Performance Testing

Load, Stress, Spike, Soak, Scalability, Capacity, Volume

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg

Have a Question?

Software University

sli.do

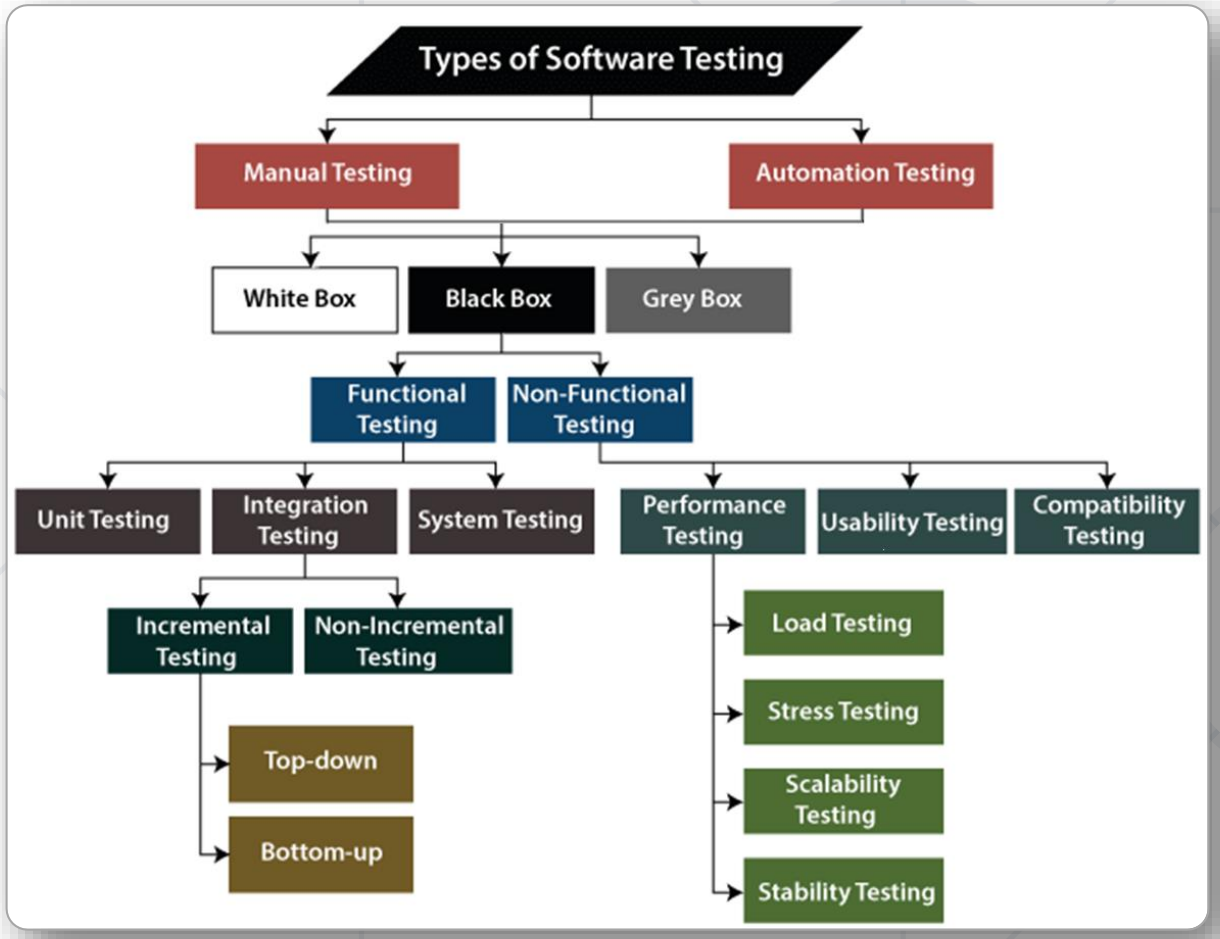
#QA-Auto-BackEnd

Table of Contents

Functional vs. Non-Functional Testing

Workload Stability

Functional vs. Non-Functional Testing



- **Functional testing**
  - True / False
- **Non-functional testing**
  - Evaluates how well the system or application performs under certain conditions and constraints

