Information Technology & Telecommunications

Citywide Policy for Performance Testing of Public-Facing Applications

Final 2.0 - PUBLIC

2/2/2016

City of New York
Department of Information Technology and Telecommunications
Application Development Management – Quality Assurance

Table of Contents

1.0 Overview 3
1.1 Introduction 3
1.2 Audience 3
1.3 Purpose 3
1.4 Scope 4
2.0 Policy 4
2.1 Policy Statement 4
2.2 Performance Testing Requirements 4
2.2.1 Required Performance Tests 4
2.2.2 Required Performance Standards 5
2.2.3 Required Activities for Performance Testing 5
2.2.4 Performance Testing Tools 5
2.3 Roles and Responsibilities 5
2.3.1 If DoITT Conducts Performance Testing 5
2.3.2 If the City Agency or Third Party Conducts Performance Testing 6
3.0 Authority 7
4.0 Ownership and Contact 7
5.0 Change History 7
6.0 Appendix A: Entry / Exit Criteria 8
7.0 Appendix B: Required Activities for Performance Testing 10
8.0 Appendix C: DoITT Performance Test Services Questionnaire 18
9.0 Appendix D: Performance Test Results Sample Sheet 19
10.0 Appendix E: Risks Addressed by Performance Test Types 20
11.0 Appendix F: Risk Types Addressed by Performance Tests 21
12.0 Appendix G: Glossary 23

1.0 Overview

1.1 Introduction

This document is based on information technology (IT) industry standards and best practices. It has been developed to guide Citywide agencies in the performance testing of applications. The primary tasks of application performance testing activities are to validate application stability and scalability, and to collect relevant information to help stakeholders make informed decisions related to the overall quality of the application being tested.

Performance testing also helps to identify bottlenecks in a system, establish a baseline for future testing, and determine fulfillment of performance goals and requirements. In addition, analysis of performance testing results can help to estimate the hardware and software configurations required to support an application when it "GOES LIVE" to production. For these reasons, performance testing is strongly recommended for all applications.

Specifically for any applications that face the public however, performance testing is mandatory in all circumstances, including mobile applications serving multiple users through the connection to server-based application infrastructure. Some of the reasons performance testing is required include, but are not limited to:

- Poorly performing applications damage the City's IT reputation, regardless of what team or agency developed them.
- Public-facing applications may attract far greater public traffic than the intended user base, which may cause unexpected load increases. This is especially the case when applications are covered by the media through public announcements, news outlets, or other public communications vehicles.

By providing definite standards for compliance, this document requires that this important part of pre-deployment is performed consistently and reliably for all public-facing City applications.

1.2 Audience

The document is written for City agency project managers and quality assurance (QA) staff that will be responsible for the performance testing of public-facing systems and applications before deployment. It is intended both for City agency employees and for external contractors, consultants, and business partners, including architects, system integrators, or technical leads working at agencies.

1.3 Purpose

The purpose of this document is to define the policy and standards to be followed during performance testing of all public-facing applications.

1.4 Scope

a) The policies and standards in this document apply to all City agencies. An agency is defined as an administrative unit of City government as designated in the New York City Charter, by Executive Order or by Local Law.

b) The policies and standards in this document apply to all new or modified public-facing systems and applications. This document can also be used as guidance for testing internal applications, although such tests are not mandated by this policy.

2.0 Policy

2.1 Policy Statement

a) All Citywide public-facing applications, including mobile applications, must be subject to performance testing that meets the requirements in this document (Section 2.2).

b) A City agency can use its own QA services, enlist a third party, or leverage DoITT QA services to conduct performance testing. However, in every case, all performance testing for Citywide public-facing applications must be approved by DoITT prior to deployment, contingent on DoITT's validation of the testing procedures and exit criteria. In any scenario, the DoITT QA team is equipped to serve as a guiding resource toward meeting performance testing requirements.

2.2 Performance Testing Requirements

2.2.1 Required Performance Tests

The expected load of an application is determined during the application's business analysis phase. Once that expected load is established, the following tests are mandatory for all public-facing applications:

- **Stress Test:** A Stress Test is executed to determine if the application will perform sufficiently if its load goes well above the expected maximum. It helps application administrators determine the application's robustness, availability, and error handling under heavy load scenarios, such as in extreme load. It must be executed with at least 3 hours of steady state run (with ramp up/ramp down time excluded) with an applied load of at least 120% of the estimated expected load.

