, this is typically one person who's responsible for the following:

- developing the overall performance test strategy
- collecting and quantifying the performance requirements
- determining and documenting the user community model(s)
- creating scripts representing the user community model(s)
- executing the scripts and analyzing the results
- working with the developers as part of the collaborative testing and tuning team

Performance-Related Focus:

The focus of the performance test engineer is simply to design, develop, execute, analyze, and report on the results of performance tests that accurately represent conditions likely to occur in production, and to assist in the tuning effort when these results fall short of requirements and/or expectations.

Contribution to Collaborative Tuning:

The chief contribution of the performance test engineer consists of the tests themselves, the interpreted results of those tests, and recommendations based on those results.

Outside Experts

Outside experts obviously aren't a permanent part of the team. Sometimes, however, a performance issue is uncovered that no permanent member of the team has the skills to resolve. Outside experts are typically treated as consultants to the project, even if they're employees of the company.

Performance-Related Focus:

Outside experts are focused on the very specific issue they were retained to resolve. They tend to be retained to configure and tune new third-party software or hardware.

Contribution to Collaborative Tuning:

The outside expert very simply contributes a performance improvement to a specific issue detected by the testing and tuning team that they were unable to resolve themselves.

How to Get More Involved

Unless you're extremely lucky, you're not currently part of a well-formed testing and tuning team. If you're not, the following suggestions will help you become a part of, or develop, that team. If you're lucky and are already a member of such a team, you may want to take personal inventory and see if you could make improvements in any of these areas.

Know the Technologies

The single most important thing you can do to gain the respect of the development staff and start building a collaborative environment is to know about the technologies involved in the application you're working on. Being able to participate in technical discussions and ask intelligent questions will immediately promote you from "just a tester" to a "techy." For example, I became a "techy" about three weeks into a recent project when I asked, "Does it really add a lot of value to SSL-enable this application when it will be deployed on a secure network and all the data transmitted across the network will be in compressed format?"

The question led to a lengthy discussion about performance versus real security versus policy. The outcome of the discussion wasn't particularly important. What was important was that I presented a technical issue intelligently and was able to support my position and generally contribute to the technical aspects of the conversation. That was the point in that project when I ceased being "just a tester."

Attend Meetings

As you might imagine, I would never have had the opportunity to participate in the discussion I just described if I didn't attend meetings, particularly design meetings. If you're thinking, "But meetings are a waste of time!" I generally agree with you. The key is to attend the right meetings. I try to avoid administrative meetings if at all possible. However, I always try to attend meetings where the developers will be. I've found that I can learn more about the application I'm testing during an hour-long architectural review meeting than I can in an hour of reading documentation or exploration. As a side note, when you do attend meetings, make sure that you participate in the meeting. Attending as a bump on a log doesn't help establish you as a "techy."

Educate Your Team

The next most important thing you can do is educate your entire project team about performance testing. Most folks involved in the software development process know very little about performance testing either conceptually or technically. I have a series of one-hour workshops that I like to do for a new project team. I find that the more of these I'm able to present, the more collaborative the effort becomes. Here are the titles of those workshops:

- "Performance Engineering Overview" (general audience)
- "Collecting and Quantifying Performance Requirements" (general audience)
- "Determining the User Community Model(s) to Be Tested" (general audience)
- "Understanding Protocols and Script Creation" (technical audience)
- "Interpreting Basic Reports" (general audience)
- "Collecting and Analyzing Performance Data" (technical audience)
- "Performance Unit Testing" (technical audience)

While all of these topics are important and valuable, the ones that pay the largest dividends are "Understanding Protocols and Script Creation" and "Collecting and Analyzing Performance Data." In my experience, the vast majority of developers come to these sessions thinking they already know what I'm going to say, and leave saying, "Wow! This is really cool! I've got a bunch of ideas on how this can help me. When do you have time to talk about it with me?" As soon as I hear the words "Wow, this is really cool!" I know I've laid the groundwork for a collaborative testing and tuning team.

