### **Persistent: Runtime settings in detail?** Includes think time, pacing, logging, iteration count, browser cache, and error handling. Parameterization : Parameterization is the process of replacing hard-coded values (like usernames, IDs, etc.) with variable data from external sources (like .csv files) to simulate real-time user behavior.

**What are Web\_reg\_save\_param arguments ?**  
  
web\_reg\_save\_param is used in **LoadRunner** to capture **dynamic values** from server responses.

| **Argument** | **Description** |
| --- | --- |
| "paramName" | Name of the parameter to store the dynamic value. |
| "LB=..." | Left Boundary – text before the target value. |
| "RB=..." | Right Boundary – text after the target value. |
| "Ord=n" | *Optional*: Which occurrence to capture (e.g., Ord=1, Ord=ALL). |
| "SaveLen=n" | *Optional*: Number of characters to save. |
| "SaveOffset=n" | *Optional*: Skip n characters from captured value start. |
| "NotFound=empty" | *Optional*: If value not found, return blank instead of error. |
| "Search=Body/Header/All" | *Optional*: Where to search the dynamic value. Default is Body. |
| "RelFrameID=ALL" | *Optional*: For handling frames or iframes. |
| "Scope=All/Body/Headers" | *Optional*: Limit search scope within response. |

### **Different protocols?** ➤ HTTP/HTML, Web Services, TruClient, Java, Oracle NCA, SAP, etc. **How to remove unnecessary URLs after recording?** ➤ Use recording options and manually remove irrelevant requests (e.g. ads, analytics). **How to give % to actions in Run logic?** ➤ Use Run Logic > Random > Assign % to actions (e.g. 30%, 30%, 40%). **Browser cache – good or not?** ➤ Disable browser cache to simulate new user behavior. **Different load test executions?** ➤ Load Endurance, Stress, Spike (sudden load). we have 3 requirement and one requirement taking 180 seconds but connection timeout at 60 seconds, how will happen? The request will **fail at 60s** due to timeout unless you **increase the timeout setting** to match or exceed 180 seconds. CTS: ****1. Challenges Faced in Sampling & How You Resolved It:****

**Challenge:**

* Incorrect or insufficient sampling leading to unrealistic user behavior.
* Sampling only peak traffic, missing business-hour variations.

**Solution:**

* Coordinated with business teams to understand real user flow.
* Collected production logs and analytics to model realistic workloads.
* Used pacing, think time, and test data variation to simulate real behavior.

**3. Thread Dump & Heap Dump - Command to Take It:**

**Thread Dump:**

bash

CopyEdit

jstack <PID> > threadDump.txt

**Heap Dump:**

bash

CopyEdit

jmap -dump:format=b,file=heapDump.hprof <PID>

Or use JVM options:

bash

CopyEdit

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/path

**4. How Often Should Heap & Thread Dumps Be Taken?**

* During performance tests:
  + Take dumps **at steady state**, **when spikes occur**, and **at end of test**.
* During issues:
  + Heap dump → once or twice.
  + Thread dump → 3-5 times, 10 seconds apart, to observe thread state trends.

**5. Application Architecture (Brief Overview):**

* **3-tier or microservices**: Web → App → DB.
* **Load Balancer** distributes traffic.
* **App Layer** has multiple nodes (stateless preferred).
* **DB Layer** may be SQL/NoSQL with replication/failover.

**6. Load Balancer & Algorithm:**

**Load Balancer Role:**

* Distributes incoming traffic across multiple servers.

**Algorithms:**

* **Round Robin** – equal distribution.
* **Least Connections** – based on active sessions.
* **IP Hash** – based on client IP for session persistence.
* **Weighted** – based on server capacity.

**7. Workload Modeling:**

* Analyze real user traffic (from logs, analytics).
* Identify peak/off-peak hours.
* Define:
  + **Transaction mix**
  + **Concurrent users**
  + **Pacing/Think time**
  + **User ramp-up/down**

**8. Test Plan vs Test Strategy:**

**Test Strategy (High-level):**

* Objectives, tools used, environment, approach (load, stress, soak).

**Test Plan (Detailed):**

* Specific scenarios, user load, scripts, ramp-up, schedule, entry/exit criteria.

### gc , major GC, Minor GC ✅ 1. GC (Garbage Collection):

**Definition:**  
Automatic process in Java to reclaim memory by removing unused (unreachable) objects from the heap.

**✅ 2. Minor GC:**

**What it is:**

* Happens in the **Young Generation**.
* **Fast and frequent**.
* Reclaims memory from **short-lived** objects.
* Uses **copying algorithm** (from Eden to Survivor space).

**Impact:**

* **Short pause time**.
* Usually doesn’t affect performance significantly unless very frequent.

**✅ 3. Major GC (also called Full GC):**

**What it is:**

* Happens in the **Old Generation** (Tenured).
* Collects **long-lived** objects.
* Also may collect **Young Gen + Old Gen + Metaspace** together (Full GC).

**Impact:**

* **Longer pause time**.
* **Affects application response time** if too frequent.
* Usually triggered when:
  + Old Gen is full
  + System.gc() is called
  + Heap compaction is needed

**10. Cloud Technology Experience:**

* **Experience in:**
  + **AWS/GCP/Azure** performance testing.
  + Auto-scaling behavior, cloud load balancers, S3/Blob storage performance.
* **Tools used:** JMeter in EC2, Azure DevOps Pipelines, Grafana Cloud.