- **Stability/Soak/Endurance Test:** Soak Testing is usually done to determine if the application can sustain the continuous expected load without performance getting degraded over time. The applied load can be the same as with the stress test or lower, but must be at least equal to the expected load. The test must run for at least 12 hours in a steady state.

- Note: For more technical detail, please refer to Appendix A for entry and exit criteria on performance testing.

The following test is not mandatory, but highly advisable:

- **Breakpoint Test:** The goal of the Breakpoint Test is to determine the maximum load the system can support. It is usually done by gradually increasing load and it continues to run until the system's behavior reaches an unacceptable level (e.g. significant increase of response time; CPU usage nears 100%, etc.). The Breakpoint Test is recommended to be run prior to a Stress Test.

- Note: Performance testing is one of the several activities necessary to deploy an application. Further resources and templates are available on the NYC Project site.

2.2.2 Required Performance Standards

DoITT standards for performance testing must be met according to the Entry and Exit Criteria outlined in Appendix A.

If DoITT QA does not perform the testing itself, it must validate that the Exit Criteria have been met in order to approve any testing conducted by a City agency or third party before the application is deployed.

2.2.3 Required Activities for Performance Testing

Detailed descriptions and suggested guidelines for these activities can be found in Appendix B. If DoITT QA does not perform the testing itself, it must review the testing scenarios and validate that the required activities have been conducted by the City agency or third party doing the test before the application is deployed.

2.2.4 Performance Testing Tools

DoITT maintains test infrastructure and test tools to conduct performance testing for agencies. These resources are also available to agencies choosing to conduct their own testing. If an agency chooses to use alternative tools, these tools are expected to include 1) monitoring and reporting capabilities necessary to conduct the required performance testing activities; and 2) capabilities to collect and report necessary data sufficient to decide if exit criteria are met as outlined in Appendix A.

2.3 Roles and Responsibilities

2.3.1 If DoITT Conducts Performance Testing

Efficiencies are gained at both the agency and the City level by leveraging DoITT's centralized QA testing service. City agencies can leverage DoITT skillsets and existing licenses. Additionally, the cost of tools, training, and testing environments can be mitigated.

DoITT: DoITT will conduct required performance tests through its testing services.

City Agency: Agencies can submit a request for these services directly to DoITT. The agency will be required to complete a brief questionnaire such as found in Appendix C.

*IMPORTANT: Because DoITT performance testing resources are limited, agencies should contact DoITT **six weeks before Go-Live** to give DoITT's QA team enough time to meet agency requirements and expectations.*

Agencies MUST NOT publicly commit to an application launch date until completion of the performance testing and DoITT QA signoff.

2.3.2 If the City Agency or Third Party Conducts Performance Testing

These are the required responsibilities in the event a City agency decides to conduct its own performance testing or uses a third party vendor.

DoITT: DoITT will review and approve performance testing as conducted by the City agency or third party prior to deployment. DoITT remains on-hand with expertise, tools, and resources to aid any City agency in its effort to conduct performance testing.

City agency: The City agency must first notify DoITT that it plans to conduct performance testing on a public-facing application by contacting DoITT QA Services.

The City agency or hired third party must adhere to the required tests, activities, and standards in this policy (section 2.2).

The following documentation must be submitted to DoITT **six weeks before Go-Live** so that it can review and approve the application prior to deployment:

a) Brief description of application (or provide demo)
b) Description of application environment and infrastructure (infrastructure diagram is sufficient)
c) Description of the test approach, expected load and test scenarios (test scenario should follow guidelines in Appendix B), list of transactions to be measured for performance
d) Request for the DoITT performance testing access (if DoITT toolset is needed); OR a description of the tools being used (name, license type, infrastructure description)
e) Performance test results provided in DoITT accepted format, sufficient to validate that the application satisfies performance requirements. See Appendix D for sample sheet and explanation.

If the required tests have been run, the required activities have been followed, and the test results show that the application performance is satisfactory, DoITT QA will sign off. This approval must be achieved before the application is deployed.

Engaging DoITT early and often for guidance and resources during the performance testing process will reduce risk that the application will not receive DoITT QA signoff.

*IMPORTANT: Agencies should submit performance test results and documentation to DoITT QA **six weeks before Go-Live** to give DoITT's to allow adequate time for review and potential retest.*

Agencies MUST NOT publicly commit to an application launch date until completion of the performance testing and DoITT QA signoff.