Compatibility Testing

- Tests software **compatibility** with **different environments**
- Ensures **consistent behavior** across **various systems**
  - **Browser Compatibility**: Application's performance on different web browsers
  - **Operating System Compatibility**: Ensures smooth operation across different OS versions
  - **Device Compatibility**: Verifies functionality on various devices, including mobiles and tablets
  - **Network Compatibility**: Assesses application performance across network configurations

Usability Testing

- Assesses how **user-friendly** and **intuitive** the application is
- Focuses on **user satisfaction** and **ease of use**
  - **Learnability**: How quickly can a new user learn to navigate the app?
  - **Efficiency**: How swiftly can users accomplish tasks?
  - **Memorability**: After some time away, can users re-engage with the app easily?
  - **Error Rate**: How many errors do users make, and how severe are they?
  - **Satisfaction**: How pleasant is the experience of using the application?

Performance Testing

Stabilizing Workload

Performance Testing: Overview

- Type of **non-functional testing** aimed at **evaluating** various **performance aspects** of a software application under specific **workload conditions**
- Its primary goals include **identify** and **eliminate** the **performance bottlenecks**
- Key Focus:
  - **Speed** - Assessing the application's responsiveness and speed
  - **Scalability** - Evaluating the maximum load of users the application can support
  - **Stability** - Testing the application's robustness with varying loads

Why do Performance Testing?

- Provides valuable insights into the application's **speed**, **stability**, and **scalability**
- Highlights **areas for improvement before** the product's **release**, enhancing overall quality
- Prevents **common problems** like **slowdowns** under heavy user load
- Helps **avoid negative** market **reception** and **poor** user **reviews**
- Aims to **safeguard against loss of sales** due to performance shortcomings

Downtime Costs

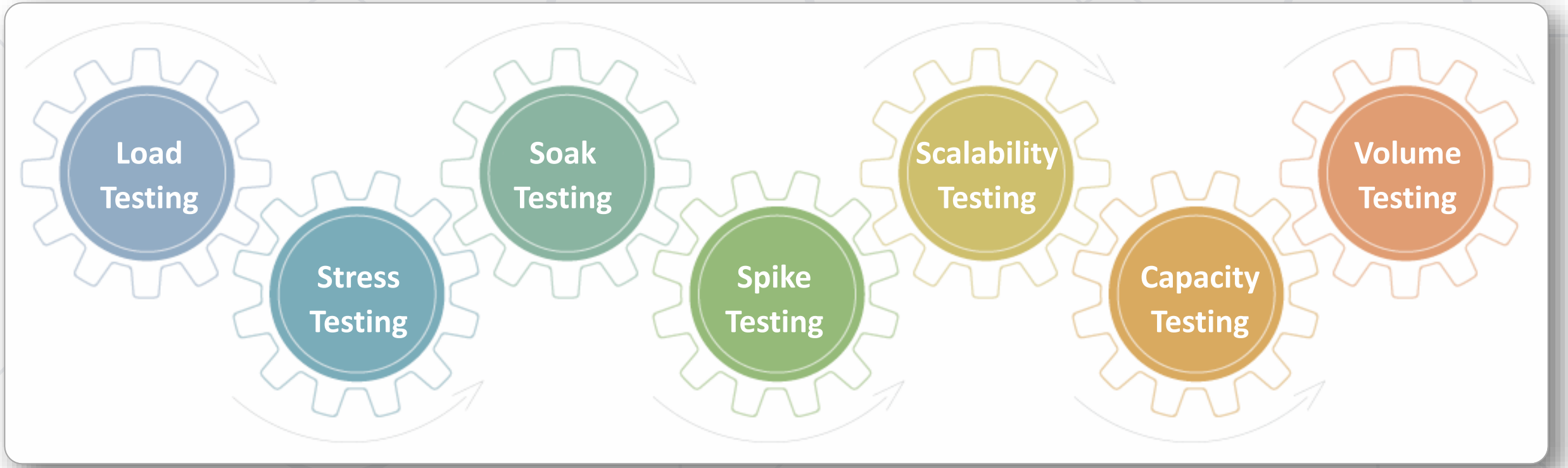
- The **manufacturing industry** sees costs of **$260,000 per hour**
- The **enterprise sector** can experience downtime costs exceeding **$1 million per hour**
- **44% of enterprises** report **hourly downtime costs** between **$1 million to over $5 million**
- A **single hour** of downtime can average **over $300,000** in lost business and productivity
- The **average cost** of downtime across businesses is **$1,467 per minute**, or **$88,000 per hour**, highlighting significant impacts on all business sizes