**11. 90th Percentile & Standard Deviation:**

* **90th Percentile:**  
  90% of response times are below this value (used to measure user experience).
* **Standard Deviation (SD):**  
  Shows response time variation. Lower SD = stable performance.

**Work load modelling**

### ✅ ****Workload Modelling in Performance Testing****

**Definition:**  
Workload modelling is the process of simulating real-world user behavior and system usage patterns in a test environment. It ensures your performance test reflects actual production usage.

### 🔑 ****Key Components of Workload Modelling:****

| **Element** | **Description** |
| --- | --- |
| **User Load** | Number of concurrent or total virtual users |
| **Transaction Mix** | Percentage split of different user actions (e.g., login – 20%, search – 40%) |
| **Think Time** | Simulated delay between user actions to mimic human behavior |
| **Pacing** | Delay between iterations of a virtual user’s script |
| **Test Duration** | How long the test runs (e.g., 1 hour, 8 hours for soak test) |
| **Ramp-Up/Ramp-Down** | Gradual increase or decrease of users to simulate load pattern |

**🔹 1. Bottleneck Issues Faced So Far**

**Common Bottlenecks:**

* **CPU Spike:** App server CPU >85% under load.
* **Memory Leak:** Memory usage growing, not releasing.
* **DB Query Slowness:** Long-running queries, missing indexes.
* **GC Overhead:** Frequent full GCs during stress test.
* **Thread Contention:** Threads waiting for resources.

**How Resolved:**

* Optimized DB queries and added indexes.
* Tuned JVM heap size and GC algorithm.
* Worked with Devs to implement caching for repeated calls.
* Increased thread pool/configured async processing.

**🔹 2. Sampling Challenges Faced (Before in R&D / Projects)**

**Challenges:**

* Sample size too small or not diverse (missed edge cases).
* Sampling only peak hours (ignored normal load behavior).
* Inconsistent logs/analytics data.

**Resolutions:**

* Took samples from full-day logs.
* Used APM tools like Dynatrace/NewRelic for realistic flow.
* Validated with BA/dev team for business-critical scenarios.

**🔹 3. How Frequently You Take Heap Dump & Thread Dump?**

**During Test:**

* **Heap Dump:** Only during memory-related issues (e.g., OutOfMemoryError), usually **once** at failure point.
* **Thread Dump:**
  + During high CPU/response time issues.
  + Taken **3-5 times**, **10 seconds apart** to compare thread behavior.

**🔹 4. In Endurance Test, You See Low Response Time (RT) – Possible Reasons?**

**Low RT During Endurance Test – Causes:**

* App is well-optimized or underutilized.
* Backend caching (results returned faster).
* Load is not high enough to stress the system.
* DB connection pool or JVM memory tuned well.
* GC or thread pool not causing bottlenecks.

🔸 *Still validate memory leaks, CPU trends, DB locks to confirm no hidden issues.*

**🔹 5. Final Report – Key Metrics to Present:**

| **Metric** | **Why It’s Important** |
| --- | --- |
| Avg Response Time | Overall performance indicator |
| 90th/95th Percentile RT | User experience during peak load |
| Throughput (TPS) | System capacity (transactions/sec) |
| Error Rate (%) | Stability, reliability of application |
| CPU & Memory Usage | Resource consumption per tier |
| GC Stats (GC time %, count) | JVM health, pause impact |
| DB Query Response Time | Backend efficiency |
| Hits/sec & Requests/sec | Load consistency |
| Network Usage (optional) | Detect bandwidth bottlenecks |
| Pass/Fail for each Scenario | Based on SLA/KPI |

### Wabtech: 1. OOPS Concepts:

The 4 main pillars of OOPS are:

1. **Encapsulation:** Wrapping data (variables) and code (methods) into a single unit (class).
2. **Abstraction:** Hiding implementation details and showing only essential features (e.g., abstract classes, interfaces).
3. **Inheritance:** Acquiring properties and behavior of one class into another (parent-child relationship).
4. **Polymorphism:** The ability to take multiple forms (e.g., method overloading and overriding).

**2. HTTP Methods & Difference Between PUT and POST:**

* **GET:** Retrieve data from the server.
* **POST:** Create a new resource (non-idempotent).
* **PUT:** Update a resource or create if not exists (idempotent).
* **DELETE:** Remove a resource.
* **PATCH:** Partial update of a resource.

**PUT vs POST:**

* **PUT:** Replaces or creates a resource at a specific URI. Idempotent (multiple calls result in the same output).
* **POST:** Creates a resource, usually at a server-generated URI. Not idempotent.

**3. Database Life Cycle & SDLC:**

* **Database Life Cycle:**
  1. Requirement Analysis.
  2. Database Design (ER modeling).
  3. Implementation (tables, schemas).
  4. Testing (queries, indexes).
  5. Deployment.
  6. Maintenance (backup, performance tuning).
* **SDLC (Software Development Life Cycle):**
  1. Requirement Gathering.
  2. Design (HLD, LLD).
  3. Development.
  4. Testing.
  5. Deployment.
  6. Maintenance.

