PerfTestPlus

Better Testing... Better Results

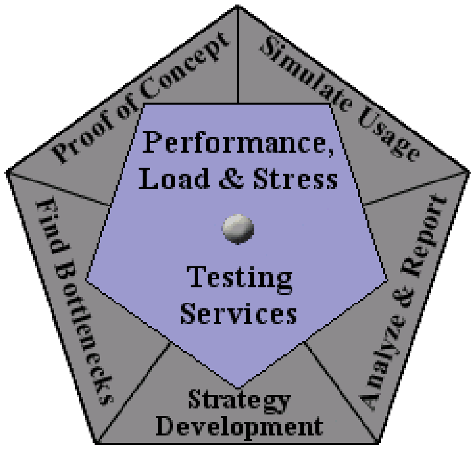
BPT Part 2: A Performance Engineering Strategy

Without a strategy, performance engineering is simply an exercise in trial and error. Following a sound strategy in the engineering effort will increase your performance engineering team's efficiency and effectiveness. This article outlines a strategy that complements the Rational Unified Process® approach and is easily customizable to your project and organization, and that's been validated by numerous clients worldwide. The templates I provide will give you a starting point for documenting your performance engineering engagement. Applying this strategy, coupled with your own experience, should significantly improve your overall effectiveness as a performance engineer.

Beyond Performance Testing

by:

R. Scott Barber



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 1 the 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.

A Closer Look at Performance Engineering

As defined in Part 1, performance engineering is the process by which software is tested and tuned with the intent of realizing the required performance. Let's look more closely at this process. In the simplest terms, this approach can be described as shown in Figure 1.

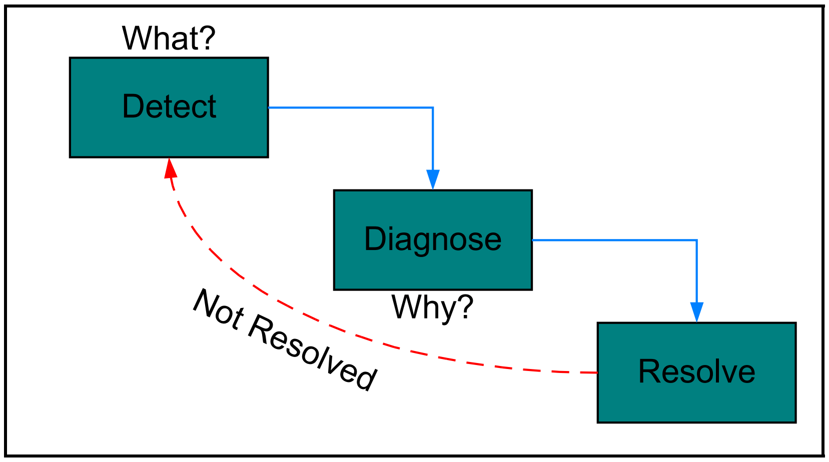


Figure 1: Performance engineering in its simplest terms

I've seen this chart, or a similar one, in many software performance presentations and seminars. Although this chart makes great common sense, it doesn't shed much light on what we really want to discuss here, which is "How, exactly, do I detect, diagnose, and resolve?" Figure 2 gives a much more detailed picture of the various aspects of the process.