Performance Testing Types

Assessing Speed, Scalability, and Stability

Types

- **Seven Types** of Performance Testing



Load Testing

- Evaluates the performance of an application under the **expected real-world load**
- Determines the application's behavior when **multiple users access** or **use it simultaneously**
- The objective is to detect performance problems **before the software's launch**
- Aims to ensure the application's **stability and smooth functioning** under **normal circumstances**

Load Testing Example

- Newly developed application
- **Anticipated load** of around **1000 concurrent users**
- A load test script is created and configured with **1000 virtual users** and run for say **1-hour duration**
- After the load test completion, the test result are analyzed to determine how the application will behave at the expected peak load

Stress Testing

Software University

- Places a system under **higher-than-expected traffic loads**
- Determines **how** the **system functions above its capacity limits**
- Puts **strain on hardware resources** like CPUs, memory, and disk drives to find the **breaking point**
- Can reveal issues such as **slow data exchange**, **memory shortages**, **data corruption**, and **security vulnerabilities**
- Can be **conducted before or after a system is live**
- Used before **major events**, like Black Friday, to **simulate traffic**
- Subcategories: **soak testing** and **spike testing**

Stress Testing Example

- If the application is bound to serve **1000 concurrent users**
- Stress testing puts a **load of 1200 users**
- Then, the application **behavior is analyzed** to seek **answers** to the **following questions**:
  - What is the **breaking point** of the application?
  - What is the **error rate**?
  - Does it **crash**?
  - How long does it take to **recover from a crash**?
  - Are there any **memory leaks**?

Soak (Endurance) Testing

- Also called **Endurance Testing**
- Simulates a **steady increase** of end **users** or **draining tasks** over **extended period of time**
- Aims to find out how the software can **handle continuous usage** and to **identify any performance problems** that may occur after **extended use**
- Also analyzes **throughput** and **response times** after **sustained use** to show if these **metrics** are **consistent** with their status at the **beginning** of a test

Soak (Endurance) Testing Example

- For an application like **Income tax filing**
- The application is **used continuously** for a **long duration** by **different users**
- **Memory management** is critical
- For an application like these, **tests** can **run for 24 hours to 2 days duration** and **monitor** the **memory utilization** during the **whole test execution**

Spike Testing

- Checks if the system will survive the **sudden increments** and **decrements** in workload over a **short period of time**, **repeatedly**
- This sudden increase and decrease in the workload is spiking
- Assesses the **performance** of a system **under a sudden and significant increase** of simulated end **users**
- Typically **performed before** a **large event** in which a system will likely undergo **higher-than-normal traffic volumes**
- It also **involves checking** if the **application** is **able to recover** after the sudden burst of users

Spike Testing Example

- For an **e-commerce** application running an **advertisement campaign**
- Number of **users can increase suddenly** in a very **short duration**
- Or **Ed Sheeren** concert ticket sales ☺
- Spike testing is done to analyze these types of scenarios
- **Needs experts** in performance testing and cannot be conducted by common testers