3.0 Authority

The New York City Department of Information Technology and Telecommunications (DoITT) was established by Local Law 24 of 1995 as "New York City's information technology and telecommunications agency."

Chapter 48 of the New York City Charter established the authority of DoITT by assigning powers and duties "to plan, formulate, coordinate and advance information technology and telecommunications policies for the city."

Executive Order No.140 of 2010 directed DoITT to "be responsible for establishing and enforcing Citywide IT policies and for ensuring that such policies are aligned with the City's business needs and investments, as well as the individual business needs of each agency."

4.0 Ownership and Contact

This policy is owned by DoITT Application Development Management (ADM) Quality Assurance team.

5.0 Change History

Version	Change Highlights	Date
1.0	Final Draft	6-11-12
1.1	Updated Appendix E - Deployment Readiness Checklist	8-14-12
1.2	Updated Entry 10: Deployment Package in Appendix E – Readiness Checklist	10-3-12
1.3	Reformatted and revised	7-1-13
1.4	Public version information added	6-25-14
2.0	Perf testing must be approved by DoITT but not mandated to be conducted by DoITT	2-3-16

6.0 Appendix A: Entry / Exit Criteria

The following standards for performance testing entry and exit criteria must be met for all public-facing applications. If DoITT is not conducting the test, it must confirm that exit criteria have been satisfied prior to application deployment.

Entry Criteria

Performance testing is ready to begin when:

- The application must be near its final build with medium and high priority defects addressed/fixed and at least one iteration of system test pass occurring during the script execution process
- The Staging Environment has been configured
- The appropriate test tools have been set up and ready for test execution
- When DoITT services and tools are used, LoadRunner Controllers and Load Generators, or Performance Center are validated and reserved
- The appropriate performance test scripts have been coded and reviewed

Exit Criteria

Exit criteria out of Performance Test are as follows:

- Execution of the following mandatory tests are fully completed (section 2.2.1):
  - Stress Test
  - Stability/Soak/Endurance Test
- System parameters measured during the tests are as follows:
  - Peak CPU utilization is 65% or lower
  - 40/60 distribution between load balancers
  - Memory utilization should not exceed 80%
  - No memory use build up during the test and immediately after
  - No memory leak recognized after the test is completed
  - Database is not exhausted during the test, no errors occur and it is functioning as expected throughout all the time of the test execution
  - Page response time under 3 seconds on average
  - Page response time 90% under 5 seconds
  - For pages based on forms, Web Services, or build with EMC Documentum:
    - Response time under 30 seconds on average
    - Response time 90% under 50 seconds on average
  - Upload valid picture file type(about 2.92MB size) response time under 30 seconds on average

- Upload valid picture file type(about 2.92MB size) response time 90% under 50 seconds on average
- All defects of priority 1 and 2 have been resolved
- DoITT QA Director and Business Project Managers approved the performance test results

7.0 Appendix B: Required Activities for Performance Testing

The following activities are required when performance testing is done. The key to effectively implementing these activities is applying them in the manner most valuable to each individual project context.

Starting with knowledge of the project context, the QA team must begin identifying the test environment and the performance acceptance criteria more or less in parallel. This is due to the fact that all the remaining activities are affected by the information gathered in these first two activities. Generally, the team will revisit these activities periodically as the team learns more about the application, its users, its features, and any performance-related risks it might have.

Project Context

For performance testing to be successful, the testing itself must be relevant to the context of the project. Without an understanding of the project context, performance testing risks focusing on only those items that the performance tester or test team assumes to be important, as opposed to those that truly are important to the business owner and other stakeholders. This misapplication of focus frequently leads to wasted time, frustration, and conflicts.

The project context is therefore relevant to achieving project success. This may include, but is not limited to:

- The overall vision or intent of the project
- The performance testing objectives
- The performance success criteria
- The development life cycle
- The project schedule
- The project budget
- The available tools and environments
- The skill set of the performance tester and the team
- The priority of detected performance concerns
- The business impact of deploying an application that performs poorly.

Defining the test scope

Performance testing tools and infrastructure can be used for the single user application on any platform; however, the focus of this policy is performance testing of complex applications with *n*-tier architecture serving multiple users simultaneously. When applications of this type respond to users' actions, the function and resource use of all application components and

infrastructure may vary depending on the number of users acting simultaneously. As a result of that the response time may vary as well.

