{YOUR LOGO} {Client Logo}

PERFORMANCE ENGINEERING STRATEGY

Prepared By	{Name}, {Company}
Date	{Date}
Document No.	{Doc #}
Version	{version}
Status	{status}
Next Step	{Notes here}

Revision History

Date	Version	Description	Author

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Description

This Performance Engineering Strategy document defines the approach to testing the {*name*} system. It briefly describes the methods and tools used by <*Performance Engineer(s)*> to validate {*and/or tune*} the performance of the system.

1.2 Purpose

The purpose of this document is to outline the approach that the Performance Engineering team will take to ensure that the Performance Acceptance Criteria is met. Specifically, this document details the:

- Performance Acceptance Criteria
- Workload Distribution to be used to exercise and gather measurements from the application
- Performance Testing schedule
- Tests to be performed
- Measurements to be collected
- User patterns to be modeled to include user think times
- Data and data management issues

1.3 Scope

This document will provide a strategy to carry out all Performance Engineering activities for the {name} system. It will briefly discuss the resources required, including the toolset used to accomplish test execution, results analysis and application tuning. It will cover the Performance Acceptance Criteria, explain the user community models to be tested, and describe the scripts and schedules to be developed in a narrative fashion.

This document is only intended to address {list releases or builds – may be omitted of testing final release}, although it is strongly recommended that this strategy be followed for all future releases.

This strategy doesn't include functional testing, nor does it guarantee any specific performance results.

The strategy defined in this document may be used for both scheduled and interim releases, however, interim releases may not allow for complete adherence to this test strategy. The approach used for interim releases will be dependent upon both the criticality of the release and the completeness of the functionality included in the release.

The primary objectives for this testing effort are to:

- Validate that the Performance Acceptance Criteria are met by the system AND/OR
- Identify and ensure that performance related defects are addressed prior to deployment.

1.4 Related Documents

The following documents are referenced in this document.

Ref.	Name (Document No.)	Date
1.	<*Organization*> Performance Engineering Methodology	
2.	Architecture doc (doc #)	
3.	Project plan (if applicable)	
4.	Requirements doc (if applicable)	
5.	SOW (if applicable)	

2 PERFORMANCE ACCEPTANCE CRITERIA

2.1 Introduction

Performance efforts always have two sets of criteria associated with them. The first are performance criteria (requirements and goals), and the second are engagement completion criteria. In the sections below, both types of criteria are explained in general and in specific detail for the <*application*> performance engineering effort. The performance effort will be deemed complete when either all of the performance criteria are met, or any one of the engagement completion criteria is met.

2.2 Performance Criteria

Performance criteria are the specific target performance requirements and goals of the system under test. In the case of the <*application*>, <*Performance Team*> and <*stakeholders*> have worked collaboratively through mutual experience, conversations and workshops to develop the criteria enumerated below. The preferred result by both <*Performance Team* > and < *stakeholders*> of the performance engineering effort is to validate that the application meets all of these goals and requirements currently and/or tune the application until these goals are met. If this is not possible, at least one of the engagement completion criteria from the next section must be met for overall performance acceptance.

<The following examples indicate the type of acceptance criteria that should be listed in this section and at what level of detail it should be written. The actual performance acceptance criteria for the project should be included in the format of the examples seen here:

example 1

2.2.1 Requirements

Requirements are those criteria that must be met for the application to "go live" and become a production system.

2.2.1.1 General Requirements

1. *System stable and responsive with 250 hourly users accessing <system> via Intranet in accordance with the User Community model.*
2. *<system> exhibits not more than a 10 second response time (via Intranet) 90% of the time under a 250 hourly user load with less than 5% abandonment.*
3. *<system> stable and responsive under spike loads.*
4. *<system> stable and responsive under stress load.*
5. *<system> stable and responsive under expected load during scheduled maintenance/batch processing.*

2.2.1.2 Time Sensitive/High Profile Activity Requirements

1. *Check Out Item - 3 seconds*
2. *Check In Item - 3 seconds*

2.2.2 Goals

Goals are those criteria that are desired for the application which are in some way different than the previously stated requirements. Testing and tuning will continue until either all goals are met, or until time/money is at an end and the Requirements are met.