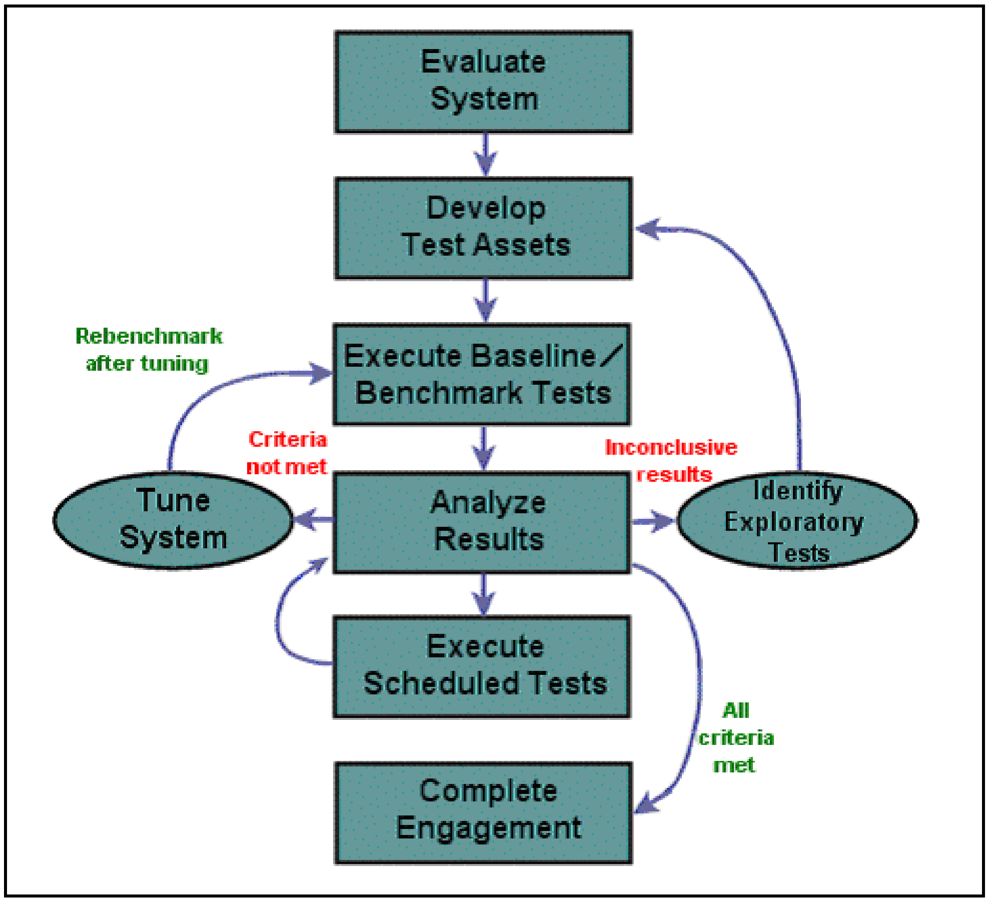


Figure 2: Aspects of the performance engineering process

The strategy detailed in Figure 2 has been applied successfully in many performance engineering projects and has been adopted successfully internationally. Following is a short overview of each of the eight major aspects of this performance engineering strategy, indicating where else in this "Beyond Performance Testing" (BPT) series or the previous "User Experience, Not Metrics" (UENM) series more information on that aspect can be found. My Web site (www.perftestplus.com) gives a full description of each of these aspects and their subcomponents, as well.

Please note that while many people would refer to these aspects as *phases*, I'm using the word *aspect* here intentionally to make a distinction. The word *phase* implies a certain sequence. While some of the aspects of the performance engineering process are performed in order, many of them are completed in a very fluid manner throughout the project. Don't think of this as a step-by-step approach, then, but rather as a list of things to consider.

Evaluate System

Evaluation of the system under test is critical to a successful performance testing or engineering effort. The measurements gathered during later aspects are only as accurate as the models that are developed and validated in this aspect. The evaluation also needs to define acceptable performance; specify performance requirements of the software, system, or component(s); and identify any risks to the effort before testing even begins.

Evaluating the system includes but isn't limited to the following steps:

- determine all requirements related to system performance (BPT Part 3)
- determine all expected user activity, individually and collectively (UENM Part 2, Part 3, Part 4)
- develop a reasonable understanding of potential user activity beyond what's expected (UENM Part 2, Part 3, Part 4)
- develop an exact model of both the test and production architecture
- identify and schedule all non-user-initiated (batch) processes (UENM Part 2, Part 3, Part 4)
- develop a reasonable model of actual user environments
- identify any other process/systems using the architecture
- define all system/component requirements in testable terms (BPT Part 3)
- define expected behavior during unexpected circumstances (BPT Part 3)

As performance engineers, we need to become intimate with the core functions of the system under test. Once we know and understand those functions, we can guide the client to develop performance acceptance criteria as well as the user community models that will be used to assess the application's success in meeting the acceptance criteria.

Develop Test Assets

A test asset is a piece of information that will remain at the completion of a performance engineering project. Some people refer to these items as "artifacts." These assets are developed during this aspect of the process:

- Performance Engineering Strategy document (discussed later in this article)
- Risk Mitigation Plan (discussed later in this article)
- automated test scripts (referenced throughout both series)

The Develop Test Assets aspect begins before performance testing is scheduled to start. The Performance Engineering Strategy document and the Risk Mitigation Plan can be started immediately upon completion of the Evaluate System aspect. Automated test script development can begin only after development of a stand-alone component or when the entire application is believed to be stable and has undergone at least initial functional testing.

This aspect concludes when:

- the Performance Engineering Strategy document has been completed and approved by the stakeholders,
- mitigation strategies have been defined for all known risks, and
- all load-generation scripts have been created and individually tested (for the "testable" sections of the application).