There are several infrastructure and application architecture components affecting user experience for the *n*-tier multiuser application. These may include, but are not limited to the following:

- User workstation, mobile device, or other server of the other application (for system-to-system connection);
- Any type of network (internal or public, wired or wireless, etc.),
- Multiple ties of servers including web, application and database servers and load balancers

This policy and DoITT QA performance testing services focus on measuring **response time** and **resource use** as well as proper functioning of the server tier of the application architecture, which applies mainly to load balancing. Any other application components (e.g. end user device or network) are simulated by the performance test infrastructure and are not the subject of the test.

To prepare the test, a tester creates and customizes a script that simulates the load provided by end-user equipment, and then uses this script during test execution to generate the appropriate amount of load. The load simulation can be based on the data recorded by specialized recording tools or in some cases can be generated by creating custom code for the simulation.

To execute the server performance tests mentioned above, the process of applying the load during the test does not depend on the computer or device where the load was originally recorded. The load is applied from the computers called Load Generators, which are part of the reusable performance testing infrastructure and is maintained by performance testing team.

Identify the Test Environment

The team must identify the test environment to completely understand the similarities and differences between the test and production environments. Some critical factors to consider are:

- **Hardware**
  - Configurations
  - Machine hardware (processor, RAM, etc.)
- **Network**
  - Network architecture and end-user location
  - Load-balancing implications
  - Cluster and Domain Name System (DNS) configurations
- **Tools**

  - Load-generation tool limitations
  - Environmental impact of monitoring tools
- **Software**
  - Other software installed or running in shared or virtual environments
  - Software license constraints or differences
  - Storage capacity and seed data volume
  - Logging levels
- **External factors**
  - Volume and type of additional traffic on the network
  - Scheduled or batch processes, updates, or backups
  - Interactions with other systems

Further guidelines when identifying the test environment include:

- Identify the amount and type of data the application must be seeded with to emulate real-world conditions.
- Identify critical system components. Do any of the system components have known performance concerns? Are there any integration points that are beyond the team's control for testing?
- Check the configuration of load balancers.
- Validate name resolution with DNS. This may account for significant latency when opening database connections.
- Validate that firewalls, DNS, routing, and so on treat the generated load similarly to a load that would typically be encountered in a production environment.

Identify Performance Acceptance Criteria

Classes of characteristics that frequently correlate to a user's or stakeholder's satisfaction typically include:

- **Response time.** For example, the product catalog must be displayed in less than three seconds.
- **Throughput**. For example, the system must support 25 payments per second.
- **Resource utilization**. For example, processor utilization is not more than 75 percent. Other important resources that need to be considered for setting objectives are memory, disk input/output (I/O), and network I/O.

Consider the following when identifying performance acceptance:

- Business requirements

- User expectations
- Contractual obligations
- Regulatory compliance criteria and industry standards
- Service Level Agreements (SLAs)
- Resource utilization targets
- Various and diverse, realistic workload models
- The entire range of anticipated load conditions

Plan and Design the Test

Planning and designing performance tests involves identifying key usage scenarios, determining appropriate variability across users, identifying and generating test data, and specifying the metrics to be collected.

The team's goal should be to create real-world simulations in order to provide reliable data that will enable the agency to make informed business decisions. Real-world test designs will significantly increase the relevancy and usefulness of results data.

Key usage scenarios for the application typically surface during the process of identifying the desired performance characteristics. If this is not the case for the test project under evaluation, the team will need to explicitly determine the usage scenarios that are the most valuable to script. The team must consider the following when identifying key usage scenarios:

- Contractually obligated usage scenario(s)
- Usage scenario(s) implied or mandated by performance-testing goals and objectives
- Most common usage scenario(s)
- Business-critical usage scenario(s)
- Performance-intensive usage scenario(s)
- Usage scenario(s) of technical concern
- Usage scenario(s) of stakeholder concern
- High-visibility usage scenario(s)

The team must consider the following when planning and designing tests:

- Realistic test designs are sensitive to dependencies outside the control of the system, such as humans, network activity, and other systems interacting with the application.
- Realistic test designs are based on what the team expects to find in real-world use, not theories or projections.

- Realistic test designs produce more credible results and thus enhance the value of performance testing.
- Extrapolating performance results from unrealistic tests can create damaging inaccuracies as the system scope increases and frequently leads to poor decisions.

Implement the Test Design

The details of creating an executable performance test are extremely tool-specific. Regardless of the tool being using, creating a performance test typically involves scripting a single usage scenario and then enhancing that scenario and combining it with other scenarios to ultimately represent a complete workload model.

