

|  |
| --- |
| Performance Test Strategy |
| RightFind Enterprise |

|  |  |
| --- | --- |
| Abbreviations and Acronyms | |
| VU | Virtual User |
| RFE | RightFind Enterprise |
| QA | Quality Assurance |
| SLA | Service Level Agreement |
| KPI | Key Performance Indicator |
| SaaS | Software as a Service |

Table of Contents

[1. INTRODUCTION 5](#_Toc74331462)

[1.1 Description 5](#_Toc74331463)

[1.2 Purpose 5](#_Toc74331464)

[1.3 Scope 5](#_Toc74331465)

[2. PERFORMANCE ACCEPTANCE CRITERIA 5](#_Toc74331466)

[2.1 Introduction 5](#_Toc74331467)

[2.2 Performance Criteria 5](#_Toc74331468)

[2.2.1 Requirements 6](#_Toc74331469)

[2.2.2 Goals 6](#_Toc74331470)

[3. WORKLOAD DISTRIBUTION 7](#_Toc74331471)

[3.1 Introduction 7](#_Toc74331472)

[3.2 Workload Distribution for RightFind Enterprise 7](#_Toc74331473)

[4. WORKFLOW 8](#_Toc74331474)

[4.1 Introduction 8](#_Toc74331475)

[4.2 Iterative Performance Testing Activities 8](#_Toc74331476)

[4.2.1 Activity 1: Understand the Project Vision and Context 8](#_Toc74331478)

[4.2.2 Activity 2: Identify Reasons for Testing Performance 10](#_Toc74331480)

[4.2.3 Activity 3: Identify the Value Performance Testing Adds to the Project 11](#_Toc74331482)

[4.2.4 Activity 4: Configure the Test Environment 12](#_Toc74331484)

[4.2.5 Activity 5: Identify and Coordinate Tasks 13](#_Toc74331486)

[4.2.6 Activity 6: How often should run performance tests 14](#_Toc74331488)

[4.2.7 Activity 7: Analyze Results, compare with previous executions and Report 15](#_Toc74331490)

[4.2.8 Activity 8: Revisit Activities 1-3 and Consider Performance Acceptance Criteria 16](#_Toc74331492)

[4.2.9 Activity 9: Reprioritize Tasks 17](#_Toc74331494)

[5. PERFORMANCE TEST TYPES (WITH PRIORITIES) 18](#_Toc74331496)

[5.1 Server-side tests *(high priority)* 18](#_Toc74331497)

[5.1.1 Pipe-clean (Baseline) test 18](#_Toc74331498)

[5.1.2 Capacity (Ramp up) test 18](#_Toc74331499)

[5.1.3 Fixed load (Benchmark) test 18](#_Toc74331500)

[5.1.4 Longevity (Endurance) tests 19](#_Toc74331501)

[6. METRICS AND CALCULATIONS 20](#_Toc74331502)

[6.1 Metrics 20](#_Toc74331503)

[6.2 Calculations 20](#_Toc74331504)

[6.2.1 Averages 22](#_Toc74331505)

[6.2.2 Percentiles 22](#_Toc74331506)

[6.2.3 Medians 23](#_Toc74331507)

[6.2.4 Normal Values 23](#_Toc74331508)

[6.2.5 Standard Deviations 23](#_Toc74331509)

[6.2.5 Uniform Distributions 23](#_Toc74331510)

[6.2.6 Normal Distributions 23](#_Toc74331511)

[6.2.7 Statistical Significance 24](#_Toc74331512)

[6.2.8 Statistical Equivalence 24](#_Toc74331513)

[6.2.9 Statistical Outliers 24](#_Toc74331514)

[6.2.10 Confidence Intervals 25](#_Toc74331515)

[7. TOOL SELECTION APPROACH 27](#_Toc74331516)

[7.1 Performance Testing Tool Architecture 27](#_Toc74331517)

[7.1.1 Scripting module 27](#_Toc74331518)

[7.1.2 Test management module 27](#_Toc74331519)

[7.1.3 Load injector(s) 27](#_Toc74331520)

[7.1.4 Analysis module 27](#_Toc74331521)

[7.1.5 Optional modules 27](#_Toc74331522)

[7.2 Choosing a Performance Testing Tool 27](#_Toc74331523)

[7.2.1 Considerations 28](#_Toc74331524)

[8. PERFORMANCE MANAGEMENT FRAMEWORK ARCHITECTURE 30](#_Toc74331525)

# 1. INTRODUCTION

## Description

This document defines the approach of performance testing the RightFind Enterprise application. It describes methods and tools used by Performance Engineer(s) to validate and tune the performance of the application.

## Purpose

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

* Performance Acceptance Criteria;
* Workload Distribution to be used to exercise and gather measurements from the application;
* Performance Testing workflow;
* Test types to be performed;
* Measurements to be collected;
* User patterns to be modeled that include user think times;
* Tools and infrastructure.

## Scope

This document provide strategy to carry out all performance engineering activities for the RightFind Enterprise application. It discusses resources required, includes toolset to accomplish test execution, results analysis, and application performance tuning. It covers Performance Acceptance Criteria, explains the user interaction models to be tested, and describes scripts and schedules to be developed. 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.

# 

# 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 RightFind Enterprise performance optimization 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 of performance requirements and goals for the system under test. In the case of the dev team, performance engineer(s) and stakeholders have worked collaboratively through mutual experience, conversations, and workshops to develop the criteria enumerated below. The preferred result by both performance engineer(s) 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.

### 2.2.1 Requirements

Non-Functional Requirements must be defined and met for the RightFind Enterprise application.

### 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.

One of the goals is to ensure that RightFind Enterprise application performance do not have significant performance degradation.

# 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 load generation tool 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 RightFind Enterprise

The sections below describe, in detail, the workload distributions to be used for testing RightFind Enterprise. The Workload Distribution percentages presented in this document were created in order to simulate actual user activity on the system. These numbers however, may be modified or changed during the testing effort based on request from stakeholders in order to simulate alternate system load.

