## Performance Testing

Lesson 1: Introduction to Performance Testing

# Lesson Objectives

- To understand the following topics:
  - Performance Testing – An Overview
  - Which Applications should we Performance Test ?
  - Performance Problems
  - Objectives of Performance Testing
  - What is not Performance Testing
  - Types of Performance Testing
  - When To Do Performance Testing
  - Why Performance Testing
  - Performance Testing Metrics
  - Examples of Performance Test Cases
  - Performance Testing Fallacies
  - Summary

# Performance Testing

# Functional & Performance Testing

| Functional Testing | Performance Testing |
|---|---|
| Client side | Server side |
| Records each user action | Records only interaction |
| Mostly only one user | Multiple users |
| Examples: UFT, Selenium | Examples: LoadRunner, Jmeter |

## Performance Testing – An Overview

- Performance testing is a generic term that can refer to many different types of performance related testing, each of which addresses a specific problem area and provides its own benefits, risks, and challenges
- Testing conducted to evaluate the compliance of a system or component with specified performance requirements
  - "Goal of Performance testing is not to find bugs, but to eliminate the bottlenecks"
- A bottleneck is a stage in a process that causes the entire process to slow down or stop

**What is not Performance Testing ?**

- GUI test
- High-volume Functional Test

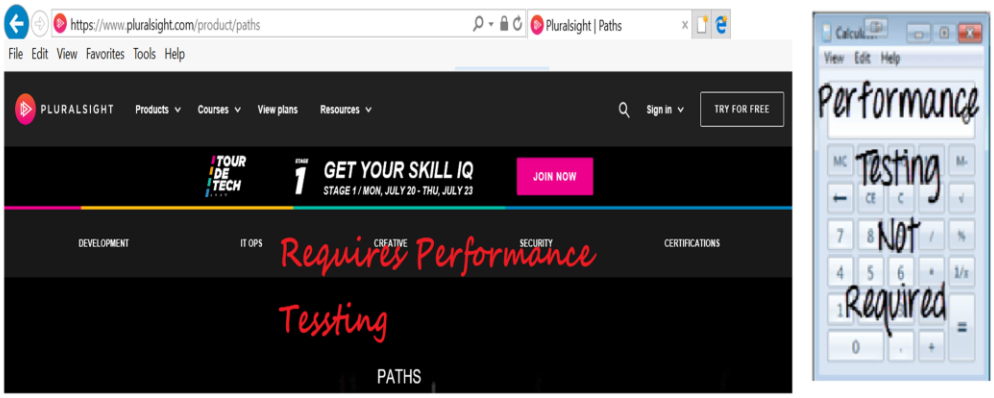
## Performance Testing – An Overview (Cont.)

- Performance Testing checks the speed, response time, reliability, resource usage, scalability and stability characteristics of an application, thereby providing an input to making sound business decisions
- Focuses on determining if the user of the system will be satisfied with the performance characteristics of the application
- ***Performance Testing*** is popularly called "***Perf Testing***" and is a subset of ***performance engineering***.

**Performance Testing** is a discipline concerned with ***testing and reporting*** the current performance of a software application under various parameters.
**Performance engineering** is a discipline concerned with ***testing and tuning*** *the* software application with the intent of realizing the required performance.

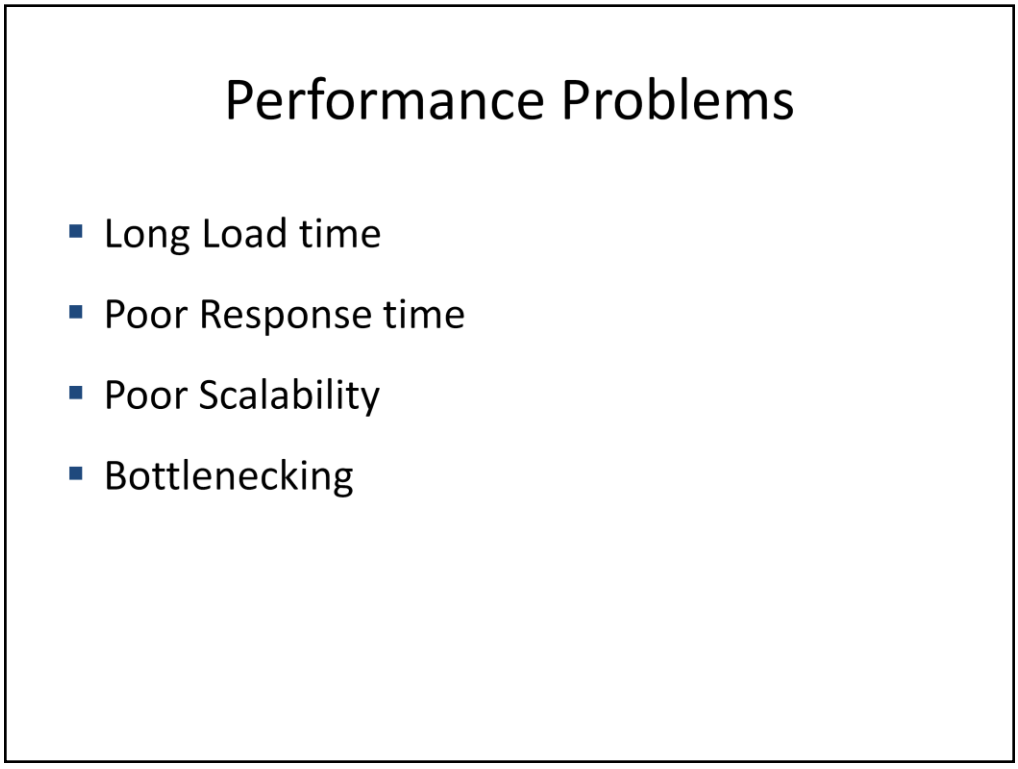
## Which Applications should we Performance Test ?