2.2.2.1 General Goals

1. *System stable and responsive with 500 hourly users accessing <system> via Intranet in accordance with the User Community model.*
2. *<system> exhibits not more than a 5 second response time (via Intranet) 85% of the time and not more than an 8 second response time 95% of the time under a 250 hourly user load with less than 5% abandonment.*
3. *All field validations exhibit not more than a 3 second response time (via Intranet) 95% of the time under maximum expected user loads.*

2.2.2.2 Time Sensitive/High Profile Activity Goals

1. *Check Out Item – 1.5 seconds*
2. *Check In Item – 1.5 seconds*

example 2

The objectives of the Performance Engineering Effort are:

- *To validate the scalability of the technical architecture and operability on a shared platform (up to 1000 concurrent users).*
- *To validate system performance of:*
  - *All user actions that require a page or screen to be loaded or refreshed will be fully displayed in 6 seconds 95% of the time when accessed over a 10 Mbs Lan while there is a 200 user load on the system.*
  - *To validate that the system does not exhibit any critical failures under stress (unrealistic load)*
  - *Identify and ensure that performance issues uncovered outside of the stated performance criteria are documented and/or addressed prior to deployment.*

example 3

The objectives of the Performance Engineering Effort are:

- *To validate the scalability of the technical architecture and operability on a shared platform (up to 1000 concurrent users).*
- *To validate system performance of the following average page load times over a 100 Mbs Lan with the distribution of account sizes described later in this document.*

Screen Name	*Timer Name*	*Page Load time, no Load*	*Page Load w/ 200 User Load*
Group Summary	*tmr_grp_sum*	*12 sec*	*20 sec*
Employee Listing	*tmr_emp_lst*	*12 sec*	*20 sec*
Search Results View	*tmr_srch_rslt*	*12 sec*	*20 sec*
Customer Adjustment Popup	*tmr_cust_adj*	*6 sec*	*10 sec*
Term Confirmation Popup	*tmr_term_conf*	*9 sec*	*15 sec*

>

2.3 Engagement Complete Criteria

In cases where performance requirements or goals cannot be achieved due to situations outside of the control of the Performance Engineering Team, the performance effort will be considered complete when any of the following conditions are met:

<This is an example of engagement completion criteria. This should be used as an example. The actual engagement acceptance criteria should be listed here in the format of the example below

example

- *All bottlenecks preventing the application from achieving the performance criteria are determined to be outside Performance Engineering Team control/contract.*
- *The pre-determined engagement end date is reached.*
- *The Performance Engineering Team and stakeholders agree that the application performs acceptably, although some performance requirements or goals have not been achieved.*

>

3 WORKLOAD DISTRIBUTION

3.1 Introduction

A Workload Distribution is a representation of the functions performed by a user community on a system. For example, during the course of a day on a retail-based website, most users are shopping, some are doing a search for a specific product, some are checking out and all this while a single administrator may be updating prices on products. A Workload Distribution is based on a percentage of users performing a specific function over a given period of time. Using the above example a Workload Distribution could be: shopping – 83%, searching - 5%, checking out – 10% and administration - 2%.

Performance engineering is separate from functional testing; therefore, it is not necessary to simulate all possible paths through the system. Specific functions that are expected to draw the largest number of users are scripted to exercise the critical system components. Less vital functions that may not be used as heavily are scripted to provide a varying workload, creating a more realistic mix. As the number of concurrent users grows, the load increase on the system will often expand exponentially; therefore, it is imperative to accurately simulate the expected workload of the system. Experience shows that modeling 80% of a user population provides accurate load tests.

The Workload Distribution for a series of tests is represented in Rational Test Manger suites that include individual scripts. One or more scripts are recorded to represent each unique function to be tested. The suite defines the percentage of users that will execute each script as well as how users enter the site and how many times each function will be performed by each user.

3.2 Workload Distribution for <application>

The sections below describe, in detail, the workload distributions to be used for testing the *<application>.* The diagram (figure 3.1) shows the overall workload distribution path. The vertical dashed lines show the common start point for the scripts within the testing of the site. One can see that a virtual user's activity in the site does not necessarily depend on how the user got to the site and that each virtual user may exercise a unique combination of functionality based on the determined rate of frequency discussed below.

The Workload Distribution percentages presented in this document were created in order to simulate actual user activity on the site. These numbers however, may be modified or changed during the testing effort based on request from <*stakeholders*> in order to simulate alternate system load.

<The following figure and section is an example of a sample Workload Distribution. They should be used as an example and the activities and percentages should be changed to meet the specific Workload Distribution for the application under test.

