CSS PERFORMANCE TEST STRATEGY QUICK REFERENCE GUIDE

1 Introduction

Understanding of test concepts is core to the development and fielding of quality products. The purpose of this document is to present a consolidated Quick Reference Guide (QRG) that includes the definitions of key test concepts[1]. This QRG is part of CSS' Integrated Management System (CSS IMS), and is available to our teams engaged in service delivery to CSS clients.

1.1 Performance Testing

A non-functional software testing technique that determines how the stability, speed, scalability, and responsiveness of an application holds up under a given workload. This is achieved through covering key metrics using different testing types. Importance of key metrics will differ per project and stage of the project.

1.2 Test Strategy

A high-level document that outlines the approach and principles for testing a software application. It serves as a blueprint for the testing process, ensuring that all testing activities are aligned with the project's objectives and quality standards.

1.2.1 Test Strategy Key Aspects

- **Scope and Objectives:** Defines what will be tested and the goals of the testing effort.
- **Testing Approach:** Describes the overall approach to testing, including the types of testing (e.g., functional, performance, security) and the techniques to be used.
- **Test Environment:** Specifies the hardware, software, and network configurations required for testing.
- **Resource Allocation:** Details the resources needed, including personnel, tools, and training.
- **Schedule and Milestones:** Outlines the timeline for testing activities, including key milestones and deadlines.
- **Risk Management:** Identifies potential risks and mitigation strategies.

[1] LinkedIn Learning: Online Courses for Creative, Technology, Business Skills for more in depth coverage on building a performance test strategy and the related topics included in this Quick Reference Guide (QRG).

- **Entry and Exit Criteria:** Defines the conditions that must be met to start and stop testing.
- A well-defined test strategy helps ensure that the testing process is thorough, efficient, and consistent, leading to a higher quality software product.

1.3 Automation Test Framework

A set of guidelines, best practices, and tools designed to support automated software testing. It provides a structured approach to creating, executing, and managing automated tests, ensuring consistency and efficiency.

1.3.1 Automation Framework Key Aspects

- **Reusable Components:** Includes libraries, test data, and reusable modules that can be used across multiple test cases.
- **Standardized Processes:** Establishes standard procedures for writing, executing, and reporting tests, which helps maintain consistency.
- **Integration:** Often integrates with other tools such as version control systems, continuous integration servers, and test management tools.
- **Scalability:** Designed to handle a growing number of tests and adapt to changes in the application being tested.
- **Maintenance:** Facilitates easier maintenance of test scripts, reducing the effort required to update tests when the application changes.

1.3.2 Most Common Types of Automation Frameworks

- **Linear (Record and Playback) Framework:** Simple to use but less flexible.
- **Modular Framework:** Divides the application into modules, making tests more manageable.
- **Data-Driven Framework:** Uses external data sources to drive test cases.
- **Keyword-Driven Framework:** Uses keywords to represent actions, making tests more readable.
- **Hybrid Framework:** Combines elements of multiple frameworks to leverage their strengths.

1.4 Test Scenario

A high-level description of a functionality or feature of a software application that needs to be tested. It represents a specific use case or situation that the application might encounter in the real world.

1.4.1 Test Scenario Key Aspects

- **Broad Coverage:** Test scenarios cover a wide range of conditions and interactions within the application, ensuring comprehensive testing.
- **User Perspective:** They are often written from the end user’s perspective, focusing on how the user will interact with the application.

- **Basis for Test Cases:** Test scenarios serve as the foundation for creating detailed test cases. A single test scenario can lead to multiple test cases.
- **Simplified Documentation:** They provide a simplified and understandable way to document testing requirements, making it easier for stakeholders to review.

2 Types of Performance Tests

2.1 Load Testing

A type of performance testing that evaluates how a system behaves under expected user loads. The primary goal is to ensure that the system can handle anticipated traffic without performance degradation and determine peak load that system can handle before performance starts degrading.

2.1.1 Load Testing Achieves Following Goals

- **Simulates Real-World Usage:** Load testing involves simulating multiple users accessing the system concurrently to mimic real-world usage conditions.
- **Measures Performance Metrics:** It assesses various performance metrics such as response time, throughput, and resource utilization (CPU, memory, etc.) under different load conditions.
- **Measure Peak Load:** Peak load is the point where there is max load on the system before performance degradation or system failure (errors) start happening.
- **Identifies Bottlenecks:** By applying load, it helps identify performance bottlenecks and areas where the system may fail or slow down.
- **Ensures Stability:** It ensures that the system remains stable and performs efficiently under both normal and peak load conditions.
- Load testing is crucial for applications expected to handle many users, such as e-commerce websites during sales events or online services during peak usage times.