The biggest challenge involved in a performance-testing project is getting the first relatively realistic test implemented with users generally being simulated in such a way that the application under test cannot legitimately tell the difference between the simulated users and real users.

The team must consider the following when implementing the test design:

- Ensure that test data feeds are implemented correctly. Test data feeds are data repositories in the form of databases, text files, in-memory variables, or spreadsheets that are used to simulate parameter replacement during a load test. For example, even if the application database test repository contains the full production set, the load test might only need to simulate a subset of products being bought by users due to a scenario involving, for example, a new product or marketing campaign. Test data feeds may be a subset of production data repositories.
- Ensure that application data feeds are implemented correctly in the database and other application components. Application data feeds are data repositories, such as product or order databases, that are consumed by the application being tested. The key user scenarios, run by the load test scripts may consume a subset of this data.
- Ensure that validation of transactions is implemented correctly. Many transactions are reported successful by the Web server, but they fail to complete correctly. Examples of validation are, database entries inserted with correct number of rows, product information being returned, correct content returned in html data to the clients etc.
- Ensure hidden fields or other special data are handled correctly. This refers to data returned by Web server that needs to be resubmitted in subsequent request, like session IDs or product ID that needs to be incremented before passing it to the next request.

Execute the Test

It makes sense that the process, flow, and technical details of test execution are extremely dependent on the tools, environment, and project context. Even so, there are some fairly universal tasks and considerations that need to be kept in mind when executing tests.

Test execution can be viewed as a combination of the following sub-tasks. The QA team must:

1. Coordinate test execution and monitoring with all appropriate members of the team.
2. Validate test configurations, and the state of the environments and data.
3. Begin test execution.
4. While the test is running, monitor and validate scripts, systems, and data.
5. Upon test completion, quickly review the results for obvious indications that the test was flawed.
6. Archive the tests, test data, results, and other information necessary to repeat the test later if needed.
7. Log start and end times, the name of the result data, and so on. This will allow the team to identify data sequentially after the test is done.

As the team prepares to begin test execution, it is worth taking the time to double-check the following items. The team must:

- Validate that the test environment matches the configuration that the team was expecting and/or designed the test for.
- Ensure that both the test and the test environment are correctly configured for metrics collection.
- Before running the real test, execute a quick smoke test to make sure that the test script and remote performance counters are working correctly. In the context of performance testing, a smoke test is designed to determine if the application can successfully perform all of its operations under a normal load condition for a short time.
- Reset the system (unless the scenario calls for doing otherwise) and start a formal test execution.
- Make sure that the test scripts' execution represents the workload model the team wants to simulate.
- Make sure that the test is configured to collect the key performance and business indicators of interest at this time.

The team must consider the following when executing the test:

- Validate test executions for data updates, such as orders in the database that have been completed.
- Validate if the load-test script is using the correct data values, such as product and order identifiers, in order to realistically simulate the business scenario.
- If at all possible, execute every test three times. Note that the results of first-time tests can be affected by loading Dynamic-Link Libraries (DLLs), populating server-side caches, or initializing scripts and other resources required by the code under test. If the results of the second and third iterations are not highly similar, then the test must be executed again. The team must try to determine what factors account for the difference.

Analyze Results, Report, and Retest

Managers and stakeholders need more than just the results from various tests — they need conclusions, as well as consolidated data that supports those conclusions. Technical team members also need more than just results — they need analysis, comparisons, and details behind how the results were obtained. Team members of all types get value from performance results being shared more frequently.

Before results can be reported, the data must be analyzed. The team must consider the following important points when analyzing the data returned by the performance test:

- Analyze the data both individually and as part of a collaborative, cross-functional technical team.
- Analyze the captured data and compare the results against the metric's acceptable or expected level to determine whether the performance of the application being tested shows a trend toward or away from the performance objectives.
- If the test fails, a diagnosis and tuning activity are generally warranted.
- If any bottlenecks are fixed, repeat the test to validate the fix.
- Performance-testing results will often enable the team to analyze components at a deep level and correlate the information back to the real world with proper test design and usage analysis.
- Performance test results should enable informed architecture and business decisions.
- Frequently, the analysis will reveal that, in order to completely understand the results of a particular test, additional metrics will need to be captured during subsequent test-execution cycles.
- Immediately share test results and make raw data available to the entire team.
- Talk to the consumers of the data to validate that the test achieved the desired results and that the data means what the team thinks it means.
- Modify the test to get new, better, or different information if the results do not represent what the test was defined to determine.