Execute Baseline/Benchmark Tests

The Execute Baseline/Benchmark Tests aspect is where test execution actually begins. The intention here is twofold:

- All scripts need to be executed, validated, and debugged (if necessary) collectively (as they've already been individually to move beyond the Develop Test Assets aspect).
- Baseline and benchmark tests need to be conducted to provide a basis of comparison for all future testing.

Initial baselines and benchmarks are taken as soon as the test environment is available after the necessary test assets have been developed. Re-benchmarking occurs at the completion of every successful execution of the Tune System aspect. Designed exploratory scripts are baselined and executed at volume if necessary during this aspect.

It's important to analyze the results of baseline and benchmark tests. While the methodology we're discussing makes this clear, many people I've talked to don't fully appreciate the necessity of analyzing the results of these early low-volume tests. It's our responsibility as performance engineers to ensure that this analysis isn't left out.

Analyze Results

Analysis of test results is both the most important and the most difficult part of performance engineering. Proper design and execution of tests as well as proper measurement of system and/or component activities make the analysis easier. Analysis should identify which requirements are being met, which ones aren't, and why. When the analysis shows why systems or components aren't meeting requirements, then the system or component can be tuned to meet those requirements.

Analysis of results may answer the following questions (and more):

- Are user expectations being met at various user loads? (BPT Part 3, Part 5, Part 6)
- Do all components perform as expected under load? (BPT Part 3, Part 5, Part 6)
- What components cause bottlenecks? (BPT Part 6, Part 7, Part 9)
- What components need to be or can be tuned? (BPT Part 11)
- Do additional tests need to be developed to determine the exact cause of a bottleneck? (BPT Part 8, Part 10)
- Are databases and/or servers adequate? (BPT Part 12)

- Are load balancers functioning as expected? (BPT Part 13)
- Is the network adequate? (BPT Part 13)

The Analyze Results aspect focuses on determining if the performance acceptance criteria have been met, and if not, what the bottlenecks are and whose responsibility it is to fix those bottlenecks. This aspect involves close coordination with stakeholders to ensure that both the performance engineering team and stakeholders agree that all requirements are being validated. System administrators may also be involved in results analysis. Keeping a record of the tests being analyzed and the results of that analysis is an important part of this activity.

Execute Scheduled Tests

Scheduled tests are those that are identified in the Performance Engineering Strategy document to validate the collected performance requirements. Scheduled tests shouldn't be conducted until baseline and benchmark tests are shown to meet the related performance requirements.

There are countless types of measurements that can be gathered during scheduled tests. Requirements, analysis, and design will dictate what measurements will be collected and later analyzed. Also, required measurements may change throughout the course of testing based on the results of previous tests. Measurements collected during this activity may include but aren't limited to the following:

- end-to-end system response time (user experience) (UENM Part 5, Part 8)
- transactions per second for various components
- memory usage of various components by scenario
- CPU usage of various components by scenario
- component throughput
- component bandwidth

Other measurements as requested by stakeholders may also be included. Applications or environments with back-end processes that aren't directly triggered by user activity usually require transactions-per-second measurements to be collected.

Measurements collected here will be compared to measurements collected during baseline and benchmark testing. Multiuser tests will be executed to determine actual current performance, find "knees" in performance (places where performance degrades dramatically rather than smoothly; see UENM Part 10), and determine bottlenecks (BPT Parts 6—10). Exploratory tests may need to be developed and executed to help find or tune bottlenecks based on analysis of the measurements collected at this time (BPT Part 10).

Identify Exploratory Tests

This is the aspect of performance engineering in which unplanned tests are identified to detect and exploit performance issues are developed to aid in the tuning effort. To be effective, these tests must be researched in collaboration with the technical stakeholders who have intimate knowledge of the area of the system exhibiting performance issues. The results of this research lead the project back into the Develop Test Assets aspect, where exploratory tests are created and documented.

Exploratory tests are designed to exploit the specific system area or function suspected to contain a performance bottleneck based on previous results analysis. Typically, the suspect tier or component of the bottleneck is identified and then decisions are made about the metrics that need to be collected to determine if the bottleneck does, in fact, reside in that area, and to better understand the bottleneck. Finally, the type of test that's required is identified and described so that it can be developed. We'll discuss this in detail in Part 10 of this series.