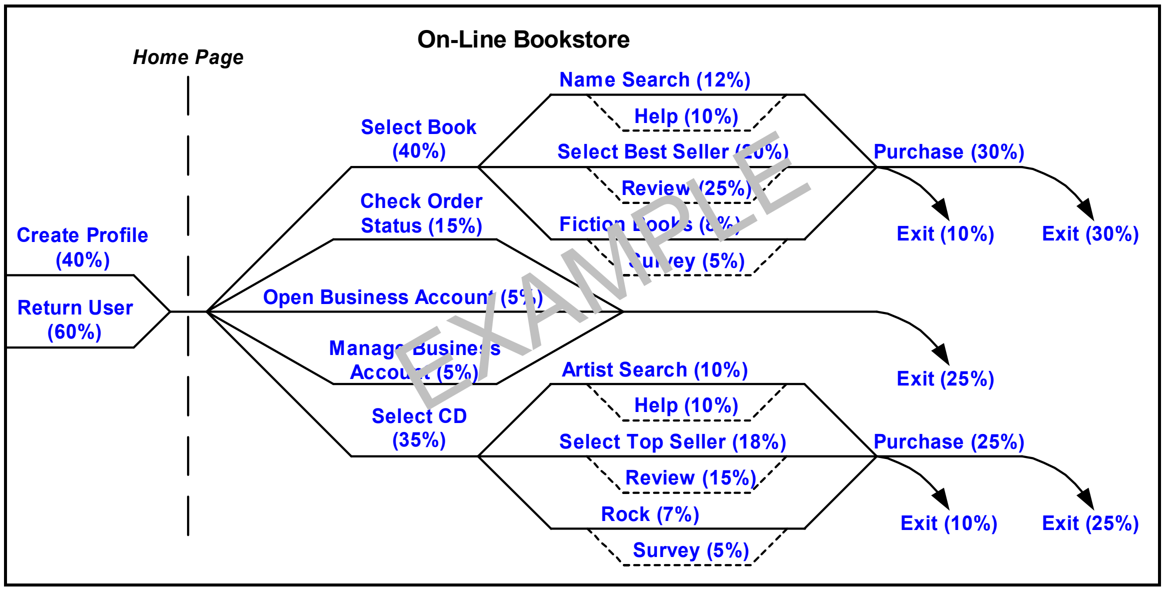

Figure 3.1 – Overall Workload Distribution

The Workload Distribution for <application> includes all of the functions to be executed during the performance testing effort. The dashed vertical line represents common start and/or end parts for the scripts (described in section 4). >

4 SCRIPT DESCRIPTIONS

4.1 Introduction

Rational TestStudio uses scripts and suites to plan and simulate user activities on an application. For the purpose of this document, a script is a block of code created to represent user activities. For this engagement, a script refers to all of the activities necessary for a user to navigate a specific path through the application. Scripts also include the code required to time the page load times, starting from a user-triggered event (i.e. clicking a button or link to request the next page) and ending when the page is fully displayed to the user. This code is commonly known as a "timer". A scenario groups scripts together and assigns them certain properties, such as number of users to simulate, when to launch the individual scripts and whether or not to repeat them.

Each script is designed to simulate a specific user activity as described in the following sections. Every script is recorded with user wait times to simulate the time an actual user would spend on the page before moving to the next page. These wait times are discussed in more detail in section 4 of this document.

<The following examples show how the remainder of this section should appear. The specific scripts that will be used for the performance effort should be detail here in the format of these examples:

Scripts that access bills will access bills of all sizes. Four different sizes of bills will be exercised; small (<50 pages) 25%, medium (51 - 200 pages) 50%, large (201-500 pages) 20%, extra large (501+ pages) 5%. This ensures that variances in page load time due to data volume will be appropriately accounted for during measurement. It will also ensure that the system is exercised realistically during multi-user tests.

The pages marked with (*)'s in figure 3.1 are effected by file size. These pages will be timed from page request until the page is rendered. This is different than page load times. These pages are presented so as to ensure that users can begin viewing data as soon as the first row is returned, but continues presenting data until all rows are returned. The time to present all the data for extremely large files may exceed 12 seconds, however, since the user will be viewing data while the rest is being presented, the timings for these pages stop when the first row of data is presented.

Five scripts, representing user groups of user actions on the system, have been identified that will be used to both exercise the Workload Distribution defined above as well as collect the single user end-to-end response (page load) times required for this engagement. Below are brief descriptions of each script as well as a page-by-page account of the measurements collected and data used by that script.

It is critical to simulate the length of time a user spends viewing a page or entering data when creating a load test. If these times are not simulated, the load test will simulate users navigating the site unrealistically fast, (view 10 pages in less than 5 seconds rather than about 5 min) which would yield artificially poor performance results. Each script will have think times that vary based on expected viewing time and distribution.

4.2 <Script Name 1> Script Overview

The view summary detail script starts where the login script leaves off, selects the view summary link then, based on the Workload Distribution in section 3.1, views one of the specific types of billing detail and logs out of the system.

	Page	***Timer Name***	***Acceptance Criteria***	***User Think Time***
1	*Summary Detail*	*tmr_summary*	*8 sec*	*15-45 sec (norm)*
2	*Home Calling and Long D*	*tmr_home_longd*	*8/12 sec*	*20-60 sec (linear)*
3	*Total Minutes*	*tmr_minutes*	*8/12 sec*	*10-70 sec (linear)*
4	*Roaming Charges*	*tmr_roaming*	*8/12 sec*	*15-35 sec (linear)*
5	*Texting Charges*	*tmr_texting*	*8/12 sec*	*12-42 sec (linear)*

4.2.1 <Script Name 1> Script Measurements

Describe all measurements collected by the script. This is generally page load times, but may sometimes include individual command processing, or component metrics such as CPU utilization or Memory usage. Include the name of each measurement and describe the meaning of the measurement