**4. Explain Project (Performance Testing Example):**

* **Domain:** E-commerce/Banking.
* **Role:** Performance Testing Engineer.
* **Tools:** LoadRunner/JMeter, Dynatrace for monitoring.
* **Tasks:** Requirement gathering, workload modeling, scripting, test execution, bottleneck identification, and reporting.
* **Challenges:** Correlation, DB bottlenecks, CPU spikes.

**5. Difference Between Checked & Unchecked Exceptions:**

* **Checked Exception:**
  + Checked at **compile-time**.
  + Must handle using try-catch or throws.
  + Examples: IOException, SQLException.
* **Unchecked Exception:**
  + Checked at **runtime**.
  + Derived from RuntimeException.
  + Examples: NullPointerException, ArithmeticException.

**6. Method Overloading & Overriding:**

* **Overloading:**
  + Same method name but different parameters (compile-time polymorphism).
  + Example:

java

CopyEdit

void add(int a, int b);

void add(double a, double b);

* **Overriding:**
  + Child class provides its own implementation of a parent class method (runtime polymorphism).
  + Example:

java

CopyEdit

@Override

void display() { ... }

**7. Basic Git Commands:**

* git init – Initialize a repository.
* git clone <url> – Clone a repository.
* git add . – Stage changes.
* git commit -m "message" – Commit changes.
* git push – Push changes to remote.
* git pull – Get latest changes.
* git status – Check file status.
* git log – View commit history.

**8. SQL – LEFT OUTER JOIN vs RIGHT OUTER JOIN:**

* **LEFT OUTER JOIN:** Returns all records from the **left table** and matched records from the right table (NULL for unmatched).
* **RIGHT OUTER JOIN:** Returns all records from the **right table** and matched records from the left table (NULL for unmatched).

**9. Difference Between DELETE and TRUNCATE:**

* **DELETE:**
  + Removes rows one by one (can use WHERE clause).
  + Slower, logs each row deletion.
  + Can be rolled back.
* **TRUNCATE:**
  + Removes all rows (no WHERE).
  + Faster, minimal logging.
  + Cannot be rolled back (DDL).

**GE Health Care:**

**✅ 1. Heap Dumps & Memory (C Language Insight not needed here)**

* **Heap Dump**: It's a snapshot of Java process memory at a point in time.
* Used to **identify memory leaks**, large object retention, and memory consumption.
* **When taken**: During high memory usage or out-of-memory errors.
* **Tools**: Eclipse MAT, VisualVM, JProfiler.

**✅ 2. Types of Bottlenecks**

In GE Healthcare CMS/SGI applications, common bottlenecks include:

* **CPU Bottlenecks**: Caused by thread loops, batch jobs.
* **Memory Bottlenecks**: Due to object retention or memory leaks.
* **Database Bottlenecks**: Slow queries or unindexed columns.
* **3rd Party Integration**: Slow API responses.
* **Load Balancer Issues**: Improper routing or sticky session missing.

**✅ 3. Jenkins Usage**

* Used to **automate test execution** (e.g., JMeter/LoadRunner tests).
* **CI/CD Integration**: Tests run after each deployment.
* **Plugins**: Can generate HTML reports, send email alerts, or trigger tests post-build.
* **Example**: GE used Jenkins pipeline to schedule nightly API performance tests.

**✅ 4. Bottleneck Issues You Faced (Real-Time Answer)**

* Faced **high CPU usage** during stress test.
* Found threads were stuck due to **code-level loops** and **frequent GC activity**.
* We used **AppDynamics** to analyze CPU/thread dump.
* Fix: Tuned GC params, fixed code logic, and introduced caching.

**✅ 5. Application Architecture at GE**

* **End-to-End Setup**:
  + **Frontend**: React/Angular
  + **Middle Layer**: Java Spring Boot APIs
  + **Backend**: Microservices with REST APIs
  + **DB**: Oracle/PostgreSQL
  + **3rd Party**: Lab/healthcare device integrations
  + **Cloud**: Azure/GCP
  + **Monitoring**: Prometheus + Grafana, or Dynatrace

**✅ 6. End-user Performance Testing**

* We test real user journeys: login, patient search, lab report upload.
* Focus on:
  + **Response Time**
  + **Concurrent Users**
  + **Think Time**
* Tools used: LoadRunner (VuGen), JMeter

**✅ 7. Memory Leaks – What, When, How**

* **Memory Leak**: Unused objects remain referenced, not garbage collected.
* **Causes**:
  + Improper cleanup in loops
  + Static object references holding memory
* **Detection**:
  + Continuous memory increase in monitoring tools.
  + Heap dump analysis.
* **Tools**: VisualVM, Eclipse MAT, AppDynamics

**✅ 8. Distributed Load Testing**

* Used when one machine can't generate enough users.
* Setup:
  + **Master-Slave** (in JMeter)
  + **Controller - Load Generator** (in LoadRunner)
* Used in large-scale tests for simulating global user base.