All information about the performance testing scope and workload distribution stored in the "CCC-ARCH\_RFE\_Performance\_Test\_Plan" document.

# 4. WORKFLOW

## 4.1 Introduction

This section describes the concepts underlying the activities necessary to make performance testing successful within an iterative process, as well as specific, actionable items that can be immediately applied to the RightFind Enterprise application in order to gain a significant return on this investment. The key to working within an iteration-based work cycle is team coordination. For this reason, performance engineer must be able to adapt what he or she measures and analyzes per iteration cycle as circumstances change.

## 4.2 Iterative Performance Testing Activities

This approach can be represented by using the following nine activities below:

* **Activity 1. Understand the Project Vision and Context**. The outcome of this activity is a shared understanding of the project vision and context.
* **Activity 2. Identify Reasons for Testing Performance**. Explicitly identify the reasons for performance testing.
* **Activity 3. Identify the Value Performance Testing Adds to the Project**. Translate the project- and business-level objectives into specific, identifiable, and manageable performance-testing activities.
* **Activity 4. Configure the Test Environment**. Set up the load-generation tools and the system under test, collectively known as the performance test environment.
* **Activity 5. Identify and Coordinate Tasks**.  Prioritize and coordinate support, resources, and schedules to make the tasks efficient and successful.
* **Activity 6. How often should run performance tests**. Execute the activities for the current iteration.
* **Activity 7. Analyze Results, compare with previous executions and Report**. Analyze and share results with the team.
* **Activity 8. Revisit Activities 1-3 and Consider Performance Acceptance Criteria**.  Between iterations, ensure that the foundational information has not changed. Integrate new information such as customer feedback and update the strategy as necessary.
* **Activity 9. Reprioritize Tasks**. Based on the test results, new information, and the availability of features and components, reprioritize, add to, or delete tasks from the strategy, and then return to activity

### 4.2.1 Activity 1: Understand the Project Vision and Context

This activity takes place only at the beginning of diving to the project. The project vision and context are the foundation for determining what performance testing activities are necessary and valuable. Because the Performance Engineer is not driving these items, the coordination aspect refers more to team education about the performance implications of the project vision and context, and to identifying areas where future coordination will likely be needed for success. A critical part of working with an iteration-based process is asking the correct questions, providing the correct value, and performing the correct task related to each step. Although situations can shift or add more questions, values, or tasks, a sample checklist is provided as a starting point for each step.

**Checklist**

**Questions to ask:**

* What are the performance implications of the project vision?
* What are the performance implications of the service the application is intended to provide, or what problem are we trying to solve for the customer?
* How does the team envision performance testing as it relates to the project schedule, structure, and available resources?

**Value provided:**

* Be involved in the product concept.
* Point out any areas of concern immediately.
* Point out assumptions related to available resources, tools, and resource-monitoring instrumentation based on the project vision and context as soon as they arise.

**Tasks accomplished:**

* Ask the whole team questions and provide answers.
* Determine the team’s perception of performance testing.
* Gain a conceptual understanding of the project’s critical performance implications.
* Begin to define equipment and/or resources needed for conducting performance testing.
* Understand resource constrains; for example, budget, people, equipment.
* Understand how the team will coordinate.
* Understand how the team will communicate.

**Coordinate with:**

* Whole team

### 4.2.2 Activity 2: Identify Reasons for Testing Performance

The underlying reasons for testing performance on a particular project are not always obvious based on the vision and context alone. Project teams generally do not include performance testing as part of the project unless there is some performance-related risk or concern they feel needs to be mitigated. Explicitly identifying these risks and areas of concern is the next fundamental step in determining what specific performance testing activities will add the most value to the project. Regardless of when a performance tester joins the team, once the project vision and context are understood, it is worth taking the time to verbalize and/or document the overall objectives of the performance-testing effort based on the risks or concerns that the team has. The following checklist should help to accomplish this step.

**Checklist**

**Questions to ask:**

* What risk(s) is performance testing intended to mitigate for this project?
* Are there specific contractual, compliance, or customer performance expectations that are already known to be required?
* What performance concerns relating to this project already exist?

**Value provided:**

* Be involved in the product concept.
* Point out any areas of concern immediately.
* Point out resource and instrumentation assumptions based on the project vision and context when they arise.
* Guide the process of collecting/determining performance-testing objectives.
* Capture implied usage scenarios of particular performance concerns.
* Capture implied performance goals, requirements, targets, and thresholds as they come up in conversation.

**Tasks accomplished:**

* Ask the whole team questions and provide answers.
* Determine the project-level objectives for conducting performance testing.
* Refine estimates of equipment and/or resources required for conducting performance testing.
* Identify disconnects between the objectives of the performance-testing effort and the equipment and resources to be made available.
* Capture implied performance goals, requirements, targets, and thresholds to be fleshed out later.
* Capture implied usage scenarios of particular concern to be fleshed out later.

**Coordinate with:**

* Whole team

### 4.2.3 Activity 3: Identify the Value Performance Testing Adds to the Project

Using information gained from activities 1 and 2, we can now clarify the value added through performance testing and convert that value into a conceptual performance-testing strategy. The point is to translate the project- and business-level objectives into specific, identifiable, and manageable performance-testing activities. The coordination aspect of this step involves team-wide discussion and agreement on which performance-testing activities are likely to add value or provide valuable information, and if these activities are worth planning for at this time.

**Checklist**

**Questions to ask:**

* What performance-testing activities will help achieve the performance-testing objectives?
* What performance-testing activities are needed to validate any contractual, compliance, project, or customer performance criteria or expectations that are known at this time?
* What performance-testing activities will help address currently known performance concerns?