Tune System

Tuning must occur at a component level while keeping the end-to-end system in mind. Even tuning each component to its best possible performance won't guarantee the best possible overall system performance under the expected user load. After tuning a component, it's important not only to retest that component but also to re-benchmark the entire system. Resolving one bottleneck may simply uncover another when system wide tests are re-executed.

The Tune System aspect may address but isn't limited to the following topics, all of which are explored in more detail in BPT Part 11 and some in Part 12 or Part 13:

- Web server configuration
- database design and configuration
- application or file server configuration
- cluster management
- network components
- server hardware adequacy
- batch process scheduling/concurrency
- load balancer configuration
- firewall or proxy server efficiency

This aspect of the project is a highly collaborative effort involving the performance engineering team and the development team. Often, once tuning begins the performance engineer must be available to retest and analyze the effects of the changes made by the developer before any other activity can occur. The information gained from that analysis is critical to the developer or system administrator who's making the actual changes to the system. It's very rare for the performance engineering team to make actual changes to the system on their own. The activities associated with tuning the system need to be at least loosely documented so that differences from the original design can be captured and lessons can be passed on to future developers and performance engineers.

Complete Engagement

Documentation is created primarily to be used as a historical reference and as validation of requirements being met. For applications that are likely to have future releases, upgrades, or increased future load, it's important to document the capabilities of the system, known bottlenecks, and areas where most improvement can be realized in the future. The format of the documentation should be agreed upon during the Develop Test Assets aspect.

The results document, which is discussed in detail below, is specifically geared to show stakeholders whether the system under test meets the performance acceptance criteria. If the criteria haven't been met, the document should explain why not, particularly if significant tuning or upgrading of the system, which may fall outside the scope of the project, is required. The document should also identify areas of future improvement if bottlenecks are detected but not resolved.

The Performance Engineering Strategy Document

Since I started writing articles and moderating forums, one of the most common questions I've been asked is "Where can I get a performance test plan template?" As much as we may want to deny the fact, we have to concede that it's important to document our projects. During my tenure as a performance engineer, with the input of countless clients, friends, and coworkers, I've developed a document template based on the process I've just outlined. I call the resulting document the Performance Engineering Strategy. You can download a .pdf version of the template if you want to use it.

Why, you may wonder, do I call this an engineering strategy rather than a test plan? It's my opinion that while these two things fundamentally serve the same purpose, they're quite different documents. A functional test plan tells the who, what, when, why, and how of the functional testing effort. It lists specific tasks assigned to specific people to be completed by specific dates. In our performance engineering strategy, on the other hand, once we execute our first test we reach a set of decision points. It's simply not possible to put a predetermined structure around activities such as "tune system" or "identify and develop exploratory tests." We don't even know if any tuning will be required, or how many exploratory tests may ultimately be developed. How can we assign the optional activity of "tune system" to a person when we don't yet know what will need to be tuned? Obviously, we can't.

What we *can* do, however, is explicitly detail a strategy to address the question "What do we do when…?" So I like to make the distinction between a test plan and an engineering strategy up front. While it's possible to build a performance test plan, I've found that it becomes more of a hindrance than a help by the conclusion of the first battery of executed tests. I find it more useful to have a document that outlines the strategy, and then when a performance issue presents itself, to create a "mini-plan" for resolving that performance issue that's consistent with the overall strategy.

Let's discuss the basics of this strategy document. The template I've provided for you to download includes example verbiage in the sections that are unique to the application under test, and since we'll be discussing some of the sections in more detail in other articles, I won't go into too much detail here. Instead, I'll outline the document and describe what I recommend including in each section.

1 Introduction

1.1 Description

Describes the document, not the performance engineering effort, to let the reader know what to expect.

1.2 . Purpose
Gives a high-level overview of the document's purpose.

1.3 Scope
Is similar to the purpose statement but focuses on boundaries and what's not covered in the document rather than what *is* covered.

1.4 Related Documents
Lists other documents that provide information referenced in this document and may also list project documents that aren't directly referenced but could be valuable to the readers of this document.

2 . Performance Acceptance Criteria

2.1 Introduction
Gives a brief definition of what we mean by performance acceptance criteria.

2.2 Performance Criteria
Defines the specific types of performance-related criteria being used for the engagement.

2.3 Requirements
Details those performance criteria that must be satisfied at a minimum in order for the application to be put into production.

2.4 Goals
Details those performance criteria that would ideally be satisfied when the application is put into production. These are always more stringent than the criteria listed in "Requirements."

2.5 Engagement Complete Criteria
Defines what it will mean to be done with the engagement. Assuming all criteria and/or goals can't be achieved, details how the performance engineering engagement will conclude.