**✅ 9. Live Linux Commands for Performance Testing**

* top, htop: CPU/memory usage
* iostat, vmstat: Disk/IO and memory info
* netstat, sar: Network usage
* jstack: Java thread dump
* jmap -dump: Create heap dump
* tail -f /var/log/app.log: Monitor logs in real-time

**✅ 10. Parameterization, JConsole, Jvinculum**

* **Parameterization**:
  + Replace static values with dynamic data (e.g., login IDs).
  + In JMeter: Use CSV Data Set Config.
  + In LoadRunner: Use parameter files.
* **JConsole**:
  + Java monitoring tool.
  + See heap, GC, thread usage, MBeans.
* **Jvinculum**:
  + Likely refers to middleware integration tool in GE or another client.
  + Used for healthcare data flow and system interoperability (please confirm exact usage in your project).

### ATOS Syntel: 1. Application Architecture

**Example for GE Healthcare or similar enterprise:**

* **Frontend (UI Layer)**: Angular/React for user interface – used by doctors or admins.
* **API Gateway**: Manages routing, authentication (OAuth2), and throttling.
* **Backend Services**:
  + Microservices developed in Java Spring Boot.
  + Handles business logic (e.g., patient management, lab results).
* **Database Layer**: PostgreSQL/Oracle – stores medical records, images, user data.
* **3rd Party Integration**: Medical devices, insurance, lab report services.
* **Cloud Setup**: Deployed on Azure or AWS with containerization (Docker/Kubernetes).
* **Monitoring & Logging**: Dynatrace, ELK Stack, Grafana for logs and metrics.

**✅ 2. Challenges Faced in Scripting**

Real-world scripting challenges and how you solved them:

* **Dynamic values**: Session tokens, order IDs keep changing.
  + **Solution**: Used **correlation** (web\_reg\_save\_param in LoadRunner, **Regular Expression Extractor** in JMeter).
* **Authentication tokens**:
  + APIs required Bearer tokens that expired quickly.
  + **Solution**: Automated token generation using pre-request scripts or login API.
* **Multipart/Form-data requests**:
  + Complex file uploads during lab report scenarios.
  + **Solution**: Analyzed raw request from Postman or Fiddler and configured manually.
* **Data Dependency**:
  + Test steps required valid data from previous steps (like patient ID).
  + **Solution**: Used **parameter passing** and **correlation** between requests.
* **SSL & Cert issues**:
  + Secure environments rejecting requests.
  + **Solution**: Imported client certificates and disabled unwanted redirects in JMeter.

**✅ 3. Performance Bottlenecks Faced**

Real-time bottlenecks faced in testing projects:

| **Type** | **Issue Faced** | **Resolution** |
| --- | --- | --- |
| CPU | High CPU (95%) during load | Identified hot threads using jstack, optimized code loops |
| Memory | Gradual memory rise | Found memory leak via heap dump, resolved unclosed DB connections |
| DB | Slow queries | Indexing and query tuning |
| 3rd Party APIs | Timeout or slowness | Added retry logic, isolated in test reports |
| Load Balancer | Session stickiness missing | Updated LB config to maintain session |
| App Threads | Thread pool exhausted | Increased pool size and optimized thread handling |

**Tech Mahindra:**✅ **1. About VUGen Scripting (VuGen Coding Basics)**

**Scenario (based on your message)**:  
"We have to simulate buyers placing orders and sellers fulfilling them. We need to validate the complete flow and compare expected vs actual."

**How to script in VuGen:**

* **Record Script** using **Web (HTTP/HTML)** or **Web Services** protocol.
* Steps to cover:
  + Buyer login
  + Product search
  + Add to cart
  + Place order
  + Seller login (optional)
  + Order fulfillment/update
* **Enhancements**:
  + **Parameterization**: Buyer IDs, product IDs
  + **Correlation**: Dynamic order ID, session tokens
  + **Checkpoints**: web\_reg\_find to verify response content
  + **Transactions**: lr\_start\_transaction() and lr\_end\_transaction() to measure response time

### ✅ ****2. What are the possible causes of high response time?****

| **Cause** | **Description** | **Resolution** |
| --- | --- | --- |
| **Batch Jobs** | Scheduled tasks consuming high CPU during test | Isolate test window from batch runs |
| **3rd Party APIs** | External calls slowing down app | Use mock/stubs during test or monitor dependencies |
| **Load Balancer Misconfig** | Improper routing or no session persistence | Update LB config for sticky sessions |
| **DB Issues** | Slow queries, table locks, no indexing | Optimize SQL, add missing indexes |
| **High GC Activity** | Due to memory leaks or large object retention | Tune heap, fix object retention |

### ✅ ****3. Can we create or update scripts during load test?****

* **Technically Possible?** Yes — but **not recommended** during live load testing.
* **Why?**
  + Might cause inconsistencies or environment instability.
  + Any new script must be **validated separately**.
* **Best Practice**:
  + Create and test new scripts in staging or QA first.
  + Include them in the next planned test cycle.

### ✅ ****4. How to Add Bearer Auth Token in JMeter (or Postman)****

#### **JMeter:**

1. Add **HTTP Header Manager**.
2. Add key-value:

nginx

CopyEdit

Authorization : Bearer ${access\_token}

1. Optionally, fetch the token via **login API** using a separate sampler, and extract with **JSON Extractor**.

**What is Parameterization?**

**Parameterization** is the process of **replacing hard-coded values** in your test script with **dynamic test data** to simulate real-world scenarios.