**Value provided:**

* Ensure team-wide support of performance-testing activities.
* Ensure that the team has adequate warning about performance-testing activities that will require the support of additional team members.
* Determine if resource and instrumentation assumptions are adequate.
* Guide the process of determining how performance-testing objectives will be measured.
* Capture additional implied usage scenarios of particular performance concerns.
* Capture additional implied performance goals, requirements, targets, and thresholds as they come up in conversation.

**Tasks accomplished:**

* Ask the whole team questions and provide answers.
* Determine a conceptual project-level strategy for determining if the objectives for conducting performance testing have been met.
* Refine estimates of equipment and/or resources required for conducting performance testing.
* Identify disconnects between the objectives of the performance-testing effort and the equipment and resources to be made available.
* Capture additional implied performance goals, requirements, targets, and thresholds to be fleshed out later.
* Capture additional implied usage scenarios of particular concern to be fleshed out later.

**Coordinate with:**

* Whole team

### 4.2.4 Activity 4: Configure the Test Environment

With a conceptual strategy in place, the tools and resources should be prepared in order to execute the strategy as features and components become available for test. This step should be performed as soon as possible, so that the team has this resource from the beginning. And the last but not least – performance testing environment should be pretty close to production environment.

This step is fairly straightforward. Set up the load-generation tools and the system under test — collectively known as the performance test environment — and ensure that this environment will meet engineering needs. The coordination component of this step typically involves asking managers and administrators to obtain and/or configure equipment and other resources that are not under the direct control of the team or performance tester.

**Checklist**

**Questions to ask:**

* Who administrates the performance-testing environment of the application under test?
* Who administrates the load-generation tool/environment?
* Who configures and operates resource monitors for the application under test?
* Is special permission needed prior to generating load of a certain volume?
* Who can reset the application under test?
* What other components require special coordination?
* What security or authentication considerations are there for simulating multiple users?
* What coordination needs to be done to enable the use of recording and/or monitoring software?

**Value provided:**

* Ensure that the load-generation and performance-test environments are ready when the team needs them.
* Ensure that the entire team knows who to contact for help with performance-testing environment support.
* Ensure that performance testing support staff knows what they are supporting.

**Tasks accomplished:**

* Performance-test environment configured and ready to begin testing.
* Load-generation environment configured and ready to begin testing.
* Support responsibilities assigned.
* Special permissions, time of day for high load tests, etc., determined.

**Coordinate with:**

* System administrators
* Network support
* Database administrators
* Infrastructure support
* Managers of those above
* Development team

### 4.2.5 Activity 5: Identify and Coordinate Tasks

Performance testing tasks do not happen in isolation. The performance specialist needs to work with the team to prioritize and coordinate support, resources, and schedules to make the tasks efficient and successful.

During the pre-iteration planning meeting, look at where the project is now and where you want to be to determine what should and can be done next. When planning for the iteration cycle, the performance tester is driven by the goals that have been determined for this cycle. This step also includes signing up for the activities that will be accomplished during this cycle.

**Checklist**

**Questions to ask:**

* What is the performance goal for this cycle?
* Where is the project in terms of the overall project performance goals?
* Has the system achieved all of its performance objectives?
* Has tuning been accomplished since the last iteration?
* What analysis, reports, or retesting will add value during this iteration?
* Who requires pairing in order to do performance testing?
* How much time is available?
* How much time does each task take?
* What is the most critical activity?

**Value provided:**

* Provide insight on how the overall project is achieving its goal.
* Provide insight on what can be measured and reported on in this cycle.
* Provide insight on any critical issues that may have arisen from the last iteration cycle.
* Make suggestions to other team members.
* Transfer lessons learned as they emerge from the test.
* Pair with developers to improve performance unit testing.
* Help reuse unit tests.
* Help reuse functional tests.

**Tasks accomplished:**

* Estimate how much work is achievable.
* Determine if anyone needs to be paired out.
* Prioritize achievable work.
* Identify primary and alternate tasks for this cycle.

**Coordinate with:**

* Managers and stakeholders
* Developers and administrators
* Infrastructure and test environment support
* Users or user representatives

### 4.2.6 Activity 6: How often should run performance tests

Conduct tasks in one- to two-day segments. See them through to completion, but be willing to take important detours along the way if an opportunity to add additional value presents itself. Step 5 defines what work the team members will sign up for in this iteration. Now it is time to execute the activities for this iteration.

**Checklist**

**Questions to ask:**

* Have recent test results or project updates made this task more or less valuable compared to other tests we could be conducting right now?
* What additional team members should be involved with this task?
* Are there other important tasks that can be conducted in parallel with this one?
* Do the preliminary results make sense?
* Is the test providing the data we expected?

**Value provided:**

* Evaluate algorithm efficiency.
* Monitor resource usage trends.
* Measure response times.
* Collect data for scalability and capacity planning.
* Transfer lessons learned as they emerge from the test.
* Improve performance unit testing by pairing performance testers with developers.
* Help reuse unit tests.
* Help reuse functional tests.

**Tasks accomplished:**

* Conduct tests.
* Collect data.
* Validate test assumptions and techniques.
* Potentially tune while testing.
* Pair with other team members; this does not mean only working with a developer or tester but can also mean working with a writer to capture his or her understanding of how the system performance works, or working with the customer directly.

**Coordinate with:**

* Developers and administrators
* Infrastructure and test environment support
* Users or user representatives
* Managers and stakeholders
* Other performance testers who are not on the project

### 4.2.7 Activity 7: Analyze Results, compare with previous executions and Report

To keep up with an iterative process, results need to be analyzed and shared quickly. If the analysis is inconclusive, retest at the earliest possible opportunity to give the team maximum time to react to performance issues. As the project is wrapped for final shipping, it is usually worth having a meeting afterward to collect and pass along lessons learned. In most cases it is valuable to have a daily or every-other-day update to share information and coordinate next tasks.