- Performance Testing is always done for client-server based systems only.




Any application which is not a client-server based architecture, must not require Performance Testing.

- **Example:** Microsoft Calculator is neither client-server based nor it runs multiple users; hence it is not a candidate for Performance Testing.
- **Example:** Pluralsight, Coursera, Degreed, DoSelect is client-server based system and runs multiple users at a time; hence it requires Performance testing

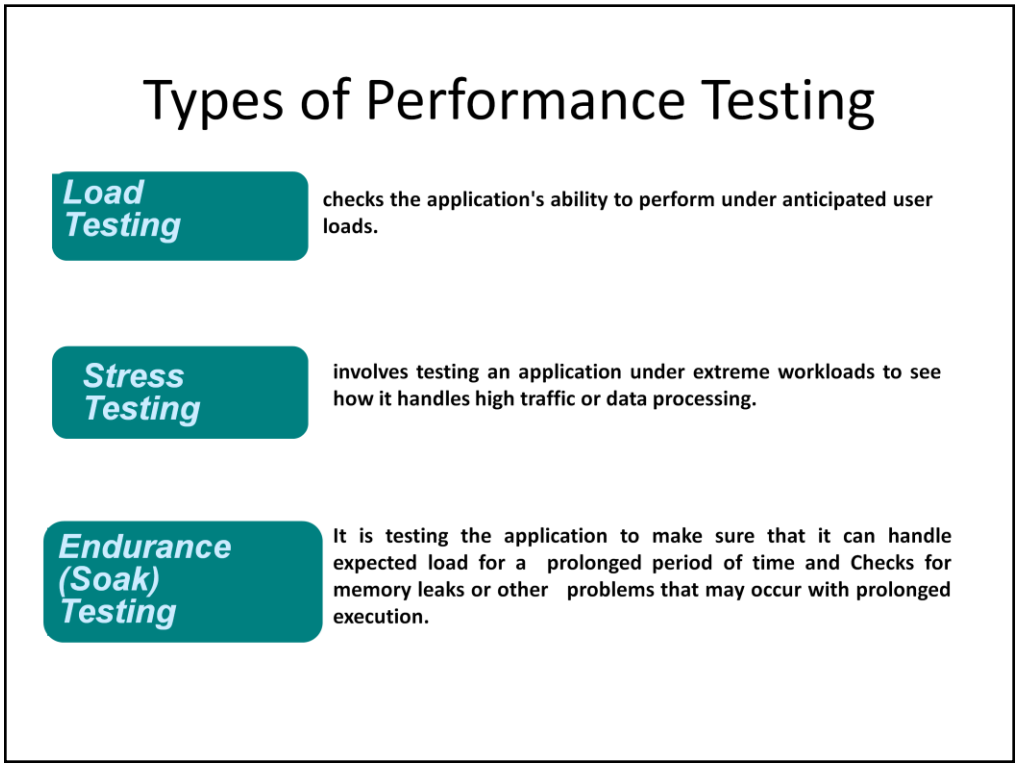



**Long Load time -** Load time is normally the initial time it takes an application to start. This should generally be kept to a minimum. While some applications are impossible to make load in under a minute, Load time should be kept under a few seconds if possible.
**Poor response time -** Response time is the time it takes from when a user inputs data into the application until the application outputs a response to that input. Generally, this should be very quick. Again if a user has to wait too long, they lose interest.
**Poor scalability -** A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users. Load Testing should be done to be certain the application can handle the anticipated number of users.
**Bottlenecking -** Bottlenecks are obstructions in a system which degrade overall system performance. Bottlenecking is when either coding errors or hardware issues cause a decrease of throughput under certain loads. Bottlenecking is often caused by one faulty section of code. The key to fixing a bottlenecking issue is to find the section of code that is causing the slowdown and try to fix it there. Bottlenecking is generally fixed by either fixing poor running processes or adding additional Hardware.
Some **common performance bottlenecks** are: CPU utilization, Memory utilization, Network utilization, Operating System limitations, Disk usage