**🔹 Why Parameterization is Needed?**

* To **avoid server-side caching**
* To simulate **multiple users with different inputs**
* To **reduce correlation errors** (due to reused values)

### IBM: ✅ 1. Architecture in JMeter (Achtices on Jmelex)

**JMeter Architecture Components**:

* **Test Plan** → Root element, contains all configurations & logic.
* **Thread Group** → Simulates virtual users.
* **Samplers** → HTTP Request, JDBC, SOAP, etc.
* **Logic Controllers** → If, Loop, While, etc.
* **Listeners** → Results view, summary, graphs.
* **Config Elements** → CSV Data Set, HTTP Header Manager.
* **Pre/Post-Processors** → Extractors, assertions, etc.

**✅ 2. Test Plan in JMeter (Imek Test-plan)**

* A **Test Plan** defines:
  + User load simulation
  + Request flow
  + Server configs
  + Data parameterization
  + Assertions and reports

Example flow:

Test Plan → Thread Group → HTTP Samplers → CSV Config → Listeners

**✅ 3. VTS in VuGen (Virtual Table Server)**

* **VTS (Virtual Table Server)**: Allows sharing and reusing dynamic data across Vusers in LoadRunner.
* **Why use it?**
  + Dynamically assign data (like session IDs, tokens) between scripts.
  + Useful in **multi-user order flows** (Buyer & Seller).
* **How it works**:
  + One Vuser writes a value to VTS.
  + Another Vuser reads it in real time.

**✅ 4. Alpha Movement – Dynamic Value Creation in VuGen**

(*"Alpla mument whe dynamical creating in vigen?" → Alpha-numeric dynamic data generation*)

* In VuGen, to create **dynamic strings**:

c

CopyEdit

char dynamicVal[20];

sprintf(dynamicVal, "User\_%d", rand()%1000);

lr\_save\_string(dynamicVal, "dynamicUser");

* Use lr\_save\_string() and sprintf() to build unique IDs or values.
* Helpful for generating **usernames, emails, orders**, etc.

**✅ 5. Suck Set UNG = Sync vs Async calls**

(*“Suck Set UNG / somchoncus” → Synchronous vs Asynchronous Calls*)

* **Synchronous**: Request waits for response before proceeding.
  + Example: Login → Get Dashboard
* **Asynchronous**: Request doesn’t wait; response is processed later.
  + Example: AJAX calls, background job status check.
* In VuGen:
  + Handle async using **web\_reg\_async\_attributes**.
  + In JMeter, handle using **Parallel Controller** or JavaScript logic.

**✅ 6. Encryption in VuGen (Encryphus in vuesch)**

* If the application encrypts data (passwords, tokens):
  + Capture the logic using **JavaScript decoding** or **custom DLLs**.
  + Use **C code or external libraries** to replicate the encryption.
* **Example**: AES, SHA256 hashing done in frontend must be replicated in VuGen.

**✅ 7. Correlation in VuGen & JMeter**

* **Correlation** = Capturing and reusing dynamic values (sessionID, orderID)

**In VuGen**:

c

CopyEdit

web\_reg\_save\_param("sessionID", "LB=Set-Cookie: JSESSIONID=", "RB=;", LAST);

**In JMeter**:

* Use **Regular Expression Extractor**
* Example:

regex

CopyEdit

Reference Name: sessionID

Regex: JSESSIONID=(.+?);

**✅ 8. Workload Modeling**

* Workload modeling defines how different user types use the system.
* Steps:
  + Identify **user personas** (Admin, Doctor, Patient)
  + Define **Business Scenarios**
  + Set **test mix** (e.g., 40% patients, 30% doctors, 30% admin)
  + Define **Load Distribution** and **Think Time**

**✅ 9. Sampling Challenges**

* **Sampling** is the way JMeter or LoadRunner collects data points.
* **Challenges**:
  + Too much data = overhead
  + Sparse sampling = less accuracy
* Ensure appropriate **granularity** in results to analyze trends.

**✅ 10. If SLA is breached (Wash SLA = SLA fail)**

**Example SLA**: 95% of transactions should respond in <2 seconds.

**If SLA is breached**:

1. **Check transaction breakdown**.
2. **Identify slow APIs** or scenarios.
3. Correlate with **CPU/Memory/DB** metrics.
4. Use **APM tool** (Dynatrace, AppD) to trace bottlenecks.
5. Create **action plan**: optimize code, queries, increase infra.

**✅ 11. Coordination Challenges (cordaten challange)**

In multi-team, multi-module apps, these are common:

| **Challenge** | **Impact** | **Solution** |
| --- | --- | --- |
| Late API changes | Script failure | Keep mock/stub ready |
| Test data sync | Invalid scenarios | Use automation to refresh data |
| Infra readiness | Missed timelines | Coordinate with DevOps early |
| Version mismatch | Unexpected errors | Sync with build release team |

### Deloitte: ✅ 1. ****Workload Modeling (Work Stood Modeling – Example)****

**Definition**:  
Workload modeling is the process of **designing realistic test scenarios** by identifying the types of users, actions they perform, and how often they do it.