**Checklist**

**Questions to ask:**

* Do the preliminary results make sense?
* Is the test providing the data we expected?
* Is the data valuable?
* Are more tests required to derive meaning from the data?
* Is tuning required? If so, do we know what to tune?
* Do the results indicate that there are additional tests that we need to execute that have not been planned for?
* Do the results indicate that any of the tests we are planning to conduct are no longer necessary?
* Have any performance objectives been met?
* Have any performance objectives been rendered obsolete?

**Value provided:**

* Evaluate algorithm efficiency.
* Monitor resource usage trends.
* Measure response times.
* Collect data for scalability and capacity planning.
* Transfer lessons learned as they emerge from the test.

**Tasks accomplished:**

* Analyze data collaboratively.
* Determine the meaning of the results.
* Share data with the whole team.
* Import lessons learned into future iteration planning.

**Coordinate with:**

* Developers and administrators
* Managers and stakeholders
* Users or user representatives
* Other performance testers who are not on the project

### 4.2.8 Activity 8: Revisit Activities 1-3 and Consider Performance Acceptance Criteria

Between iterations, ensure that the foundational information has not changed. Integrate new information, such as customer feedback, and update the strategy as necessary.

**Checklist**

**Questions to ask:**

* Have the performance implications of the project vision changed?
* Have the performance implications of the service we are trying to provide changed, or has the problem we are trying to solve for the customer changed?
* Have the project schedule, structure, or available resources changed?
* Have the performance-testing objectives changed?
* Have the performance-testing activities needed to validate any contractual, compliance, project, or customer performance criteria or expectations changed?
* What performance-testing activities will help address currently known performance concerns?

**Value provided:**

* Update resource and instrumentation assumptions and needs.
* Point out any areas of concern.
* Point out resource and instrumentation needs and/or risks.
* Update performance-testing objectives.
* Enhance and update usage scenarios of particular performance concerns.
* Enhance and update performance goals, requirements, targets, and thresholds.
* Ensure that the team has adequate warning about upcoming performance-testing activities that will require the support of additional team members.

**Tasks accomplished:**

* Enhance and update understanding of the project’s critical performance implications.
* Update resource constraints; for example, budget, people, and equipment.
* Update/improve how the team will coordinate.
* Update/improve how the team will communicate.
* Revise performance-testing strategy.
* Refine estimates of equipment and/or resources required for conducting performance testing.
* Identify incompatibilities or conflicts between the objectives of the performance-testing effort and the equipment and resources to be made available.
* Capture additional performance goals, requirements, targets, and thresholds.
* Capture additional usage scenarios of particular concern.
* Report current performance-testing status.

**Coordinate with:**

* Whole team

### 4.2.9 Activity 9: Reprioritize Tasks

Based on the test results, new information, and the availability of features and components, reprioritize, add to, or delete tasks from the strategy, and then return to activity 5.

**Checklist**

**Questions to ask:**

* What performance-testing activities will help address currently known performance concerns?
* What is the performance goal for this cycle?
* Where is the project in terms of the overall project performance goals?
* Has the system achieved all of its performance objectives?
* Has tuning been accomplished since the last iteration?
* What analysis, reports, or retesting will add value during this iteration cycle?
* Who requires pairing to do performance testing?
* How much time is available?
* How much time does each task take?
* What is the most critical activity?

**Value provided:**

* Provide insight on how the overall project is achieving its goal.
* Provide insight on what can be measured and reported on in this cycle.
* Provide insight on any critical issues that may have arisen from the last iteration.
* Make suggestions to other team members.
* Transfer lessons learned as they emerge from the test.
* Pair with developers to improve performance unit testing.
* Help reuse unit tests.
* Help reuse functional tests.

**Tasks accomplished:**

* Report current performance-testing status.
* Estimate how much work is achievable.
* Determine if anyone needs to be paired out.
* Prioritize achievable work.
* Identify primary and alternate tasks for this cycle.

**Coordinate with:**

* Managers and stakeholders
* Product owners

# 5. PERFORMANCE TEST TYPES (WITH PRIORITIES)

## 5.1 Server-side tests *(high priority)*

### 5.1.1 Pipe-clean (Baseline) test

Priority: Highest

The pipe-clean test is a preparatory task that serves to validate each performance test script in the performance test environment. The test is normally executed for a single use case as a single virtual user for a set period of time or for a set number of iterations. This execution should ideally be carried out without any other activity on the system to provide a best-case measurement. You can then use the metrics obtained as a baseline to determine the amount of performance degradation that occurs in response to increasing numbers of users and to determine the server and network footprint for each scripted use case.

#### Purpose:

To check that system is ready for performance testing and scripts are met the requirements

### 5.1.2 Capacity (Ramp up) test

Priority: Highest

Capacity of a system is the highest level of load it can take and handle without:

* Significant response times increase
* Stability decrease

In other words, capacity is the measure of how powerful the system is.

A capacity test complements load testing by determining server’s ultimate failure point. You perform capacity testing in conjunction with capacity planning, which you use to plan for future growth, such as an increased user base or increased volume of data. For example, to accommodate future loads, you need to know how many additional resources (such as processor capacity, memory usage, disk capacity, or network bandwidth) are necessary to support future usage levels. Capacity testing helps to identify a scaling strategy in order to determine whether system should scale up or scale down.

#### Purpose:

To determine how many users and/or transactions a given system will support and still meet performance goals, to find out server capacity, stability under incremental load and scalability of the system. Also, capacity testing results are key points to create performance tests of another types, for example fixed-load or stress.

### 5.1.3 Fixed load (Benchmark) test

Priority: High

Fixed load testing is conducted to verify that your application can meet your desired performance objectives; these performance objectives are often specified in a service level agreement (SLA). A load test enables you to measure response times, throughput rates, and resource-utilization levels, to verify application behavior under normal and peak load conditions.