4.2.2 <Script Name 1> Script Think Times

Describe the length and variation of all of the think times that will be a part of the script

4.2.3 <Script Name 1> Script Data

Describe any data to be parameterized, the volume of data, and how any data consumption issues are to be handled

4.3 <Script Name 2> Overview

Repeat 4.2 until all scripts have been defined
>

5 TEST EXECUTION PLAN

5.1 Introduction

Performance Engineering occurs in an iterative fashion. During the initial stages of testing, tests and analysis continue until required tuning is identified. Once required tuning is identified, the tuning takes place, then the application is re-benchmarked and tests are repeated incrementally until either the target goals are met, or further required tuning is identified. In some cases additional tests will need to be developed and executed to assist developers/architects in correctly identifying and tuning bottlenecks. This entire process is repeated until no further tuning is possible during the allotted timeframe, or both the *<stakeholders>* and *<Performance Engineering Team>* agree that further testing and analysis is not required at this time. *<Performance Engineering Team>* has determined that, within the timeframe outlined above, one of the possible end states listed in sections 2.2 and 2.3 will be achieved. See figure 5.1 below for an illustration of the iterative testing process.

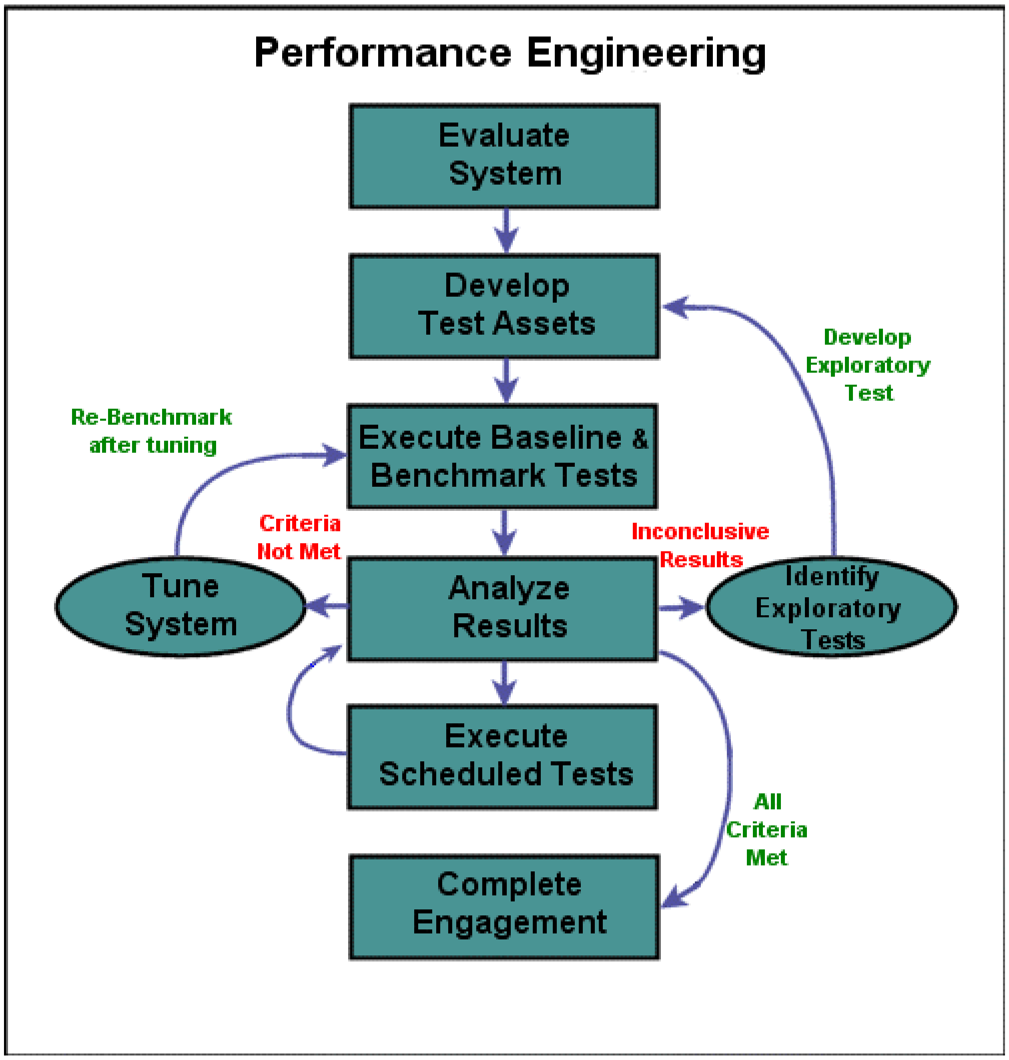

Figure 5.1 – Performance Engineering Methodology Overview

There are seven aspects of Performance Engineering as depicted in Figure 5.1. Each aspect has distinct activities. Each of these aspects and activities should be at least evaluated for every project. While there may be times when certain aspects and activities do not apply, one should always ensure that is a part of a decision, not an oversight.