#### 🔹 **Real-time Example (E-commerce site like GE Healthcare portal)**

| **User Type** | **Action** | **% Distribution** | **Frequency (Per hour)** |
| --- | --- | --- | --- |
| Patient | Login, View reports | 50% | 60 users/hour |
| Doctor | Login, Add diagnosis | 30% | 30 users/hour |
| Admin | Manage users/reports | 20% | 10 users/hour |

**Total Virtual Users**: 100  
**Duration**: 1 hour  
**Ramp-up**: Gradual over 10 mins  
**Think Time**: 2–5 seconds between actions

### ✅ 2. ****Scripting Challenges (Script Chalkinge – Example)****

Real-time scripting problems and how to solve them:

| **Challenge** | **Example** | **Solution** |
| --- | --- | --- |
| **Dynamic session token** | Login returns sessionID=XYZ123 | Use **correlation** (web\_reg\_save\_param in VuGen, **Regex Extractor** in JMeter) |
| **Encoded password** | Login form sends base64 password | Use encoding logic via JavaScript or C in script |
| **File upload in API** | Multipart/form-data with dynamic boundary | Use **Fiddler/Postman** to inspect and replicate manually |
| **Bearer Token expires** | Auth token invalid after 5 mins | Automate token fetch using a pre-login API request |

### ✅ 3. ****Difference between web\_reg\_save\_param & web\_reg\_save\_param\_ex****

**Used in VuGen (LoadRunner) for correlation**

| **Feature** | **web\_reg\_save\_param** | **web\_reg\_save\_param\_ex** |
| --- | --- | --- |
| Basic/Standard | Yes | No |
| Advanced options | No | Yes |
| Can capture **non-HTML** (JSON, XML, binary) | Limited | **Yes** |
| Scope control (Headers, Body, Both) | No | **Yes** |

#### 🔹Example of web\_reg\_save\_param\_ex:

c

CopyEdit

web\_reg\_save\_param\_ex(

"ParamName=authToken",

"LB=Authorization: Bearer ",

"RB=\\r\\n",

"Search=Headers",

LAST);

### ✅ 4. ****Web Page Diagnostics (web page Diagonethic)****

Tools to analyze web app performance at page level:

* **Chrome DevTools**:
  + Check Time to First Byte (TTFB), DOM Load, and Full Load.
  + Analyze JS/CSS/image load delays.
* **Fiddler** / **Wireshark**:
  + Capture full request-response.
  + Identify heavy resources or failed calls.
* **PageSpeed Insights / Lighthouse**:
  + Performance audit (load time, blocking resources, etc.)

### ✅ 5. ****Execution Order in JMeter (Imeler Execution Order)****

#### **Default Execution Order in JMeter:**

1. **Config Elements**
2. **Pre-Processors**
3. **Timers**
4. **Samplers** (e.g., HTTP Request)
5. **Post-Processors**
6. **Assertions**
7. **Listeners**

#### 🔹Example:

* If you use a **JSON Extractor** (post-processor) for response data, it will only run **after** the HTTP request sampler.
* Place **Header Managers** and **CSV Configs** at the top of the thread group.

### ✅ 6. ****When to Use HTML vs URL-Based Scripting in VuGen****

| **Mode** | **When to Use** | **What it Captures** |
| --- | --- | --- |
| **HTML-based** | Default option, use when app is **browser-driven** | Captures high-level user actions |
| **URL-based (HTML Off)** | When needing **detailed low-level HTTP requests** | Every request, redirection, and image call |
| **Switching Mode** | Before recording, go to **Recording Options → HTML Advanced** | Choose between HTML & URL mode |

**CGI:**In **LoadRunner (VuGen)**, there are several **default (built-in) variables and functions** that are commonly used for scripting, debugging, and performance measurements.

Here’s a list of the **most important default variables and their usage**:

**✅ 1. lr\_eval\_string()**

* Evaluates and replaces **parameter values** within a string.
* Example:

lr\_output\_message("Username is: %s", lr\_eval\_string("{UserID}"));

**✅ 2. lr\_get\_vuser\_id()**

* Returns the **unique ID** of the current Virtual User (Vuser).
* Example:

int vuser\_id = lr\_get\_vuser\_id();

**✅ 3. lr\_get\_transaction\_duration("TxnName")**

* Returns the **elapsed time** of a transaction in seconds.
* Example:

double duration = lr\_get\_transaction\_duration("Login\_Transaction");

**✅ 4. lr\_get\_host\_name()**

* Returns the name of the **host machine** running the Vuser.
* Example:

lr\_output\_message("Host Name: %s", lr\_get\_host\_name());

**✅ 5. lr\_get\_attrib\_string("AttributeName")**

* Gets the value of an **attribute** (from Controller or command line).
* Example:

char \*env = lr\_get\_attrib\_string("TestEnv");

**✅ 6. lr\_get\_user\_group\_name()**

* Returns the name of the **Vuser Group**.
* Useful for reporting or conditionally branching logic.

**✅ 7. lr\_get\_scenario\_string()**

* Retrieves the name of the **scenario** the Vuser is running in.

**✅ 8. {pIteration} (Built-in parameter variable)**

* Refers to the **current iteration number**.
* Example:

lr\_output\_message("Current Iteration: %s", lr\_eval\_string("{pIteration}"));

**✅ 9. {VuserID}**

* Built-in parameter available in Controller to identify each Vuser.
* Can be passed as part of API payloads, logs, or test IDs.