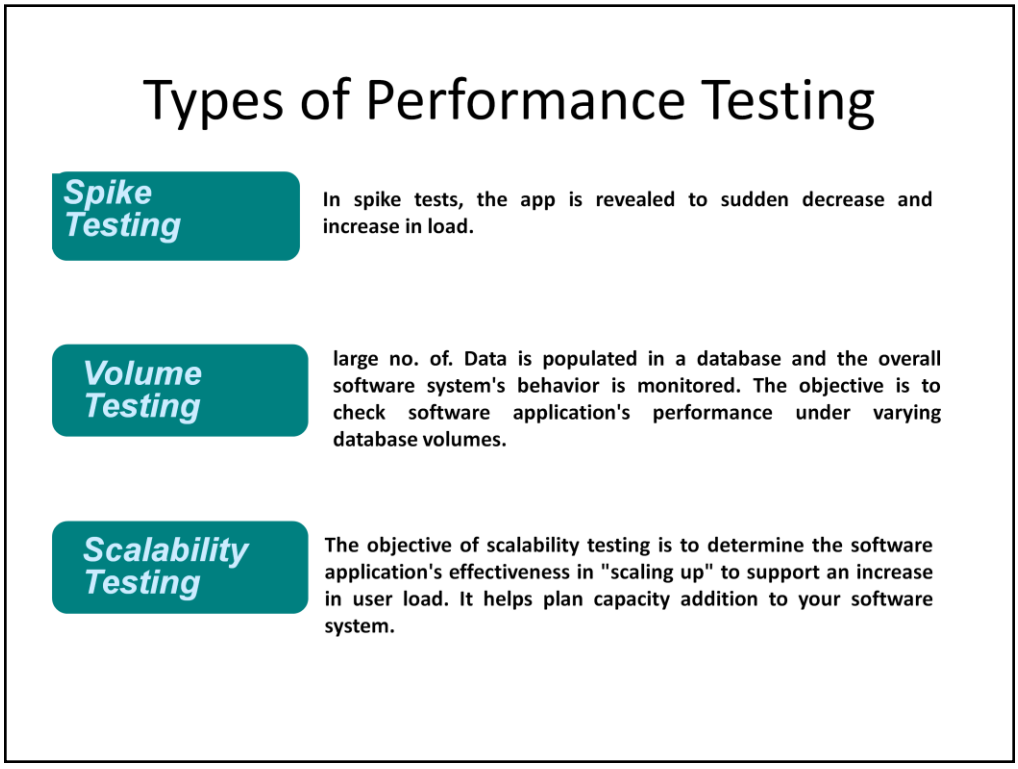
# Objectives of Performance Testing

- to identify performance bottlenecks before the software application goes live.
- to identify the breaking point of an application.

| | |
|---|---|
| ▪ Response Time | - Check whether application responds quickly? |
| ▪ Stability | - How Stable is the system under heavy work load? |
| ▪ Configuration Sizing | - Which configuration provides the best performance level |
| ▪ Capacity Planning | - What H/W does the application supports? |
| ▪ Acceptance | - Is the system stable enough to go into production? |
| ▪ Bottleneck Identification | - What is the cause of degradation in performance? |
| ▪ Regression | - Does the new version of Software adversely affect response time? |
| ▪ High data Requirements | - 500 x 60 x 5 = 1,50,000 rows |
| ▪ Infrastructure Metrics | - CPU, Memory. |
| ▪ Scalability | - Checks maximum Concurrent 'virtual' users /user load the software application can handle. |




**Stress testing**
It is also called as Negative testing or Fatigue testing which captures the stability of the application by testing it beyond its bandwidth capacity.
**Example of Endurance Testing :**
Where a system is designed to work for 3 hrs of time but same system endure for 6 hrs of time to check the staying power of system.