Scalability Testing

- The objective is to determine the software application's **effectiveness in "scaling up"** to support an increase in user load
- Helps plan **capacity addition** to the software system
- Demonstrates the **effects of projected increases** in the use of an application
- Measures performance based on the software's **ability to scale performance measure attributes up or down**
- Seeks to understand the **effect of changes in numbers of users** and other performance attributes

Scalability Testing Example

- A **video streaming service** preparing for a **major live event** expected to draw **300,000 concurrent viewers**, significantly above its **regular 50,000 viewer base**
- **Before the Event:** Scalability testing reveals the service can handle up to 200,000 streams. To prepare for the event, it scales up resources accordingly, **adding servers and increasing bandwidth**
- **After the Event:** Once the event concludes and viewer numbers return to normal, the **service scales down resources** to match the regular demand, **reducing unnecessary operational costs**

Capacity Testing

- Capacity testing can be seen as a **subset of scalability testing**
- Determines the **maximum number of users or transactions** a system can handle while **still meeting performance goals**
- It's about **finding the boundaries** of the system's capacity within **specific criteria**, such as not exceeding a predefined maximum page load time
- Identifies the **upper limits** of what a system **can handle without breaking predefined performance criteria**

Capacity Testing Example

- If the system could smoothly **handle 20 users** with a page **response time** of **3.5 seconds**
- The next step is to **determine the system's capacity**: at what **point does it fail** to maintain the 3.5-second response time?
- Is the limit 21, 30, 40, or 50 users?
- The overarching goal is to **pinpoint system's "safety zone"**
- How far can you **push the system's limits** without **negatively impacting user experience**?

Volume Testing

- Often **referred** to as **Flood Testing**
- Aimed at **evaluating** how well a software **application handles ranging volumes of data**
- Tests are done by creating a **sample file size**:
  - Either a **small amount** of data
  - Or a **larger volume**
- Then **testing** the application's **functionality** and **performance** with that file size
- Helps identify **potential issues** related to **data management** and **processing**

Volume Testing Example

- An **online bookstore** with an **inventory database**
- The application performs well with an **initial 10,000 book records**
- **Data sets for testing** are prepared with **50,000 records** (moderate load) and **100,000 records** (high load)
- The application's **response to each data set is measured**, highlighting **search speed** and **system stability**
- **Performance slightly decreases with 50,000** records but significantly **worsens at 100,000 records**, revealing scalability limits
- Based on the test, the team considers **optimizations** like **database indexing** or **server upgrades** to handle larger volumes effectively

Performance Testing Metrics

Monitoring and Analyzing Performance

Performance Testing Metrics

Software University

- A **subset** of software testing metrics
- Utilized to **evaluate the performance** of software systems under load
- Assist in **prioritizing testing efforts** based on criticality and usage scenarios
- Aid in **assessing the scalability** of the system or application under test
- Establish **baselines** for system performance under normal and expected loads
- Provide **insights on system resource use**, such as CPU, memory, and network usage
- Analyze the **performance of third-party systems** or integrated APIs