**✅ 10. {GroupName}**

* Represents the group name from the Controller (if parameterized).
* Useful in large-scale tests where multiple groups run different scripts.

**✅ Usage Tip:**

You can use these variables inside **transaction names**, **log messages**, or **dynamic test logic** to:

* Debug effectively
* Trace user activity
* Track test progress

Would you like a **cheat sheet PDF** for LoadRunner functions and default variables? I can prepare it for quick interview review.

**Some coorelation value is there and we have to add 20 and pass to the next request?**

**Great question! If you capture a correlation value (e.g., a numeric ID like itemID = 100), and you want to add 20 to it and pass it to the next request, here’s how you do it in VuGen (LoadRunner):**

### ✅ ****Step-by-Step Example (VuGen – C Scripting)****

#### 🔹1. **Capture the value using correlation**

c

CopyEdit

web\_reg\_save\_param("itemID",

"LB=itemID=",

"RB=&",

LAST);

#### 🔹2. **Convert captured string to int, add 20, then convert back**

c

CopyEdit

int originalID = atoi(lr\_eval\_string("{itemID}")); // Convert to integer

originalID += 20; // Add 20

// Convert back to string and save as new parameter

char newID[10];

sprintf(newID, "%d", originalID);

lr\_save\_string(newID, "ModifiedItemID");

#### 🔹3. **Use the new value in the next request**

c

CopyEdit

web\_submit\_data("NextRequest",

"Action=https://example.com/itemDetails",

ITEMDATA,

"Name=itemID", "Value={ModifiedItemID}", ENDITEM,

LAST);

### ✅ ****Quick Summary****

| **Step** | **Action** |
| --- | --- |
| 1 | Capture value using web\_reg\_save\_param |
| 2 | Use atoi() to convert to int |
| 3 | Add +20 |
| 4 | Use sprintf() to convert back to string |
| 5 | Use lr\_save\_string() to store new param |

**web\_set\_user?**

### ✅ web\_set\_user in LoadRunner (VuGen)

web\_set\_user is used in **VuGen** to handle **basic authentication** (HTTP-level, not application-level). It sets the **username, password, and domain** for requests to access secured servers.

### 📌 ****Syntax:****

c

CopyEdit

web\_set\_user("username", "password", "host:port");

### 📘 ****Parameters:****

| **Parameter** | **Description** |
| --- | --- |
| "username" | The login user name (e.g., domain\\user or just user) |
| "password" | Corresponding password |
| "host:port" | Target server and port (e.g., myapp.com:80) |

### ✅ ****Example:****

c

CopyEdit

web\_set\_user("mydomain\\charan", "MyPassword123", "myapp.gehealthcare.com:443");

This will authenticate all **subsequent web requests** to the myapp.gehealthcare.com server.

diff bw sapGUI & web HTTP/HTML protocol

###### ChatGPT said:

Here's a **clear and concise difference between SAP GUI and Web (HTTP/HTML) protocols** in **LoadRunner (VuGen)** – useful for interviews or real-time scripting decisions:

### ✅ ****1. Protocol Purpose****

| **Feature** | **SAP GUI Protocol** | **Web (HTTP/HTML) Protocol** |
| --- | --- | --- |
| **Used For** | Recording SAP Desktop GUI apps | Recording browser-based apps (websites, web portals) |
| **Application Type** | SAP thick client (SAP Logon Pad) | Thin client (browser-based) |
| **Interface** | GUI-based interactions (screen, controls) | Web pages, HTTP requests & responses |

### ✅ ****2. Recording Mechanism****

| **Feature** | **SAP GUI** | **Web HTTP/HTML** |
| --- | --- | --- |
| **Protocol-level recording** | Records SAP GUI internal API calls | Records HTTP requests (GET/POST) |
| **Scripting Language** | VBScript-like SAP-specific APIs | C-based scripting |
| **Correlation** | Rarely needed due to high-level calls | Frequently needed for dynamic data (session ID, tokens) |

### ✅ ****3. Usage Complexity****

| **Feature** | **SAP GUI** | **Web HTTP/HTML** |
| --- | --- | --- |
| **Ease of recording** | Easy for SAP experts, more visual | Requires understanding of HTTP, HTML |
| **Script customization** | Limited (controls like buttons, fields) | Extensive (headers, body, cookies) |

### ✅ ****4. When to Use****

| **Scenario** | **Use SAP GUI** | **Use Web HTTP/HTML** |
| --- | --- | --- |
| Testing SAP ECC, CRM modules via SAP Logon | ✅ Yes | ❌ No |
| Testing SAP Fiori, SAP Portal via browser | ❌ No | ✅ Yes |
| Working with non-SAP web apps (Flipkart, GE portals) | ❌ No | ✅ Yes |

### ✅ ****5. Protocol Combination (if needed)****

In some SAP projects, you may use:

* **SAP GUI** for backend workflows (TCode based)
* **Web HTTP/HTML** for frontend apps (Fiori/UI5)

### 📝 Example:

* **SAP GUI**:

c

CopyEdit

sapgui\_set\_text("User", "charan", LAST);

sapgui\_press\_button("Login", LAST);