3 Workload Distribution

3.1 Introduction
Describes what a workload distribution is and how it relates to performance engineering.

3.2 Workload Distribution for <application>
Details the workload distribution(s) and/or user community model(s) to be simulated during the performance engineering engagement.

4 Script Descriptions

4.1 Introduction
This introduction is very important for stakeholders who don't understand automated load-generation tools and must be customized for your particular audience. It should describe how all the tools you'll be using work, how the application will be scripted, and how this relates to the workload distribution(s) described in Section 3 of the document. This may also be a good place to describe how measurements will be collected and how that relates to

scripts.

4.2 <Script Name 1> Script Overview
Discusses what the script does and how it relates to the workload distribution.

4.2.1 <Script Name 1> Script Measurements
Discusses what measurements will be collected by the script and what measurements may be collected by other means while that script is executing.

4.2.2 <Script Name 1> Script Think Times
Discusses what the delay times and distributions are for the pages included in the script, and how those times were determined.

4.2.3 <Script Name 1> Script Data
Discusses what data will be used/required for this script to simulate real application usage, such as unique IDs and passwords for each simulated user, or "test" credit card information. If this data doesn't already exist, also describes how this data will be obtained.

4.2.4 <Script Name 2 etc.> Script Overview

5 Test Execution Plan
If you adopt the methodology we're discussing in this article, this part of the template won't change. Rather than duplicating what's written there, I'll simply include the outline here and let you review the template for more detail.

5.1 Introduction

5.2 Evaluate System

5.3 Develop Test Assets

5.4 Execute Baseline/Benchmark Tests

5.5 Analyze Results

5.6 Execute Scheduled Tests

5.7 Identify Exploratory (Specialty) Tests

5.8 Tune System

5.9 Project Closeout

6 Risks
Since this is sometimes an entirely separate document, I'll discuss it separately below and therefore have left just the basic outline here for completeness.

6.1 Introduction

6.2 <Risk 1>

6.3 <Risk 2 etc.>

That's the outline of the Performance Engineering Strategy document, then. Now I'll describe the Risk Mitigation Plan, which can be either Section 6 of the strategy document or a separate document.

The Risk Mitigation Plan

Risk identification and mitigation are critical to any project, and performance engineering is no exception. My experience has shown that it's absolutely imperative to publicly identify risks to performance engineering efforts as soon as they present themselves. Note that I'm not talking about the high-level risks that are managed by the project manager, such as "The application may not scale to the proper number of users." That type of risk should be identified in the overall project plan. The kind of risk I'm talking about is more along the lines of "The performance test environment may be late, thus eliminating some of the time dedicated to performance testing prior to 'go live'."

Risks like these need to be raised and documented so that at the end of the project there will be a history of all the identified risks, their potential impact, the mitigation strategy, and the resolution. I like to document these risks in the Performance Engineering Strategy document but some organizations prefer that this be a separate document. Either way, I've found the following format to be easy to use for tracking risks:

- **Risk name** – Give each identified risk a descriptive name that stakeholders will immediately recognize. This will ensure that the rest of the discussion about that risk gets reviewed.
- **Discussion** – The discussion of the specific risk and its potential impacts needs to be very detailed. The discussion doesn't pass judgment but simply states facts. Quantifiable facts are best.
- **Mitigation strategy** – The mitigation strategy includes two parts: (1) How are we going to try to keep this risk from happening? (2) If this risk does happen, how do we minimize its impact? The more detailed the plan, the better. The mitigation strategy is the most important part of risk management.
- **Owner** – Each risk should have an owner, preferably an individual rather than a group or an organization. The owner of the risk isn't necessarily responsible for taking all of the action related to that risk but rather is responsible for ensuring that any required action is accomplished by the right people at the right time.
- **Status** – Because this is a living document and should be updated no less often than weekly, a current status should always be included.

It's beyond the scope of this article to discuss formal risk mitigation techniques, and there are plenty of quality books and resources available on this topic that go far deeper than I could in one small section of an article. The point I want to make here is that risks associated with a performance engineering engagement should be managed and documented independently of general project risks.

The Performance Engineering Results Document

One of the most often overlooked aspects of a performance engineering engagement is the documentation of results. What typically happens is that performance testing falls behind, testing continues frantically until "go live" day, the application goes live, it doesn't crash, and everyone forgets about performance. Then a couple of months later someone finds a performance problem and asks, "Did we find this during testing? Does anyone know how to fix this? Don't we have scripts to help isolate the problem?" And no one can answer these questions.