Define the load level vs capacity:

* Low-load (~10% of capacity)
* Mid-load (~45% of capacity)
* High-load (~80% of capacity)
* Or some defined level of load (e.g. production-like, expected, etc.)

Duration should be:

* long enough to make results statistically meaningful
* short enough to avoid biased errors

#### Purpose:

* to get response times (and some other metrics) statistics under different levels of load and compare them against target/previous release (build, sprint, etc.) results
* to check that system under particular level of load is stable for a particular period of time
* (optional) system resources/application profiling for problematic transactions

Generally speaking, the goal of checking system stability is correct for any performance test with fixed level of load.

### 5.1.4 Longevity (Endurance) tests

Priority: High

This is a long high load test to check if system can work with no issues for a long period of time. This is the most important stability test because it assumes no extreme cases, but rather a normal operation over time.

Basically, it`s the same as fixed load test, next parameters should be defined:

* Load
* Duration

The tricky part is how to balance load/duration:

* Less load -> more duration is needed: production/expected level of load for a week or so (ideal case)
* More load -> less duration is possible: higher load (e.g. 80% of capacity) for 24-48 hours (a more common case)

#### Purpose:

* to assess system stability during long-time load
* to find memory leaks
* to check if backend background activities affect system performance

# 6. METRICS AND CALCULATIONS

## 6.1 Metrics

When identified and captured correctly, metrics provide information about how well or poorly application is performing as compared to performance objectives. In addition, metrics can help identify problem areas and bottlenecks within application.

Using the desired performance characteristics, identify metrics to be captured that focus on potential pitfalls for each scenario. The metrics can be related to both performance and throughput goals as well as providing information about potential problems; for example, custom performance counters that have been embedded in the application.

The following table describes performance metrics in terms of related performance objectives.

**Processor:**

• Processor utilization

**Process:**

• Memory consumption

• Processor utilization

• Process recycles

• Garbage collection

**Memory:**

• Memory available

• Memory utilization

**Disk:**

•Disk utilization

•I/O operations rate

**Network:**

• Network utilization

• Latency

**Transactions/business metrics:**

• Transactions/sec (throughput)

• Transactions succeeded

• Transactions failed

• Samples succeeded

• Samples failed

**Threading:**

• Contentions per second

• Deadlocks

• Thread allocation

**Response times:**

• Transactions times

• Samples response times

## 6.2 Calculations

This section refers to three exemplar data sets for the purposes of illustration, namely:

• Data Set A

• Data Set B

• Data Set C

**Data Sets Summary**

The following is a summary of Data Sets A, B, and C.

Graphical user interface, application

Description automatically generated

**Data Set A**

Chart, histogram

Description automatically generated

100 total data points, distributed as follows:

* 5 data points have a value of 1
* 10 data points have a value of 2
* 20 data points have a value of 3
* 30 data points have a value of 4
* 20 data points have a value of 5
* 10 data points have a value of 6
* 5 data points have a value of 7

**Data Set B**

Graphical user interface, text, application

Description automatically generated

100 total data points, distributed as follows:

* 80 data points have a value of 1
* 20 data points have a value of 16

**Data Set C**

Chart, bar chart, histogram

Description automatically generated

100 total data points, distributed as follows:

* 11 data points have a value of 0
* 10 data points have a value of 1
* 11 data points have a value of 2
* 13 data points have a value of 3
* 11 data points have a value of 4
* 11 data points have a value of 5
* 11 data points have a value of 6
* 12 data points have a value of 7
* 10 data points have a value of 8

### 6.2.1 Averages

An average - also known as an arithmetic mean, or mean for short - is probably the most commonly used, and most commonly misunderstood, statistic of all. To calculate an average, you simply add up all the numbers and divide the sum by the quantity of numbers you just added. What seems to confound many people the most when it comes to performance testing is that, in this example, Data Sets A, B, and C each have an average of exactly 4. In terms of application response times, these sets of data have extremely different meanings. Given a response time goal of 5 seconds, looking at only the average of these sets, all three seem to meet the goal. Looking at the data, however, shows that none of the data sets is composed only of data that meets the goal, and that Data Set B probably demonstrates some kind of performance anomaly. Use caution when using averages to discuss response times and, if at all possible, avoid using averages as the only reported statistic. When reporting averages, it is a good idea to include the sample size, minimum value, maximum value, and standard deviation for the data set.

### 6.2.2 Percentiles

Few people involved with developing software are familiar with percentiles. A percentile is a straightforward concept that is easier to demonstrate than define. For example, to find the 95th percentile value for a data set consisting of 100 page-response-time measurements, you would sort the measurements from largest to smallest and then count down six data points from the largest. The 6th data point value represents the 95th percentile of those measurements. For the purposes of response times, this statistic is read “95 percent of the simulated users experienced a response time of the 6th-slowest value or less for this test scenario.”

It is important to note that percentile statistics can only stand alone when used to represent data that is uniformly or normally distributed with an acceptable number of statistical outliers. To illustrate this point, consider the exemplar data sets. The 95th percentile of Data Set B is 16 seconds. Obviously, this does not give the impression of achieving the 5-second response time goal. Interestingly, this can be misleading as well because the 80th percentile value of Data Set B is 1 second. With a response time goal of 5 seconds, it is likely unacceptable to have any response times of 16 seconds, so in this case neither of these percentile values represent the data in a manner that is useful to summarizing response time.

Data Set A is a normally distributed data set that has a 95th percentile value of 6 seconds, an 85th percentile value of 5 seconds, and a maximum value of 7 seconds. In this case, reporting either the 85th or 95th percentile values represents the data in a manner where the assumptions a stakeholder is likely to make about the data are likely to be appropriate to the data.

### 6.2.3 Medians

A median is simply the middle value in a data set when sequenced from lowest to highest. In cases where there is an even number of data points and the two center values are not the same, some disciplines suggest that the median is the average of the two center data points, while others suggest choosing the value closer to the average of the entire set of data. In the case of the exemplar data sets, Data Sets A and B have median values of 4, and Data Set C has a median value of 1.