There are three categories of tests that include all of the specific test types that are normally performed on a system. They are discussed in the three aspects titled Benchmarks, Scheduled Tests and Exploratory Tests. Various combinations of the tests included in these aspects, coupled with a scientific approach to their execution, can ultimately unveil all detectable performance issues and bottlenecks in a system.

5.2 Evaluate System

The Evaluate System aspect is a detail-oriented phase of a Performance Engineering Engagement. This aspect includes the following activities:

- Determine System Functions (Sections 3 & 4)
- Determine User Activities (Sections 3 & 4)
- Determine System Architecture (Section 5)
- Determine Acceptance Criteria (Section 2)
- Validate Acceptance Criteria (Section 2)
- Determine Usage Model (Sections 3 & 4)
- Validate Usage Model (Sections 3 & 4)
- Determine Potential Risks (Section 6)

5.2.1 System Architecture (Test)

The test environment consists of:
Proxy - *<describe>*
Web Server - *<describe>*
Appserver - *<describe>*
DB server - *<describe>*

5.2.2 System Architecture (Production)

The Production environment is a shared environment consisting of:
Load Balancer(s) - *<describe>*
Proxy – *<describe>*
Web Server(s) - *<describe>*
Appserver(s) - *<describe>*
Database - *<describe>*

If/when critical bottlenecks are detected that cannot be resolved in a short period of time, *<Performance Engineering Team>* and *<stakeholder>*managers will work together to determine the appropriate way to solve the problem.

5.3 Develop Test Assets

The Develop Test Assets aspect of performance engineering covers three activities:

- Develop Risk Mitigation Plan (Section 6)
- Develop Test/Engineering Strategy (Section 5)
- Develop Automated Test Scripts (Section 4)

These three items are all "living" items and will be updated throughout the engineering process. It is important to make these assets as complete and correct as possible early in the project to increase overall project efficiency.

5.3.1 Engineering Strategy (Schedule)

Example:

<The dates in this schedule are estimates and may be modified as the project proceeds. Task Ids 4 through 8 are sequential by module, but module testing is expected to overlap.

ID	***Date***	***Duration***	***Activities***	***Pre-Req***
1	*TBD*	*1 week*	• *Validate Requirements* • *Validate Usage Models* • *Validate Overall Strategy*	*None*
2	*TBD*	*Unknown*	• *Determine Test Environment* • *Determine Load Generation Environment* • *Determine Dates and locations for Environments*	*None*
3	*TBD*	*1 week*	• *Install and configure Test Environment* • *Install and configure Load Generation Environment* • *Validate connectivity and functionality* • *Validate performance users and authentication is configured*	*ID 2*
4	*TBD*	*1 day*	• *Record base script for login, logout and sample navigation (Common Tasks)*	*ID 2,3*
5	*TBD*	*Unknown*	• *Execute Common Tasks test at up to 100 users* • *Evaluate results* • *Follow process in section 5.1*	*ID 2,3,4*
6	*TBD*	*2-4 hours (per module)*	• *Record base scripts for module (may be done during systems testing)*	*ID 1,2,3*
7	*TBD*	*3-5 days (per module)*	• *Edit base scripts for module to match usage model.* • *Validate edited scripts*	*ID 1,2,3,6*
8	*TBD*	*5+ days (per module)*	• *Execute tests up to max load by module* • *Evaluate results* • *Follow process in section 5.1*	*ID 1,2,3,4,5, 6,7*
9	*TBD*	*1-2 weeks*	• *Execute collective load tests up to 250 total users.* • *Evaluate Results* • *Follow process in section 5.1*	*ID 1,2,3,4,5, 6,7,8*
10	*TBD*	*2 weeks*	• *Execute Stability tests* • *Evaluate Results* • *Follow process in section 5.1*	*ID 1,2,3,4,5, 6,7,8,9*
11	*TBD*	*1 week*	• *Revalidate last successful tests in Production*	*ID 1,2,3,4,5, 6,7,8,9*

12	*TBD*	*Unknown*	• *Tune*	*Unknown*
13	*TBD*	*Continual*	• *Update Strategy* • *Create and Update Results Document*	*ID 1*

>

5.4 Execute Baseline/Benchmark Tests

The Execute Baseline and Benchmark Tests aspect is geared toward executing light load scenarios are that are used to validate the correctness of the automated test scripts, identify obvious performance issues early in the testing cycle, and provide a basis of comparison for future tests. The activities included in this aspect of performance engineering are:

- Baseline Individual Scripts
- Create an Initial Benchmark
- Re-Benchmark after Tuning

For the <*application*>, both baselines and benchmarks will be used.