- Use current results to set priorities for the next test.
- Collecting metrics frequently produces very large volumes of data. Although it is tempting to reduce the amount of data, the team must always exercise caution when using data-reduction techniques because valuable data can be lost.

Most reports fall into one of the following two categories:

- Technical Reports
  - Description of the test, including workload model and test environment
  - Easily digestible data with minimal pre-processing
  - Access to the complete data set and test conditions
  - Short statements of observations, concerns, questions, and requests for collaboration
- Stakeholder Reports
  - Criteria to which the results relate
  - Intuitive, visual representations of the most relevant data
  - Brief verbal summaries of the chart or graph in terms of criteria
  - Intuitive, visual representations of the workload model and test environment
  - Access to associated technical reports, complete data sets, and test conditions
  - Summaries of observations, concerns, and recommendations

Summary

Performance testing involves a set of common core activities that occur at different stages of projects. Each activity has specific characteristics and tasks to be accomplished. These activities have been found to be present — or at least to have been part of an active, risk-based decision to omit one of the activities — in every deliberate and successful performance-testing project. It is therefore important to understand each activity in detail and then apply the activities in a way that best fits the project context.

In performing all of the required activities outlined above, and considering all of the suggested guidelines, the testing agent ensures that performance testing of all new or modified public-facing City applications will be consistently and reliably accomplished to high standards. As one important part of pre-deployment QA testing, this helps to make certain that these applications can “GO LIVE” with confidence.

8.0 Appendix C: DoITT Performance Test Services Questionnaire

Project Name:	
Agency:	
Service Requested:	
Agency Project Manager/Contact(s):	
Project URL:	
Requested Start Date:	
Requested Finish Date:	
Request Description:	
Staging Environment Ready:	
Projected # of Users	
Can a demo of the application be given?	
Additional Information:	
Security Accreditation completed:	

9.0 Appendix D: Performance Test Results Sample Sheet

This template is an example of an acceptable form of Stress Test results (and can be adapted for other test results as well). An Excel template can be found here:

Application: xxxx Release No: xxx Stress Test Results				
CPU Usage Report				
Test Execution Date	xx/xx/xxxx	xx/xx/xxxx		
Release	xx	xx		
Execution Cycle	Pass x	Pass x		
# of Users	xxx	xxx		
Servers				
prtl-stg-web1	10-15%	10-15%		
Memory Usage Report				
Test Execution Date	xx/xx/xxxx	xx/xx/xxxx		
Release	xx	xx		
Execution Cycle	Pass x	Pass x		
# of Users	xxx	xxx		
Servers				
prtl-stg-web1				

Other Usage Report				
Date	Load Balance	VM Build Up	Memory Leak	Database Errors
7.26.2006	49.9%/50.1%	No	No	No

Stress Test Results - Transaction Response Times				
Test Execution Date	xx/xx/xxxx	xx/xx/xxxx	xx/xx/xxxx	xx/xx/xxxx
Release	xx	xx	xx	xx
Execution Cycle	Pass x	Pass x	Pass x	Pass x
Build #	xxx	xxx	xxx	xxx
Logging Level	ERROR	ERROR	ERROR	ERROR
# Of Vusers	xxx	xxx	xxx	xxx
Duration	3Hrs	3Hrs	3Hrs	3Hrs
Transaction Name	Average	90 Percent	Average	90 Percent
SCRIPT: Script1				
Application_Script1a_step1				
Application_Script1b_step2				
Application_Script1c_step3				
Application_Script1d_step4				
Application_Script1e_step5				
Application_Script1f_step6				
SCRIPT: Script2				
Application_Script2a_step1				
Application_Script2b_step2				
Application_Script2c_step3				
Application_Script2d_step4				
Application_Script2e_step5				
Application_Script2f_step6				

10.0 Appendix E: Risks Addressed by Performance Test Types

A full glossary of definitions is available in Appendix G.