### 6.2.4 Normal Values

A normal value is the single value that occurs most often in a data set. Data Set A has a normal value of 4, Data Set B has a normal value of 3, and Data Set C has a normal value of 1.

### 6.2.5 Standard Deviations

By definition, one standard deviation is the amount of variance within a set of measurements that encompasses approximately the top 68 percent of all measurements in the data set; in other words, knowing the standard deviation of data set tells you how densely the data points are clustered around the mean. Simply put, the smaller the standard deviation, the more consistent the data. To illustrate, the standard deviation of Data Set A is approximately 1.5, the standard deviation of Data Set B is approximately 6.0, and the standard deviation of Data Set C is approximately 2.6.

A common rule in this case is: “Data with a standard deviation greater than half of its mean should be treated as suspect. If the data is accurate, the phenomenon the data represents is not displaying a normal distribution pattern.” Applying this rule, Data Set A is likely to be a reasonable example of a normal distribution; Data Set B may or may not be a reasonable representation of a normal distribution; and Data Set C is undoubtedly not a reasonable representation of a normal distribution.

### 6.2.5 Uniform Distributions

Uniform distributions - sometimes known as linear distributions - represent a collection of data that is roughly equivalent to a set of random numbers evenly spaced between the upper and lower bounds. In a uniform distribution, every number in the data set is represented approximately the same number of times. Uniform distributions are frequently used when modeling user delays, but are not common in response time results data. In fact, uniformly distributed results in response time data may be an indication of suspect results.

### 6.2.6 Normal Distributions

Also known as bell curves, normal distributions are data sets whose member data are weighted toward the center (or median value). When graphed, the shape of the “bell” of normally distributed data can vary from tall and narrow to short and squat, depending on the standard deviation of the data set. The smaller the standard deviation, the taller and more narrow the “bell.” Statistically speaking, most measurements of human variance result in data sets that are normally distributed. As it turns out, end-user response times for Web applications are also frequently normally distributed.

### 6.2.7 Statistical Significance

Mathematically calculating statistical significance, or reliability, based on sample size is a task that is too arduous and complex for most commercially driven software development projects. Fortunately, there is a commonsense approach that is both efficient and accurate enough to identify the most significant concerns related to statistical significance. Unless you have a good reason to use a mathematically rigorous calculation for statistical significance, a commonsense approximation is generally sufficient.

There is no way to avoid arbitrariness in the final decision as to what level of significance will be treated as really ‘significant.’ That is, the selection of some level of significance, up to which the results will be rejected as invalid, is arbitrary.

Typically, it is fairly easy to add iterations to performance tests to increase the total number of measurements collected; the best way to ensure statistical significance is simply to collect additional data if there is any doubt about whether or not the collected data represents reality. Whenever possible, ensure that you obtain a sample size of at least 100 measurements from at least two independent tests. Although there is no strict rule about how to decide which results are statistically similar without complex equations that call for huge volumes of data that commercially driven software projects rarely have the time or resources to collect, the following is a reasonable approach to apply if there is doubt about the significance or reliability of data after evaluating two test executions where the data was expected to be similar. Compare results from at least five test executions and apply the rules of thumb below to determine whether or not test results are similar enough to be considered reliable:

1. If more than 20 percent (or one out of five) of the test-execution results appear not to be similar to the others, something is generally wrong with the test environment, the application, or the test itself.

2. If a 90th percentile value for any test execution is greater than the maximum or less than the minimum value for any of the other test executions, that data set is probably not statistically similar.

3. If measurements from a test are noticeably higher or lower, when charted side-byside, than the results of the other test executions, it is probably not statistically similar.

4. If one data set for a particular item (e.g., the response time for a single page) in a test is noticeably higher or lower, but the results for the data sets of the remaining items appear similar, the test itself is probably statistically similar (even though it is probably worth the time to investigate the reasons for the difference of the one dissimilar data set.

### 6.2.8 Statistical Equivalence

The method above for determining statistical significance actually is applying the principle of statistical equivalence. Essentially, the process outlined above for determining statistical significance could be restated as “Given results data from multiple tests intended to be equivalent, the data from any one of those tests may be treated as statistically significant if that data is statistically equivalent to 80 percent or more of all the tests intended to be equivalent.” Mathematical determination of equivalence using such formal methods as chi-squared and t-tests are not common on commercial software development projects. Rather, it is generally deemed acceptable to estimate equivalence by using charts similar to those used to determine statistical significance.

### 6.2.9 Statistical Outliers

From a purely statistical point of view, any measurement that falls outside of three standard deviations, or 99 percent, of all collected measurements is considered an outlier. The problem with this definition is that it assumes that the collected measurements are both statistically significant and distributed normally, which is not at all automatic when evaluating performance test data.

Outliers are atypical, infrequent observations: data points which do not appear to follow the distribution of the rest of the sample. These may represent consistent but rare traits, or be the result of measurement errors or other anomalies which should not be modeled.

Note that this (or any other) description of outliers only applies to data that is deemed to be a statistically significant sample of measurements. Without a statistically significant sample, there is no generally acceptable approach to determining the difference between an outlier and a representative measurement.

Using this description, results graphs can be used to determine evidence of outliers - occasional data points that just don’t seem to belong. A reasonable approach to determining if any apparent outliers are truly atypical and infrequent is to re-execute the tests and then compare the results to the first set. If the majority of the measurements are the same, except for the potential outliers, the results are likely to contain genuine outliers that can be disregarded. However, if the results show similar potential outliers, these are probably valid measurements that deserve consideration.

After identifying that a dataset appears to contain outliers, the next question is, how many outliers can be dismissed as “atypical infrequent observations?”