5.4.1 Baselines

Baseline results represent each script run individually over multiple iterations. Baselines are used primarily to validate that the scripts have been developed correctly. Occasionally, baseline tests will reveal performance problems that should be addressed immediately. All baselines are executed a minimum of 25 times. All reported times are statistical calculations (90th Percentile) of all 25 (or more) iterations. This eliminates the possibility of a statistical outlier skewing the result set. Each test execution run is separated by at least a minute, and every user wait time (time between user interactions with the system) will be exactly 8 seconds to ensure baseline tests are identical. For these tests all client side caching must be disabled, and each page must be pre-compiled. It is recommended that a method be put in place to pre-compile all pages automatically every time the test/production system is restarted. Baselines are in no way intended to represent actual users.

If after 25 or more iterations have been executed, the standard deviation among the results is high (more than half of the average time for that measurement), research may be required to determine the reason, as baseline tests should provide statistically similar results each time they are executed. If a baseline is found to be significantly slow, tuning may need to occur before subsequent tests are executed.

5.4.2 Benchmarks and Re-Benchmarks

A benchmark, or light load, scenario is generally a small community of users compared to the target load. This community of users must be large enough to represent a reasonable sample of the entire user community. Executing these tests ensures that the testing environment behaves as expected under light load before more demanding testing begins.

Additionally, the results of these tests are used as a benchmark to compare with all future tests results. Performance results obtained under the benchmark load should meet or exceed all indicated performance requirements; otherwise tuning must begin with the benchmark load. Assuming no performance problems are noticed during this scenario, the results obtained can be used as "best case" results. These results indicate how the system performs when it is not under noticeable stress, but is still performing all required functions, thus allowing conclusions to be drawn about the performance of the system during the higher load tests.

The <*application*>will be benchmarked, by module, in the environment described in section 5.2. This benchmark is intended to provide a basis of comparison for future testing. Tuning may occur during this benchmarking effort if critical bottlenecks are detected. Once any required tuning is complete, end-to-end page load times will be collected for each timer indicated in section 4 of this document.

Each module will be re-benchmarked each time an iteration of either tuning or development has been completed on a module. This ensures that there is always a known valid point of comparison for all scheduled tests. The Benchmark load for all modules will be 10 users, entering the system randomly over a 5-minute period and performing the tasks outlined in section 4 for either one hour or 5 complete iterations at a realistic user's pace.

If benchmark results do not meet the stated performance acceptance criteria, <*Performance Engineering Team*> and <*stakeholder*>will work together to either resolve the bottlenecks or revise the test strategy. Continuing on to the next phase of testing without fixing the performance issues will not add value to the project.

5.5 Analyze Results

The Analyze Results aspect is where the decisions are made about how to continue with the engineering effort. All of the remaining aspects are reached as a result of the decisions made in the Analyze results aspect. The activities included in this aspect of performance engineering are:

- Evaluate Results
- Determine if Acceptance Criteria is Met
- Determine of the Test is Conclusive
- Determine if Tuning is Required

Each test execution, regardless of type, must be analyzed to identify obvious or potential performance issues and/or the effects of the tuning effort. This analysis may or may not require formal reporting. At a minimum, notes should be kept and results saved, even for tests that are determined to be invalid. Analysis is often more of an art than a science. Testers, developers and architects should all be involved in the analysis process, since the results of the analysis drive the tuning effort.

5.5.1 Evaluate Results

Evaluating results is where the art part of Performance Engineering comes in. There is no single correct way to evaluate all of the data that results from a performance test execution. In general, though, evaluation is comparing to previous tests, finding and analyzing patterns, and applying past experiences. Some specific areas of evaluation for this project are:

<Example:

- *End-to-End response time (user experience)*
- *Weblogic web server response, configuration and tuning*
- *User authentication response, configuration and tuning (LDAP)*
- *<stakeholder>network response from various relevant geographic locations*
- *Tuxedo application server response, configuration and tuning*
- *Oracle database server response, configuration and tuning*
- *<Package Application> specific response and configuration*

- *Batch process response and efficiency*

>

5.5.2 Acceptance Criteria Met?

Determining if the acceptance criteria has been achieved is often the easiest determination made during a performance engineering engagement. This activity is simply comparing the results from the most recent test, or suite of tests, to the acceptance criteria. If all of the results meet or exceed the criteria, engineering is complete. Move on to the Complete Engagement Aspect. If not, continue evaluating results.

5.5.3 Conclusive?

Determining if a particular test or suite of tests is conclusive is often challenging. Most generally, if the results do not meet the acceptance criteria, you can't figure out what is causing the poor performance and the test results are reproducible, the test is probably inconclusive. In the case of an inconclusive test, move on to the Exploratory Test Needed aspect. Otherwise, continue evaluating the results.

5.5.4 Tune Now?

If you get this far, there is either a detected performance issue with the system, or more tests need to be conducted to validate compliance with additional acceptance criteria. If all the tests that have been conducted so far have met their associated acceptance criteria, but there are more criteria to be tested, move on to the Scheduled Tests aspect. If this is not the case, there are two options remaining. 1) re-execute the last test to see if the results are reproducible or 2) move on to the Tune aspect.