Performance Test Type	Risk(s) Addressed
Capacity	• Is system capacity meeting business volume under both normal and peak load conditions?
Component	• Is this component meeting expectations? • Is this component reasonably well optimized? • Is the observed performance issue caused by this component?
Endurance (or Soak or Stability Test)	• Will performance be consistent over time? • Are there slowly growing problems that have not yet been detected? • Is there external interference that was not accounted for?
Breakpoint	• How many users can the application handle before undesirable behavior occurs when the application is subjected to a particular workload? • How much data can my database/file server handle? • Are the network components adequate?
Smoke	• Is this build/configuration ready for additional performance testing? • What type of performance testing should I conduct next? • Does this build exhibit better or worse performance than the last one?
Stress	• What happens if the production load exceeds the anticipated load? • What kinds of failures should we plan for? • What indicators should we look for in order to intervene prior to failure?

11.0 Appendix F: Risk Types Addressed by Performance Tests

Risks	Performance test types					
	Capacity	Component	Endurance	Breakpoint	Smoke	Stress
Speed-related risks						
User satisfaction			X	X		X
Synchronicity		X	X	X		X
Service Level Agreement (SLA) violation			X	X		
Response time trend		X	X	X	X	
Configuration			X	X	X	X
Consistency		X	X	X		
Scalability-related risks						
Capacity	X	X	X	X		
Volume	X	X	X	X		
SLA violation			X	X		
Optimization	X	X				
Efficiency	X	X				
Future growth	X	X		X		
Resource consumption	X	X	X	X	X	X
Hardware / environment	X	X	X	X		X

Risks	Performance test types					
	Capacity	Component	Endurance	Breakpoint	Smoke	Stress
Service Level Agreement (SLA) violation	X	X	X	X		
Stability-related risks						
Reliability		X	X	X		X
Robustness		X	X	X		X
Hardware / environment			X	X		X
Failure mode		X	X	X		X
Slow leak		X	X	X		
Service Level Agreement (SLA) violation		X	X	X		X
Recovery		X				X
Data accuracy and security		X	X	X		X
Interfaces		X	X	X		X

12.0 Appendix G: Glossary

The glossary below includes DoITT abbreviations used in this document, performance testing terminology, and terms with specific DoITT Enterprise Architecture definitions.

Term	Definition
Capacity	The *capacity* of a system is the total workload it can handle without violating predetermined key performance acceptance criteria.
Capacity test	A *capacity test* complements load testing by determining the server's ultimate failure point, whereas load testing monitors results at various levels of load and traffic patterns. Capacity testing is performed in conjunction with capacity planning, which is used to plan for future growth, such as an increased user base or increased volume of data. For example, to accommodate future loads, the QA team needs 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 the application should scale up or scale out.
Component test	A *component test* is any performance test that targets an architectural component of the application. Commonly tested components include servers, databases, networks, firewalls, and storage devices.
DEV	Application Development
Endurance test	An *endurance test* (also referred to as *soak* or *stability test*) is a type of performance test focused on determining or validating performance characteristics of the product under test when subjected to workload models and load volumes anticipated during production operations over an extended period of time. Endurance testing is a subset of load testing.
Guideline	A *guideline* is a principle or rule to inform decisions and achieve rational outcomes. Guidelines recommend the use of standards and procedures. While guidelines are not enforced, they are strong recommendations.
Investigation	*Investigation* is an activity based on collecting information related to the speed, scalability, and/or stability characteristics of the product under test that may have value in determining or improving product quality. Investigation is frequently employed to prove or disprove hypotheses regarding the root cause of one or more observed performance issues.
IT Sec Ops	IT Security Operations
Latency	*Latency* is a measure of responsiveness that represents the time it takes to complete the execution of a request. Latency may also represent the sum of several latencies or subtasks.
Metrics	*Metrics* are measurements obtained by running performance tests as expressed on a commonly understood scale. Some metrics commonly obtained through performance tests include processor utilization over time and memory usage by load.
Performance	*Performance* refers to information regarding the application's response times, throughput, and resource utilization levels.