There is no set number of outliers that can be unilaterally dismissed, but rather a maximum percentage of the total number of observations. Applying the spirit of the two definitions above, a reasonable conclusion would be that up to 1 percent of the total values for a particular measurement that are outside of three standard deviations from the mean are significantly atypical and infrequent enough to be considered outliers.

In summary, in practice for commercially driven software development, it is generally acceptable to say that values representing less than 1 percent of all the measurements for a particular item that are at least three standard deviations off the mean are candidates for omission in results analysis if (and only if) identical values are not found in previous or subsequent tests. To express the same concept in a more colloquial way: obviously rare and strange data points that can’t immediately be explained, account for a very small part of the results, and are not identical to any results from other tests are probably outliers.

A note of caution: identifying a data point as an outlier and excluding it from results summaries does not imply ignoring the data point. Excluded outliers should be tracked in some manner appropriate to the project context in order to determine, as more tests are conducted, if a pattern of concern is identified in what by all indications are outliers for individual tests.

### 6.2.10 Confidence Intervals

Because determining levels of confidence in data is even more complex and time consuming than determining statistical significance or the existence of outliers, it is extremely rare to make such a determination during commercial software projects. A confidence interval for a specific statistic is the range of values around the statistic where the ‘true’ statistic is likely to be located within a given level of certainty.

Because stakeholders do frequently ask for some indication of the presumed accuracy of test results - for example, what is the confidence interval for these results? - another commonsense approach must be employed.

When performance testing, the answer to that question is directly related to the accuracy of the model tested. Since in many cases the accuracy of the model cannot be reasonably determined until after the software is released into production, this is not a particularly useful dependency. However, there is a way to demonstrate a confidence interval in the results.

By testing a variety of scenarios, including what the team determines to be “best,” “worst,” and “expected” cases in terms of the measurements being collected, a graphical depiction of a confidence interval can be created, similar to the one below.

**Chart

Description automatically generated**

In this graph, a dashed line represents the performance goal, and the three curves represent the results from the worst-case (most performance-intensive), best-case (least performance-intensive), and expected-case user community models. As one would expect, the solid curve from the expected case falls between the best- and worst-case curves. Observing where these curves cross the red line, one can see how many users can access the system in each case while still meeting the stated performance goal. If the team is 95-percent confident (by their own estimation) that the best- and worst-case user community models are truly best- and worst-case, this chart can be read as follows: the tests show, with 95-percent confidence, that between 100 and 200 users can access the stem while experiencing acceptable performance. it be 100-percent confident that the test results accurately represent the model being tested.

# 7. TOOL SELECTION APPROACH

## 7.1 Performance Testing Tool Architecture

Automated performance test tools typically have the following components:

### 7.1.1 Scripting module

Enables the recording of end-user activity and may support many different middleware protocols. Allows modification of the recorded scripts to associate internal/ external data and configure granularity of response-time measurement. The term middleware refers to the primary protocol used by the application to communicate between the client and first server tier (for web applications this is principally HTTP or HTTPS).

### 7.1.2 Test management module

Allows the creation and execution of performance test sessions or scenarios that represent different mixes of end-user activity. These sessions make use of nominated scripts and one or more load injectors depending on the volume of load required.

### 7.1.3 Load injector(s)

Generates the load—normally from multiple workstations or servers, depending on the amount of load required. Each load injector generates multiple “virtual” users, simulating a large amount of end-user activity from a relatively small number of physical or virtual machines. The application client memory and CPU footprint can have a significant impact on the number of virtual users that can run within a given injector platform, affecting the number of injectors required.

### 7.1.4 Analysis module

Provides the ability to analyze the data collected from each test execution. This data is typically a mixture of autogenerated reports and configurable graphical or tabular presentation. There may also be an expert capability that provides automated analysis of results and highlights areas of concern.

### 7.1.5 Optional modules

Complements the aforementioned components to monitor server and network performance while a load test is running or allow integration with another vendor’s software.

## 7.2 Choosing a Performance Testing Tool

Many performance testing projects run into problems during the scripting stage due to insufficient technical evaluation of the tool being used. Most testing service providers keep a toolbox of solutions from a range of vendors, which allows them to choose the most appropriate performance testing tool for a particular performance testing requirement.

With the current predominance of web technology, every serious tool vendor provides HTTP/S support. However, there are a lot of subtleties to web design, particularly at the client end, so if you make heavy use of JavaScript, ActiveX or Microsoft Silverlight, for example, then be very certain you understand the capabilities and limitations of support in the tools that you shortlist.

### 7.2.1 Considerations

**Protocol support:** The most important single requirement when you are choosing a performance testing tool is to ensure that it supports your application tech stack, specifically how the application client talks to the next application tier. For example, in most cases a typical browser client will be using HTTP or HTTPS as its primary communication protocol.

**Licensing model:** Having satisfied yourself on the tech-stack support, look next at the licensing model. Most performance testing tool vendors offer a licensing model based on the following components:

* The largest load test you can execute in terms of virtual users
* Additional protocols that the tool can support
* Additional plug-ins for integration and specific tech-stack monitoring

**Scripting effort:** Many performance tool vendors claim that there is little or no need to make manual changes to the scripts that their tools generate. This may be true for simple browser use cases that navigate around a couple of pages, but the reality is that Performance Engineer will have to delve into code at some point during scripting

**Solution versus load testing tool:** Some vendors offer only a load testing tool, whereas others offer a performance testing solution. Solution offerings will inevitably cost more but generally allow for a much greater degree of granularity in the analysis they provide. In addition to performance testing, they may include any or all of the following:

* automated requirements management
* automated data creation and management
* pre-performance test application tuning and optimization
* response-time prediction and capacity modeling
* APM providing analysis to class and method level
* integration with end-user experience (EUE) monitoring after deployment
* dashboarding to provide visibility of test results and test assets