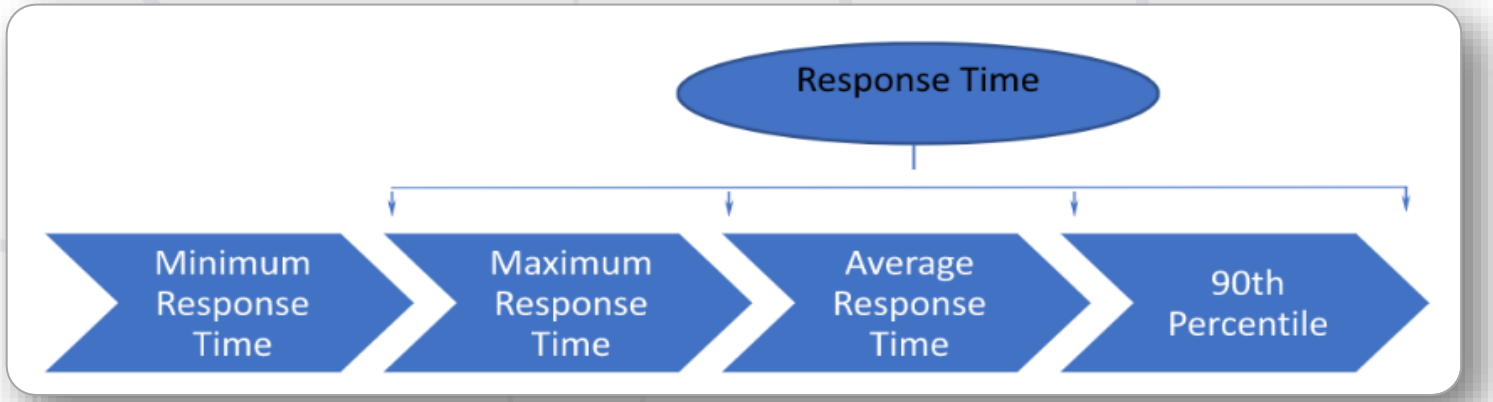
Key Performance Test Metrics

Software University

- **Response Time**
  - Measures the total time it takes a system to respond to a user request
  - The interval between a user's request and the arrival of the response
  - It is one of the most critical metrics as it ensures systems are responsive and are meeting user expectations
  - There are **4 subcategories** of response time

Key Performance Test Metrics

- **Minimum Response Time**
  - Measures the shortest amount of time the system takes to respond to a user request (**best-case scenario**)
- **Maximum Response Time**
  - The longest amount of time the system takes to respond to a user request (**worst-case scenario**)



Key Performance Test Metrics

Software University

- **Average Response Time**
  - Measures the sum of all the response times divided by the total number of requests (**typical response time**)
    Average response time = Total response time / Number of requests
- **90th Percentile**
  - Represents the **time required for 90% of requests to be completed successfully**
    90th percentile response time = Average response time corresponding to the fastest 90% of requests

Key Performance Test Metrics

- **Throughput**
  - Measures the number of requests that can be processed by a system in a given time
  - Generally measured in units of bytes per second or transactions per second
    Throughput = Total no. of requests/ Total time taken
- **Error Rate** (Also known as an **Error Percentage**)
  - Measures the percentage of requests that failed or didn't receive a response
    Error Rate = (Number of failed requests / Total number of requests) x 100

Key Performance Test Metrics

- **CPU Utilization**
  - Measures the percentage of CPU capacity utilized while processing the requests
    CPU utilization (%) = (1 - (Idle time / Total time)) * 100
- **Memory Utilization**
  - Measures the amount of memory that is being used by a system or application, compared to the total amount of memory available
    Memory utilization (%) = (Used memory / Total memory) * 100

Key Performance Test Metrics

- **Average Latency Time** (Also known as plain "**latency**")
  - Measures the amount of time it takes for a system or application to respond to a user's request. Generally measured in milliseconds
    Latency = Processing time + Network transit time
- **Network Latency** (Also known as "**network delay**" or "**lag**")
  - Refers to the delay that occurs during data transmission over a network
  - Can be caused by various factors such as distance between the sender and receiver, limited bandwidth, type of network technology used
    Network Latency = Time taken for response - Time spent

Key Performance Test Metrics

Software University

- **Wait Time**
  - Indicates how much time elapses from the moment a request is sent to the server until the first byte is received
  - Can be viewed from both perspectives i.e., from users and applications
  - User: the **time spent waiting for the system to respond to their request**, e.g. time taken to load a page, perform a search, or complete a transaction
    Wait time = Response time - Processing time (user's perspective)
  - Application: the time taken by the **system to process a user request after it has been received**, e.g. network latency, resource contention, or database performance issues
    Wait time = Processing time - Queue time (application's perspective)

Key Performance Test Metrics

- **Concurrent User Capacity**
  - The maximum number of users that can use a system or application simultaneously without degrading performance or causing errors
- **Transaction Pass/Fail**
  - **Transaction pass** occurs when a transaction has been completed as expected without any error or delay
    Transaction pass = (No. of successful transactions / Total Transactions) x 100%
  - **Transaction failure** occurs when the transaction is initiated and attempted to complete, but fails due to some error. For example, a user enters incorrect payment details, which causes the payment to fail
    Transaction fail = (No. of failed transactions / Total Transactions) x 100%

Performance Testing Tools

Maximizing Efficiency

What are Performance Testing Tools?