* **Web HTTP/HTML**:

c

CopyEdit

web\_submit\_data("Login",

"Action=https://app.example.com/login",

ITEMDATA,

"Name=username", "Value=charan", ENDITEM,

"Name=password", "Value=abc123", ENDITEM,

LAST);

### what all enhancement will do in SAP GUI script. 1. Parameterization

* Replacing hardcoded values with dynamic test data.
* Example:
  + User IDs, Material Codes, Customer Numbers, Sales Order IDs.
* **How**:

c

CopyEdit

sapgui\_set\_text("User", "{UserID}", LAST);

**✅ 2. Transaction Handling**

* Use **lr\_start\_transaction()** and **lr\_end\_transaction()** to measure response times for business processes.
* Example:

c

CopyEdit

lr\_start\_transaction("Login");

sapgui\_logon("user", "pass");

lr\_end\_transaction("Login", LR\_AUTO);

**✅ 3. Think Time**

* Add realistic user delays to mimic actual interaction.
* Use lr\_think\_time() between actions.
* Example:

c

CopyEdit

lr\_think\_time(5);

**✅ 4. Correlation (Rare but Possible)**

* Usually less needed in SAP GUI compared to Web, but may be required for values generated at runtime (e.g., Document Number after creation).
* Use sapgui\_get\_text() to capture values:

c

CopyEdit

sapgui\_get\_text("SalesOrderNum", "param\_SalesOrder", LAST);

**✅ 5. Error Handling / Validation**

* Add checks to verify that screens/controls are present or specific values appear.
* Example:

c

CopyEdit

sapgui\_wait\_for\_active\_window("SAP Easy Access", 20);

* Check for error messages:

c

CopyEdit

sapgui\_is\_object\_available("Error\_Message\_Box", "TRUE");

**✅ 6. Modularization**

* Break script into **functions or actions** for reusability.
  + E.g., Login, Create Sales Order, Display Order, Logout
* Helps with maintainability and scalability.

**✅ 7. Logging and Debugging**

* Add lr\_output\_message() to debug values, like captured sales order numbers.
* Useful in iterations or for test verification.

**✅ 8. Custom Code Logic**

* Write logic for conditional flows (e.g., if order exists, cancel it).
* Use standard **C language** syntax:

c

CopyEdit

if (strcmp(lr\_eval\_string("{OrderStatus}"), "Pending") == 0) {

// cancel logic

}

**✅ 9. Data-Driven Testing**

* Use **parameter files (dat files)** to run tests with different user/data combinations in each iteration or for different Vusers.

**✅ 10. Clean-up Steps**

* Add steps like closing screens, returning to main menu, or logging out cleanly.

**✅ Optional Enhancements (Advanced):**

* **Use VTS (Virtual Table Server)** for shared data across users (e.g., passing document number from creator to approver).
* **Integrate with Jenkins** for CI/CD.
* **SAP Message Validation**: Check SAP status bar messages after transactions.  
    
    
    
    
  **Oracle : 1st round.**

**🔶 1. Integration Testing – Explanation**

**Integration Testing** checks the interaction between integrated units/modules of an application. It ensures combined parts work correctly together.

✅ **Example:**

* Suppose you have:
  + Module A = Login Page
  + Module B = Dashboard

In Integration Testing, you will verify:

* After login (A), does Dashboard (B) load correctly?
* Is the user data from A passed correctly to B?

🛠️ **Types:**

* Top-down
* Bottom-up
* Big Bang
* Sandwich

**🔶 2. Explain about LoadRunner (LR)**

**LoadRunner** is a performance testing tool by Micro Focus used to test applications under load.

✅ **Main Components:**

* **VuGen (Virtual User Generator)**: Create test scripts
* **Controller**: Design the load scenario
* **Load Generators**: Simulate multiple users
* **Analysis**: Analyze results (throughput, response time, hits/sec, etc.)

🛠️ **Use Cases:**

* Web, SAP, Oracle apps, APIs
* Detect bottlenecks like CPU, DB, memory

**🔶 3. Java Program – Find Duplicate Words in String**

java

CopyEdit

import java.util.\*;

public class DuplicateWords {

public static void main(String[] args) {

String input = "test saf telt ake test";

String[] words = input.toLowerCase().split(" ");

Map<String, Integer> wordCount = new HashMap<>();

for (String word : words) {

wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);

}

System.out.println("Duplicate words:");

for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {

if (entry.getValue() > 1) {

System.out.println(entry.getKey() + " - " + entry.getValue());

}

}

}

}

✅ **Output for "test saf telt ake test"**

bash

CopyEdit

Duplicate words:

test - 2

**🔶 4. Asked Completely Concept-Based Questions Like:**

**"For a New App – What are the things we need to check?"**

As a **Performance Engineer**, you must consider:

✅ **Checklist Before Testing a New App:**

* Understand architecture (frontend, backend, DB, cache, API layers)
* Protocol used (HTTP, HTTPS, SAP GUI, Oracle NCA)
* Identify business critical flows (login, search, payment)
* User load expectations (peak vs. average)
* Think time, pacing, SLA for response times
* Authentication type (basic, token, SSO)
* Environment readiness (dev, QA, pre-prod)
* Monitoring tools access (AppDynamics, Dynatrace, Grafana)

📌 **Examples of Flows to Test:**

* Login
* Search
* Add to cart
* Checkout
* API Load Test