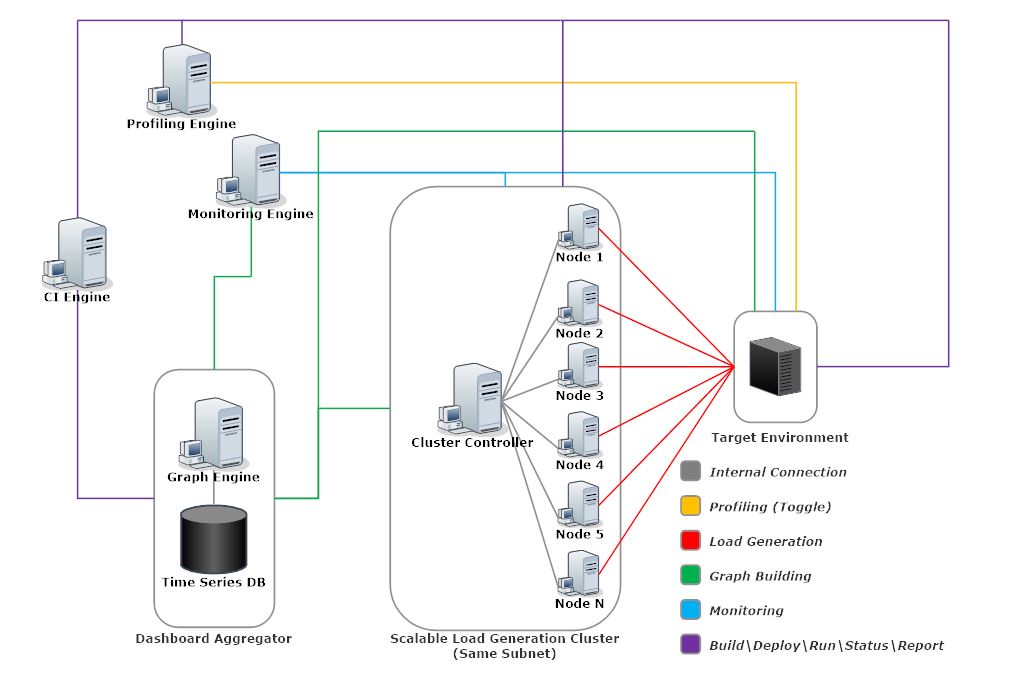
**In-house versus outsourced:** If you have limited resources internally or are working within strict time constraints, consider outsourcing the performance testing project to an external vendor. Some tool vendors offer a complete range of services, from tool purchase with implementation assistance to a complete performance testing service, and there are many companies that specialize in testing and can offer the same kind of service using whatever performance toolset is considered appropriate. This has the advantage of moving the commercials to time and materials and removes the burden of selecting the right testing tool for your application. The main disadvantage of this approach is that if you need to carry out frequent performance tests, the cost per test may rapidly exceed the cost of buying an automated performance testing solution outright.

**The alternatives:** If your application is web-based and customer-facing, you might want to consider one of a growing number of Software as a Service (SaaS) vendors who provide a completely external performance testing service. Essentially, they manage the whole process for you, scripting key use cases and delivering the load via multiple remote points of presence. This approach has the advantages of removing the burden of providing enough injection hardware and of being a realistic way of simulating user load, since these vendors typically connect directly to the Internet backbone via large-capacity pipes. This is an attractive alternative for one-off or occasional performance tests, particularly if high volumes of virtual users are required. These vendors often provide an EUE monitoring service that may also be of interest to you.

The downside is that you will have to pay for each test execution, and any server or network KPI monitoring will be your responsibility to implement and synchronize with the periods of performance test execution. This could become an expensive exercise if you find problems with your application that require many test reruns to isolate and correct.

## 8. PERFORMANCE MANAGEMENT FRAMEWORK ARCHITECTURE

The following architecture concept is very flexible and isn’t tool dependent. The idea is to have a modular structure with variety of possible tooling options



**Target environment.** Test environment should be configured in such a way that it mirrors production environment as closely as possible, noting and accounting for all differences between the two. Build deployment should be automated.

**Scalable load generation cluster.** Perfectly cluster should handle some room for growth over existing project needs. One of solutions is dynamical creation of AWS cluster of spot instances with containerized preconfigured load generators.

*Suggested load tools:*

The Apache JMeter™ application is open source software, based on Java, designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

**Monitoring engine.** This module is responsible for resource utilization monitoring of target environment and load generation cluster.

*Suggested tools:*

Telegraf is a lightweight, non-license collector for gathering metrics from some of the most popular systems, apps, and services like Docker, NoSQL and more. The extensible architecture allows creation of custom integrations without limits on the kind of metrics to collect. Well integrated with time series database InfluxDB.

**Profiling engine.** This module is responsible for profiling of target environment and could be toggled when deep research is required.

*Suggested tools:* TBD

**Results visualization.** This module allows variety of options for flexible graphing, unavailable out-of-the-box for load generation tools. It consists of Graph Engine and Time-Series DB. It collects data from target environment, load generation cluster and monitoring engine. Integration with profiling engine is possible, but optional

*Suggested tools:*

Grafana is visualizing engine most commonly used for visualizing time series data. Grafana features pluggable panels and data sources allowing easy extensibility and a variety of panels, including fully featured graph panels with rich visualization options. Also it`s supports real-time system management and alerting.

InfluxDB is an open source database specifically to handle time series data with high availability and high performance requirements.

**CI Engine.** This module is responsible for building and deployment of environments and tests, running tests and observing test status.

*Suggested tools:* Jenkins

| REVISION HISTORY | | | | | |
| --- | --- | --- | --- | --- | --- |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Draft | Denys Hurt  [dhurt@copyright.com](http://dhurt@copyright.com) | 06/11/2021 |  |  |
| 1.1 | Test Plan references were added | Denys Hurt  [dhurt@copyright.com](http://dhurt@copyright.com) | 07/08/2021 |  |  |