5.6 Scheduled Tests

The Execute Scheduled Tests aspect includes those activities that are mandatory to validate the performance of the system. These tests need to be executed even if no performance issues are detected and no tuning is required. There are only two activities that can be conducted in this aspect. They are

- Execute User Experience Tests
- Execute Stability Tests
- Execute Production Validation Tests

All of the tests described below are considered to be Scheduled tests no matter how many times they are executed due to development or tuning iterations.

5.6.1 User Experience Tests

User Experience Tests constitute what are considered to be expected real-world loads, from best case to worst case. Applying less than the expected worst-case load is useful in identifying major failings in a system, but does so in a way that doesn't highlight many of the more minor failings, allowing an easier analysis of results. When the load is equivalent to the expected real-world worst-case load, actual performance of the system can be measured, and associated problems can be clearly identified. Executing Load Tests before moving to other types of testing allows for the more major problems of a system to be identified and corrected separately prior to the smaller issues.

Upon completing benchmarks, load tests will be executed according to the Workload Distribution models outlined in section 3 of this document. These tests are designed to validate that the performance requirements outlined in section 2 have been met. Once the goals have been met, load testing ends. If the tests show that the performance goals have not been met, then bottlenecks must be determined and the application must be tuned if possible. These tests will be executed on the **<***?????***>** environment.

If critical bottlenecks are found, <*Performance Engineering Team*> and <*stakeholder*>will work together to determine how those bottlenecks are to be tuned. Tuning may require <*stakeholder*>resources and/or change requests to the <*Performance Engineering Team*> contract. While this is not expected, one cannot predict what performance tests may reveal.

Each module will be tested with up to 50 virtual users individually initially. When each individual module meets the acceptance criteria, the modules will be testing together at loads of 50, 100, 150, 200 and 250 virtual users.

5.6.2 Common Tasks Tests

<Example:

The first tests to be conducted will not be module specific. These tests will ensure that 100 users can log into the system in a 1 hour period and traverse the <package application> navigation tree with no performance issues. If these activities to not meet the acceptance criteria, tuning will begin here.
>

5.6.3 Remote Location Tests

<Example:

The <package application> production system will have geographically remote users. After scheduled tests have been conducted locally and are determined to be generally acceptable, remote users will be asked to test the performance of the system manually. This is not a formal test, but rather a sanity check. If the remote users agree that the performance is generally the same, no further remote testing is required.

If, however, the remote users determine that performance is significantly worse, remote testing will be done to determine the cause of the problem. This could take one of several forms:

a. *Manual timed downloads of same file from same server locally and remotely to estimate network latency.*
b. *Remote agent scripted downloads of same file from same server locally and remotely to determine network latency.*
c. *Remote agent execution of developed performance scripts to determine other latency.*

>

5.6.4 Stability Tests

Stability scenarios test a system at and beyond the worse expected demand it is likely to face. The majority of critical deficiencies in the system will have already been identified during the execution of load tests, so this phase deals more with assessing the impact on performance and functionality under a heavy or unreasonable load. Stability scenarios will also identify many other system bottlenecks not previously noticed, which may in fact be partially responsible for earlier identified problems.

Heavy load scenarios are generally designed to be far more than a system can handle. They are used not to identify ***if*** a system fails, but ***where*** it fails first, how badly, and why. By answering the *why* question, it can be determined whether a system is as stable as it needs to be.

The analysis performed in this phase can vary depending on the exact goals and objectives of the load testing. Results obtained from the test automation tool are often used in conjunction with results obtained directly from the system, where white-box testing has been employed. These tests are designed to find more subtle performance issues, such as memory leaks, caching and database locking but are not designed to collect end-to-end user experience measurements in most cases.

5.6.4.1 Stress Tests

Stress Tests are tests that use real-world distributions and user communities, but under extreme conditions. It is common to execute stress tests that are 100% of expected peak expected user-load sustained over 8-12 hours, and 150% expected peak user-load with normal ramp up and ramp down time.

5.6.4.2 Spike Tests

Spike Tests are tests that used real-world distributions and user communities, but under extremely fast ramp up and ramp down times. It is common to execute stress tests that ramp up to 100% or 150% of expected peak expected user-load in a matter of minutes rather than about an hour.

5.6.4.3 Hammer Tests

Hammer Tests have little or no resemblance to real-world distributions and user communities. These test take all existing load generation scripts and methods, eliminate user think times and increase load until something stops functioning properly. These tests are designed to make the system fail. Once failures occur, it can be decided how to mitigate the risk of that particular failure.

5.6.5 Batch Baselines

Batch process testing will be treated as a validation exercise unless/until any batch process does not complete within its allotted time window during validation. Batch processes will be launched manually and timed. Any batch that completes in under 90% of its allotted time window will be considered acceptable. Any batches that do not complete within 90% of their time window will be evaluated, by component, and iteratively tuned to the greatest degree possible.