2.1.2 Real World Examples of Load Testing

- **Web Applications:** Determine what is the expected load and production server configuration. For example: login screen needs to handle 100 users, at ‘x’ network speed and prod server config of 16 Virtual CPU’s and 64gb virtual ram. Ensure Performance or QA server has required hardware configuration and design load test scenario with incremental user load and validate the through put.
- **Backend Data:** Ensure performance or QA server database server is at production db. server configuration and test data is available. Execute the full and/or incremental loads and determine throughput to confirm the performance.

2.2 Endurance Testing

A type of performance testing that evaluates how a system performs under a sustained load over an extended period. The primary goal is to ensure that the system can handle prolonged usage without experiencing performance degradation, memory leaks, or system failures.

2.2.1 Endurance Testing Key Aspects and Goals

- **Prolonged Load:** The system is subjected to a continuous load for an extended duration, which could range from several hours to days.
- **Memory Leaks:** It helps identify memory leaks, where the system fails to release memory that is no longer needed, potentially leading to system slowdowns or crashes.
- **Performance Stability:** Monitors the system's response time, throughput, and resource utilization to ensure consistent performance over time.
- **Real-World Scenarios:** Simulates real-world usage patterns to ensure the system can handle actual user behavior over lengthy periods.
- Endurance testing is crucial for applications that need to run continuously without interruption, such as banking systems, online services, and critical infrastructure.

2.2.2 Real World Examples of Endurance Testing

- **Web Applications:** Determine the expected load, the peak load system can handle and production server configuration. For example: login screen needs to handle 100 users, at 'x' network speed, peak load of 150 users and prod server config of 16 Virtual CPU's and 64gb virtual ram. Ensure Performance or QA server has required hardware configuration and design an endurance test scenario at required load of 100users and 4 hrs. of run time and at peak load 150 users and 4hrs of run time.
- **Backend Data Loading:** Ensure performance or QA server database server is at production db. server configuration and test data is available. Start data full and/or incremental load for more than required time i.e., for example if typical full load takes 2hrs – run an endurance test for 8hrs and review system performance.

2.3 Stress Testing

Executing most common functional scenarios with incrementally increasing the system load past the peak load till servers completely shut down and seeing how the system recovers from such catastrophic events gracefully.

2.3.1 Stress Testing Key Aspects and Goals

- **Extreme Load Conditions:** The system is subjected to high levels of load, such as - many concurrent users or high data volumes, to see how it handles stress.
- **Identifies Weaknesses:** It helps uncover vulnerabilities, such as memory leaks, bottlenecks, and other performance issues that may not be evident under normal conditions.

- **Error Handling and Recovery:** Evaluates how well the system manages errors and recovers from crashes or failures.
- **Robustness and Stability:** Ensures that the system remains robust and stable even when pushed beyond its limits.
- Stress testing is crucial for applications that need to maintain high performance and reliability under unexpected or peak load conditions, such as financial systems during market surges or online services during major events.

2.3.2 Real World Examples of Stress Testing

- **Web Applications:** Determine the expected load, the peak load system can handle and production server configuration. For example: login screen needs to handle 100 users, at 'x' network speed, peak load of 150 users and prod server config of 16 Virtual CPU's and 64gb Virtual ram. Ensure Performance or QA server has required hardware configuration and design a stress that incrementally increases load starting at around 80 users by 10 users every 10minutes till such load that system completely fails.
- **Backend Data Loading:** Ensure performance or QA server database server is at production db. server configuration and test data is available. Start data full and/or incremental load and keep incrementally increasing the load to exceed peak sustainable load and see how the system recovers from this catastrophic event.

2.4 Spike Testing

A type of performance testing that evaluates how a system handles sudden and extreme increases or decreases in load. The primary goal is to determine the system's robustness and recovery capabilities under these abrupt changes.

2.4.1 Spike Testing Key Aspects and Goals

- **Sudden Load Changes:** The system is subjected to rapid increases and decreases in load to simulate real-world scenarios like flash sales or viral content.
- **Performance Metrics:** It measures how quickly the system can scale, how fast it can recover, and whether these spikes cause any performance bottlenecks or issues.
- **Recovery Time:** Evaluates the system's ability to return to normal performance levels after the spike.
- **Identifies Weaknesses:** Helps uncover vulnerabilities that may not be evident under steady load conditions.
- Spike testing is crucial for applications that may experience sudden surges in traffic, such as e-commerce platforms during major sales events or websites during high-profile announcements.

2.4.2 Real World Examples of Spike Testing

- **Web Applications:** Determine the expected load, the peak load system can handle and production server configuration. For example: login screen needs to handle 100 users, at 'x' network speed, peak load of 150 users and prod server config of 16 Virtual CPU's and 64gb

Virtual ram. Ensure Performance or QA server has required hardware configuration and design spike test scenarios with initial 10 user load and increase that number 150 users in 10 mins and so on.

- **Backend Data loading:** Ensure performance or QA server database server is at production db. server configuration and test data is available. Start data full and/or incremental load and keep fluctuating the load at high frequencies at short intervals and repeat this for lengthy periods