**Spike Testing :**
Spike testing is subset of Stress Testing.

**Scalability Testing :**
The main aim if this testing is to understand at what peak the system prevent more scaling.

# When To Do Performance Testing

- During Design and Development:
  - What is the best server to support target load
  - Define system performance requirements
- Before Release
  - Hardware evaluation
  - Configuration changes
  - Is the system reliable enough to go in to production
  - After functional testing done
- Post Deployment -
  - What is the cause of performance degradation

## Why Performance Testing

- The failure of a mission-critical application can be costly
- Assure performance and functionality under real-world conditions
- Locate potential problems before your customers do

- Mission-critical applications like space launch programs or life-saving medical equipment should be performance tested to ensure that they run for a long period without deviations.
- Without Performance Testing, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability.
- Applications sent to market with poor performance metrics due to nonexistent or poor performance testing are likely to gain a bad reputation and fail to meet expected sales goals.

## Some Facts that specify the importance of Performance Testing

- According to Dunn & Bradstreet, 59% of Fortune 500 companies experience an estimated 1.6 hours of downtime every week. It has minimum of 10,000 employees - the labor part of downtime costs for such an organization would be $896,000 weekly, translating into more than $46 million per year.
- Only a 5-minute downtime of Google.com (19-Aug-13) is estimated to cost the search giant as much as $545,000.
- Yahoo News: Fri Jun 6, 2008: Amazon Inc US website was down for about 2 hrs. due to which Amazon shares fell 4.59%
- Internet Business News: July 27, 2006: Australia's JetStar airline website crashed as too many users went online to take advantage of holiday promotions.
- **Hence, performance testing is important.**

Performance of your applications = Performance of your business

# Performance Testing Metrics

*The basic parameters monitored during performance testing include:*

- **Processor Usage -** an amount of time processor spends executing non-idle threads.
- **Memory use -** amount of physical memory available to processes on a computer.
- **Disk time -** amount of time disk is busy executing a read or write request.
- **Bandwidth -** shows the bits per second used by a network interface.
- **Private bytes -** number of bytes a process has allocated that can't be shared amongst other processes. These are used to measure memory leaks and usage.

# Performance Testing Metrics

- **Committed memory -** amount of virtual memory used.
- **Memory pages/second -** number of pages written to or read from the disk in order to resolve hard page faults. Hard page faults are when code not from the current working set is called up from elsewhere and retrieved from a disk.
- **Page faults/second -** the overall rate in which fault pages are processed by the processor. This again occurs when a process requires code from outside its working set.
- **CPU interrupts per second -** is the avg. number of hardware interrupts a processor is receiving and processing each second.
- **Disk queue length -** is the avg. no. of read and write requests queued for the selected disk during a sample interval.

# Performance Testing Metrics

- **Network output queue length -** length of the output packet queue in packets. Anything more than two means a delay and bottlenecking needs to be stopped.
- **Network bytes total per second -** rate which bytes are sent and received on the interface including framing characters.
- **Response time -** time from when a user enters a request until the first character of the response is received.
- **Throughput -** rate a computer or network receives requests per second.
- **Amount of connection pooling -** the number of user requests that are met by pooled connections. The more requests met by connections in the pool, the better the performance will be.

# Performance Testing Metrics

- **Maximum active sessions -** the maximum number of sessions that can be active at once.
- **Hit ratios -** This has to do with the number of SQL statements that are handled by cached data instead of expensive I/O operations. This is a good place to start for solving bottlenecking issues.
- **Hits per second -** the no. of hits on a web server during each second of a load test.
- **Rollback segment -** the amount of data that can rollback at any point in time.
- **Database locks -** locking of tables and databases needs to be monitored and carefully tuned.

# Performance Testing Metrics

- **Top waits -** are monitored to determine what wait times can be cut down when dealing with the how fast data is retrieved from memory
- **Thread counts -** An applications health can be measured by the no. of threads that are running and currently active.
- **Garbage collection -** It has to do with returning unused memory back to the system. Garbage collection needs to be monitored for efficiency.

## Examples of Performance Test Cases

- Verify response time is not more than 4 secs when 1000 users access the website simultaneously.
- Verify response time of the Application Under Load is within an acceptable range when the network connectivity is slow
- Check the maximum number of users that the application can handle before it crashes.
- Check database execution time when 500 records are read/written simultaneously.
- Check CPU and memory usage of the application and the database server under peak load conditions
- Verify response time of the application under low, normal, moderate and heavy load conditions.

During the actual performance test execution, vague terms like acceptable range, heavy load, etc. are replaced by concrete numbers. Performance engineers set these numbers as per business requirements, and the technical landscape of the application.

# Summary

- In this lesson, you have learnt:
  - What is Performance Testing?
  - Objectives of Performance Testing
  - Performance Testing Metrics
  - Examples Performance Test Cases