Ask Questions

I've mentioned it briefly in other articles but I want to stress it here: Ask smart questions about things you don't know. A healthy thirst for knowledge will establish you as a person who's interested in the details of the application, not just the surface-level application interface. Many developers assume that testers don't understand or care about how the application works as long as it passes the documented test cases. It's critically important to demonstrate that as a performance test engineer you're interested in knowing and capable of understanding how the application works.

Offer Assistance

You have skills and tools at your disposal that the development team doesn't have and that can assist them in a variety of ways, some of which you'll probably never even think of. Regardless of who your direct supervisor is, you should make your skills and tools available to the development team to assist them. Sometimes, you can record a script in a few minutes that can provide feedback that would otherwise take them hours, days, or longer to collect.

One example is load balancer configuration testing. If you have an environment available to generate load from multiple IPs, in a matter of minutes you can create a script that will validate the configuration of the load balancer. Without that script the only other way to test that configuration may involve a large number of coordinated manual users in a test that could never be repeated exactly.

Be Available and Approachable

As a performance test engineer, you probably work in a room by yourself with a whole bunch of computers in it on the other side of the building from the rest of your project team. While this is a blessing in some ways, it tends to lead to your being the person who's "out of sight, out of mind." Don't allow that to happen. Once people start forgetting that you're part of the project team, you can't be effective there. Even if you're in a different part of the building or in a different building entirely, ensure that you're available and approachable. If you're doing all of the things I've discussed in this article, you're already doing this. I list this suggestion separately mostly to remind you not to allow yourself to become isolated.

Tear Down the QA-versus-Development Barrier

All in all, you're trying to tear down the perceived barrier between QA and development. This barrier has been built up in most organizations over a long period of time for a variety of reasons, not the least of which is that the two groups have very different skill sets. As a performance test engineer, you probably have a skill set that's a hybrid of those skills typically associated with each group. The bottom line is that if you don't work with both groups, you'll never be able to do your whole job. It's up to you to work on both sides of this barrier and deal with any side effects of not fully aligning with one group or the other. Don't expect someone else to do it for you.

Summing It Up

Collaborative testing and tuning teams are very powerful when it comes to detecting and resolving performance-related issues. However, these types of teams are still rare and often resisted. By involving yourself with the developers and honing your technical skills, you should be able to establish this kind of collaboration and ultimately realize the value of this team approach. In the next three articles, we'll look at specific things you can do to assist with testing and tuning in common areas of poor performance.

About the Author

Scott Barber is a System Test Engineer and Quality Assurance Manager for AuthenTec, Inc. and a member of the Technical Advisory Board for Stanley-Reid Consulting, Inc. With a background in network architecture, systems design, database design and administration, programming, and management, Scott has become a recognized thought leader in the context-driven school of the software testing industry. Before joining AuthenTec, he was a consultant specializing in Performance Testing/Analysis, a Company Commander in the United States Army, a DBA and a Government Contractor in the transportation industry.

Scott is a co-founder of WOPR (the Workshop on Performance and Reliability), a semi-annual gathering of performance testing experts from around the world, a member of the Context-Driven School of Software Testing and a signatory of the Agile Manifesto. He is a Discussion Facilitator in the Rational Developer Network public forums and a moderator for the Performance Testing and Rational TestStudio related forums on QAForums.com. Scott's Web site complements this series. Please visit it to find more detail on some topics and view slides from various presentations he's given recently. You can address questions/comments to him on either forum or contact him directly via e-mail.

AuthenTec, Inc. is a leading semiconductor company providing advanced biometric fingerprint sensors to the PC, wireless, PDA, access control and automotive markets. AuthenTec's FingerLoc and EntréPad product families utilize the Company's patented TruePrint™ technology, the first technology capable of imaging everyone under virtually any condition.

Stanley Reid Consulting, Inc. is a small, niche consulting company that focuses on IT organizational improvement and recruiting and staffing of technical experts. Stanley Reid Consulting helps your team achieve consistent, successful delivery of IT projects through the following services: IT Organizational Improvement Consulting, Expert Supplemental Staffing, Technical Recruiting & Placement.