Why does this happen? Because we didn't document the results. And now, not only is it our fault that performance "suddenly" got bad, but we're also being accused of not really doing a good job of testing in the first place!

There's one simple way to fix this – by compiling a Performance Engineering Results document. I've created a template for this document that I'll outline below. You can download a .pdf version of the template if you want to use it. You'll notice that this document duplicates much of the information in the strategy document. Experience shows that stakeholders like to have all of this information in one place, rather than having to go back and forth between two documents. I recommend that you discuss this format with your stakeholders to ensure that it meets their needs before starting the document.

1. Executive Summary

This one-page summary of the results should provide the information that a high-level stakeholder needs to make a "go live" decision about the application. Focus is on the actual performance of the application at the time of the final test and may include recommendations if appropriate.

2. Introduction

2.1. Scope
Is similar to the purpose statement but focuses on boundaries and what's not covered in the document rather than what *is* covered.

2.2. Purpose
Gives a high-level overview of the document's purpose.

2.3. Related Documents
Lists other documents that provide information referenced in this document and may also list project documents that aren't directly referenced but could be valuable to the readers of this document.

3. Performance Acceptance Criteria

3.1. Introduction
Gives a brief definition of what we mean by performance acceptance criteria.

3.2. Performance Criteria
Defines the specific types of performance-related criteria being used for the engagement.

3.2.1.Requirements
Details those performance criteria that must be satisfied at a minimum in order for the application to be put into production.

3.2.2.Goals
Details those performance criteria that would ideally be satisfied when the application is put into production. These are always more stringent than the criteria listed in "Requirements."

3.3. Engagement Complete Criteria
Defines what it will mean to be done with the engagement. Assuming all criteria and/or goals can't be achieved, details how the performance engineering engagement will conclude.

4. Workload Distribution

4.1. Introduction
Describes what a workload distribution is and how it relates to performance engineering.

4.2. Workload Distribution for <application>
Details the workload distribution(s) and/or user community model(s) to be simulated during the performance engineering engagement.

5. Baseline Results

5.1.Introduction
Describes the baseline tests as they were actually conducted, in detail.

5.2. System Architecture
Describes the environment that the baseline tests were conducted against.

5.3. Baseline Results
Summarizes results. Supporting data can be included in an appendix as appropriate.

6. Benchmark Results

6.1. Introduction
Describes the benchmark tests as they were actually conducted, in detail.

6.1.1. System Architecture
Describes the environment that the benchmark tests were conducted against.

6.2. Benchmark Results
Summarizes results. Supporting data can be included in an appendix as appropriate.

6.2.1. Benchmark Results <test1 etc.>
Briefly summarizes any points of interest for specific benchmark test executions.

7. Other Scheduled Test Results
Follows the same format as used for baselines and benchmarks for each of the types of tests conducted.

7.1. Scheduled Tests

7.1.1. User Experience Tests

7.1.2. User Experience Test Results

7.1.3. Common Tasks Tests

7.1.4. Remote Location Tests

7.1.5. Stability Tests

7.1.6. Batch Baselines

7.1.7. Production Validation Tests

7.2. Exploratory (Specialty) Tests

7.2.1. Concern/Issue 1

7.2.2.Concern/Issue 2

8. Conclusions and Recommendations

8.1. Consolidated Results
Contains charts and narratives that summarize the overall results, as described in UENM Parts 8—10. This is more detailed than the executive summary but still doesn't include the supporting data.

8.2. Tuning Summary
Summarizes the performance bottlenecks that were found and how they were resolved. It's not necessary to give a complete list of all the activities leading to detecting the bottleneck and the resolution.

8.3. Conclusions
Is an expanded version of the executive summary, providing more detail for an audience of stakeholders rather than technical team members.

8.4. Recommendations
Is a discussion of all of the performance test team's recommendations, not just a yes/no on the "go live" decision. Often includes recommendations for future testing and tuning of the application, and insight into capacity and scalability planning.

Summing It Up

This article has laid the groundwork for the rest of this series by describing the approach we'll follow. I've outlined the eight aspects of a performance engineering strategy that will enable you to detect, diagnose, and resolve performance problems in the applications you test. I've also provided templates for two essential documents, the Performance Engineering Strategy document and the Performance Engineering Results document, and outlined a Risk Mitigation Plan that may or may not be part of the strategy document. Now we're ready to tackle the topic of performance-related requirements in the next article.

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.