Software University

- Applications designed to facilitate the **planning**, **execution**, **management**, **monitoring**, **reporting**, and **analysis** of performance tests for software systems, applications, and websites
- Most performance testing tools have **3 major capabilities**:
  - **Simulate load conditions** of the System Under Test (SUT)
  - **Monitor** system **behavior**
  - **Analyze performance metrics** to make recommendations
- Performance testing tools generally **differ in their scope**, but they all come with features to support testers across the performance testing life cycle

JMeter: Free Performance Testing Tool

Software University

- Apache **JMeter** is **free**, open-source, powerful performance tool
- 100% pure **Java application**
- Around since **1998**
- Often referred to as a "**grandfather**" in the world of performance testing, due to its age
- Designed to **load test** functional behavior and **measure performance**
- Wide **range** of **plugins** and **integrations**
- Full featured **Test IDE** that allows fast **Test Plan recording**
- Download here: https://jmeter.apache.org

JMeter Features

- **Can load test different kinds of applications**: Performance testing of all kinds of apps (web apps, web services, databases, LDAP, shell scripts, etc.)
- **Platform independent**: Since, it is 100% Java-based, so it is platform-independent and can run on multiple platforms
- **Record and Playback** feature, along with **Drag and Drop** features, makes it easier and faster to create scripts
- **Customizable**: Its source code can be customized as per their specific needs
- **Distributed load testing**: Master-slave set up for carrying out load tests on multiple machines
- Good **community support** and **freely available plugins** that help in different aspects of script creation and analysis

BlazeMeter

- BlazeMeter - powerful and flexible, cloud-based load testing platform
- Built on top of Apache Jmeter
- Extends the functionality of Jmeter to provide advanced load testing capabilities
  - Distributed testing: Enables testing from multiple geographic locations
  - Real-time reporting
  - Advanced analytics

BlazeMeter Features

- **Mock Services:** Easily create Lightweight Virtual Services for Any Test
- **Synthetic Test Data:** Allows to source load test data from spreadsheets, generate synthetic test data, extract data from TDM Database Models, or utilize a mix of these options
- **API Testing & Monitoring:** Easily validate test data and complex API workflows
- **Selenium:** Using existing Selenium scripts with BlazeMeter

K6

- Modern, open-source load testing tool – https://k6.io
  - **Testing framework** based on **JavaScript** (very powerful)
  - Local & cloud **script executor** based on **Go** (high performance)
  - **Script recorder** (Chrome plugin) → generates JS code
- Tests are plain **JavaScript code**
  - No XML configuration, no need for complex UI
  - Very **powerful**: JavaScript can test anything
  - Easy to use with **continuous integration** scripts

Loader.io

- Loader.io - cloud-based service that provide s load testing for web apps and APIs
- Supports up to **50,000 concurrent connections** for free
- Users can quickly register apps for testing through a **simple web interface** or API
- **Real-time test monitoring**, allowing to watch performance as it happens
- Compatible with PaaS providers, continuous integration tools, and browsers for seamless workflow integration
- You can start testing immediately without any setup on your servers