Term	Definition
Performance budgets or allocations	*Performance budgets* (or *allocations*) are constraints placed on developers regarding allowable resource consumption for their component.
Performance goals	*Performance goals* are the criteria that the team wants to meet before product release, although these criteria may be negotiable under certain circumstances. For example, if a response time goal of three seconds is set for a particular transaction but the actual response time is 3.3 seconds, it is likely that the stakeholders will choose to release the application and defer performance tuning of that transaction for a future release.
Performance objectives	*Performance objectives* are usually specified in terms of response times, throughput (transactions per second), and resource-utilization levels and typically focus on metrics that can be directly related to user satisfaction.
Performance requirements	*Performance requirements* are those criteria that are absolutely non-negotiable due to contractual obligations, service level agreements (SLAs), or fixed business needs. Any performance criterion that will not unquestionably lead to a decision to delay a release until the criterion passes is not absolutely required — and therefore, not a requirement.
Performance targets	*Performance targets* are the desired values for the metrics identified for the project under a particular set of conditions, usually specified in terms of response time, throughput, and resource-utilization levels. Resource-utilization levels include the amount of processor capacity, memory, disk I/O, and network I/O that the application consumes. Performance targets typically equate to project goals.
Performance test	A *performance test* is a technical investigation done to determine or validate the speed, scalability, and/or stability characteristics of the product under test. Performance testing is the superset containing all other subcategories of performance testing described in this chapter.
Performance testing objectives	*Performance testing objectives* refer to data collected through the performance-testing process that is anticipated to have value in determining or improving product quality. However, these objectives are not necessarily quantitative or directly related to a performance requirement, goal, or stated quality of service (QoS) specification.
Performance thresholds	*Performance thresholds* are the maximum acceptable values for the metrics identified for the project, usually specified in terms of response time, throughput (transactions per second), and resource-utilization levels. Resource-utilization levels include the amount of processor capacity, memory, disk I/O, and network I/O that the application consumes. Performance thresholds typically equate to requirements.
PM	Project Management
Policy	A *policy* is a mandatory principle or rule to guide decisions and achieve rational outcomes. Policies impose the use of standards and procedures. Policies are enforceable, and may require assertions of compliance.
Procedure	A *procedure* is a process: a series of actions or operations that are executed in the same manner in order to obtain a specific result.

Term	Definition
Prod Support	Production Support
QA	Quality Assurance
Resource utilization	*Resource utilization* is the cost of the project in terms of system resources. The primary resources are processor, memory, disk I/O, and network I/O.
Response time	*Response time* is a measure of how responsive an application or subsystem is to a client request.
Saturation	*Saturation* refers to the point at which a resource has reached full utilization.
Scalability	*Scalability* refers to an application's ability to handle additional workload, without adversely affecting performance, by adding resources such as processor, memory, and storage capacity.
Scenarios	In the context of performance testing, a *scenario* is a sequence of steps in the application. A scenario can represent a use case or a business function such as searching a product catalog, adding an item to a shopping cart, or placing an order.
SCM	Software Configuration Management
Smoke test	A *smoke test* is the initial run of a performance test to see if the application can perform its operations under a normal load.
Spike test	A *spike test* is a type of performance test focused on determining or validating performance characteristics of the product under test when subjected to workload models and load volumes that repeatedly increase beyond anticipated production operations for short periods of time. Spike testing is a subset of stress testing.
Stability	In the context of performance testing, *stability* refers to the overall reliability, robustness, functional and data integrity, availability, and/or consistency of responsiveness for the system under a variety conditions.
Standard	A *standard* is a formally documented statement that establishes uniform technical criteria, methods, procedures, and/or practices.
Stress test	A *stress test* is a type of performance test designed to evaluate an application's behavior when it is pushed beyond normal or peak load conditions. The goal of stress testing is to reveal application bugs that surface only under high load conditions. These bugs can include such things as synchronization issues, race conditions, and memory leaks. Stress testing enables the QA team to identify the application's weak points, and shows how the application behaves under extreme load conditions.
Throughput	*Throughput* is the number of units of work that can be handled per unit of time; for instance, requests per second, calls per day, hits per second, reports per year, etc.
UAT	User Acceptance Test

Term	Definition
Unit test	In the context of performance testing, a *unit test* is any test that targets a module of code where that module is any logical subset of the entire existing code base of the application, with a focus on performance characteristics. Commonly tested modules include functions, procedures, routines, objects, methods, and classes. Performance unit tests are frequently created and conducted by the developer who wrote the module of code being tested.
Utilization	In the context of performance testing, *utilization* is the percentage of time that a resource is busy servicing user requests. The remaining percentage of time is considered idle time**.**
Validation test	A *validation test* compares the speed, scalability, and/or stability characteristics of the product under test against the expectations that have been set or presumed for that product.
Workload	*Workload* is the stimulus applied to a system, application, or component to simulate a usage pattern, in regard to concurrency and/or data inputs. The workload includes the total number of users, concurrent active users, data volumes, and transaction volumes, along with the transaction mix. For performance modeling, the QA team associates a workload with an individual scenario.