5.6.6 Production Validation Tests

Production Validation is conducted to ensure that the application performs as expected in the production environment after testing and tuning have been completed in a test environment.

Upon completing the tuning effort, a load test (250 concurrent users) will be executed according to the Workload Distribution model outlined in Section 3.   This will take place in the production environment between deployment and going live. This test is designed to validate that the performance results obtained in the *<??????***>** environment are also achieved in the production environment.

5.7 Identify Exploratory (Specialty) Tests

The Identify Exploratory Test Needed aspect of performance engineering is a three-step process used to narrow the scope of the required exploratory test as much as possible before developing the actual test, or updating documentation. The three activities in this aspect are:

- Identify area (tier/component) of the test.
- Identify the desired measurement(s) to be collected.
- Identify the type of test needed.

Exploratory (or Specialty) Tests are most commonly "Black box" and/or "White box" tests on specific components used to identify and tune the *why* behind the performance issued identified during various types of load testing described above. These tests are generally not developed until a specific bottleneck or performance issue has been identified that requires more information to be gathered. For this reason, detailed information about Black and White Box tests will not be specifically detail in the following sections. They are here to explain how investigation may occur if the application is found to have performance issues requiring tuning.

5.7.1 Black Box Tests

Black box testing traditionally has involved testing a system as a whole; however, with the advent of multi-tiered systems, black box testing has come to also refer to testing tiers or components of a total system, since those tiers are often entire systems unto themselves with only loose ties to the other tiers of the system. Tests that focus on a specific tier can determine whether the detected bottleneck resides in that tier. Black box tests generally do not point to the specific bottleneck. To find the actual bottleneck, diagnostics may be performed on the tier while generating a load.

5.7.2 White Box Tests

White box testing treats the system as a collection of many parts. During white box testing, many diagnostics will be run on the server(s), the network, and even on clients. This allows the causes of bottlenecks to be easily identified, and therefore addressed. White box testing can go as deep as individual lines of code, database query optimization, or memory management of segments of code.

White box testing requires much greater technical knowledge of the system than does black box testing. To perform white box testing, knowledge is needed about the components that make up the system or tiers, how they interact, the algorithms involved with the components, and the configuration of the components. Knowledge on how to perform diagnostics against these components, and even what diagnostics are needed and are appropriate is required to execute white box tests.

There are several problems with white box testing. First, is the affect that running diagnostics has on system performance. Ideally, running diagnostics should have no impact on performance. However, this is rarely possible since running diagnostics consumes some system resources. When considering the appropriateness of various diagnostics, consider the benefit vs. the impact on performance. Second is that white box tests are often difficult and time consuming to develop and execute. For this reason, white box tests should only be used when there is a high expectation that the tests will yield noticeable performance improvements.

5.8 Tune

The Tune aspect is at the heart of the Performance Engineering effort. While Tune only has thee enumerated activities, there are entire shelves of books written about tuning almost every known component of a system or application. The important part of Tuning within a Performance Engineering effort is the collaboration between the performance engineer and the system experts, developers, architects, DBA's, system administrators, etc. Remember to always re-baseline the system after tuning is accomplished. The activities in the Tune aspect are:

- Make Changes to the System
- Validate Improvement of the Change
- Document Improvement

5.9 Project Closeout

The Project Closeout aspect only has one activity, but that activity is important and independent enough to deserve an aspect of it's own. That activity is:

- Document Results

Typically, the Results document includes:

- 95th percentile response times for each page/activity grouped by user load (Graphed)
- Notations where actual responses did not achieve stated goals
- Analysis of reported times, in particular where times did not meet stated goals
- Appendices including:
    a. Spreadsheets of recorded times (min, max, avg, and std) for each user load
    b. Spreadsheet of average times grouped by user load
    c. Test execution reports (copied directly from playback tool) for each test executed

6 RISKS

6.1 Introduction

<Example:
The following section should detail all risks to the success of a performance engagement. The details of each risk should be explicit and mitigation strategies should be identified for each risk.

6.2 Risk 1 Prioritizing mentoring vs. testing

Discussion: <stakeholder> and Performance Engineering Team must determine, in detail, the role of the Performance Engineering Team. The Performance Engineering Team cannot reasonably be held responsible for both fully training the client employee and guaranteeing the efficiency and effectiveness of the engagement. Some work will be accomplished by the client employee that cannot be fully reviewed. Additionally,the Performance Engineering Team have years of experience that allow them to accomplish certain tasks more efficiently than a novice Performance Engineer.

Mitigation Strategy: Ensure client and Performance Engineering Team agree to the responsibilities of the Performance Engineering Team prior to starting work.

Owner: Performance Engineering Team Lead & Lead stakeholder

Status: Open

>