Comparison

Software University

Feature	JMeter	BlazeMeter	k6	Loader.io
Type	Open-source load testing tool	Commercial load testing service	Open-source load testing tool	Cloud-based load testing service
Custom Protocol Support	Extensive via plugins	Via JMeter and other tools	Limited, mainly HTTP	Primarily HTTP/HTTPS
Execution Mode	Local and distributed	Cloud-based	Local and cloud	Cloud-based
Scripting Language	XML	JMeter, Selenium, others	JavaScript	Through UI or API
Integration	Wide range of plugins, CI/CD tools	CI/CD integration, API monitoring	CI/CD tools, Grafana	PaaS providers, CI tools
Cloud Support	Through integrations (e.g., BlazeMeter)	Native	Yes, with k6 Cloud	Native
Pricing	Free	Free tier, Paid plans	Free for open-source, Paid plans for Cloud	Free tier, Paid plans

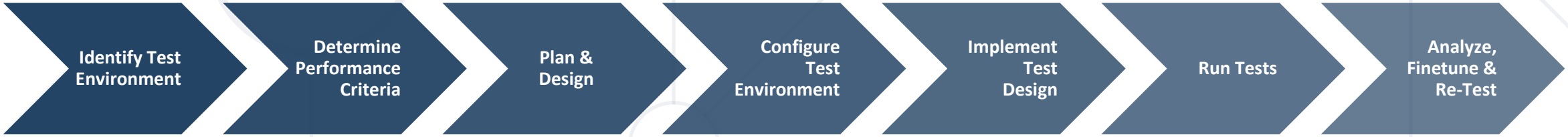
How to do Performance Testing?

Steps

How to do Performance Testing?

Software University

- The **methodology** adopted for performance testing **can vary** widely
- **Objective** for remains the same:
  - Demonstrate that software system **meets certain** pre-defined performance **criteria**
  - **Compare** the performance of **two software systems**
  - **Identify** parts of your software system which **degrade** its performance
- Below is a **generic process** on how to perform performance testing:



Step 1

- **Identify Your Testing Environment**
  - Know your physical test environment, production environment and what testing tools are available
  - Understand details of the hardware, software and network configurations used during testing before you begin the testing process
  - It will help testers create more efficient tests
  - It will also help identify possible challenges that testers may encounter during the performance testing procedures

Step 2

- **Identify the Performance Acceptance Criteria**
  - This includes goals and constraints for throughput, response times and resource allocation
  - It is also necessary to identify project success criteria outside of these goals and constraints
  - Testers should be empowered to set performance criteria and goals because often the project specifications will not include a wide enough variety of performance benchmarks.
  - Sometimes there may be none at all
  - When possible finding a similar application to compare to is a good way to set performance goals

Steps 3, 4, 5

- **Plan & Design Performance Tests**
  - Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases
  - It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered
- **Configuring the Test Environment**
  - Prepare the testing environment before execution
  - Also, arrange tools and other resources
- **Implement Test Design**
  - Create the performance tests according to your test design

Steps 6, 7

- **Run the Tests**
  - Execute and monitor the tests
- **Analyze, Tune and Retest**
  - Consolidate, analyze and share test results
  - Then fine tune and test again to see if there is an improvement or decrease in performance
  - Since improvements generally grow smaller with each retest, stop when bottlenecking is caused by the CPU
  - Then you may have the consider option of increasing CPU power

Example Performance Test Cases

- **Test Case 01:** Verify response time is not more than 4 secs when 1000 users access the website simultaneously
- **Test Case 02:** Verify response time of the Application Under Load is within an acceptable range when the network connectivity is slow
- **Test Case 03:** Check the maximum number of users that the application can handle before it crashes
- **Test Case 04:** Check database execution time when 500 records are read/written simultaneously
- **Test Case 05:** Check CPU and memory usage of the application and the database server under peak load conditions
- **Test Case 06:** Verify the response time of the application under low, normal, moderate, and heavy load conditions

Execution

- 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

- Understanding the basics of Performance Testing
- Overview of various Performance Testing Types
  - Load, Stress, Scalability, Capacity, Volume
- Key metrics to measure
  - Response time, Throughput, Resource Utilization
- Introduction to tools
  - JMeter, BlazeMeter, K6, Loader.io
- How to do Performance Testing?
  A step-by-step approach

Questions?

SoftUni

Software University | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids | Finance Academy

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg