**Software Testing – SDLC (Software Development Life Cycle)**

**1. Requirement Analysis & Information Gathering**

* First step of SDLC where project details are collected from the client.
* **Client Interaction:** Understand the project goals, budget, and delivery timelines.
* **Key Considerations:**
  + **Type of Project** (e.g., Website, Application)
  + **Resources Required:** Developers, Testers, Designers
  + **Deadline & Costing:** Define the timeline and cost estimation
  + **Team Allocation:** Allocate the project team including interns (e.g., 1 intern per 50 PHP tools/technologies)

**Example Project Plan:**

* **Team Roles:**
  + Business Analyst (BA)
  + Project Manager
  + Team Leader
* **Budget:** ₹10 Lakh
* **Duration:** 3 Months
* **Resources:** 5 Members
* **Resource Cost:** ₹1 Lakh per person
* **Project Cost:** ₹5 Lakh to ₹10 Lakh
* **Website Count:** 15–20
* **Additional Features:** 5 extra pages

**2. Design**

* Involves preparing the **architecture** and **flow of the project**
* **System Architecture:** Define front-end and back-end interaction
* **Flow Diagrams:** Use Case Diagrams, Data Flow Diagrams (DFD), etc.
* **Actor Identification:** Identify users, admins, and other actors interacting with the system

**3. Development**

* Actual coding phase begins here
* **Frontend Development:** UI/UX design using HTML, CSS, JavaScript, etc.
* **Backend Development:** Server-side logic using PHP, Python, etc.
* **Web Design:** Layout, templates, responsiveness

**4. Testing**

* Ensure the software works as expected and is bug-free
* **Manual Testing:** Performed by testers without tools (e.g., functional, regression testing)
* **Automation Testing:** Using tools (e.g., Selenium, QTP) for repetitive test cases

**5. Deployment**

* The software is launched for end-users
* **Live Hosting:** Project is made available on the internet
* **Launch:** Includes versioning, hosting setup, domain linking

**6. Maintenance**

* Ongoing support after deployment
* **Free Maintenance Period:** Usually 1 year
* **Bug Fixes and Updates:** Continuous improvements based on user feedback
* **Cost Range:** ₹20,000 to ₹30,000 per year after the free period

**Software Testing Assignment**

**Software Tester – Job Profile**

* Understand **client requirements** and convert them into a **test plan** or test file.
* Perform tests on different modules like:
  + **Login via Gmail or Twitter (with OTP)**
  + **Add to Cart** functionality
* Identify **bugs or issues**
* Compare **Expected Result** vs **Actual Result**
* Ensure the app is:
  + **Bug-free**
  + **User-friendly**
  + **Easy to navigate**

**Testing Levels / Pillars**

**🔹 Level 1 – Unit Testing**

* Testing of individual components or **single functions**
* Done by developers
* Example: Login functionality only

**🔹 Level 2 – Integration Testing**

* Testing more than one function **together**
* Ensures modules work correctly when combined
* Example: Login + Add to Cart + Checkout flow

**🔹 Level 3 – System Testing**

* Testing the **entire website or application**
* Done by **automation testers**
* Ensures complete system works as per requirements

**🔹 Level 4 – UAT (User Acceptance Testing)**

* Done by **end-users or clients**
* Validates whether software meets business needs
* Feedback is collected before final deployment

**Practical Testing Types**

**🔸 1. Database Testing**

* Verify data is properly saved, updated, and fetched
* Use **SQL** for queries

**🔸 2. API Testing**

* Check if backend and frontend communicate properly
* Use **Postman** or tools like **JMeter, REST Assured**

**🔧 Testing Tools**

* **Postman:** For API Testing
* **SQL:** For Database Testing
* **Selenium:** For UI Automation Testing (optional)
* **JIRA/Bugzilla:** For bug tracking (optional)

**What is an API?**

**API = Application Programming Interface**

* Acts as a bridge between **Frontend** (user interface) and **Backend** (server)
* Used by testers to validate data transfer and business logic

**Advantages of API Testing**

1. **Language Independent**
2. Faster and more reliable than UI testing
3. Can test core logic and data easily

**Types of APIs**

| **Type** | **Description** | **Format** |
| --- | --- | --- |
| **REST API** | Lightweight, no strict rules, less secure | JSON, XML, HTML |
| **SOAP API** | Protocol-based, strict structure, more secure | XML only |

**Common API Data Formats**

| **Format** | **Full Form** |
| --- | --- |
| JSON | JavaScript Object Notation |
| XML | eXtensible Markup Language |
| HTML | HyperText Markup Language |
| XHTML | Extensible HTML |
| TEXT | Plain text |

**HTTP Methods in API**

| **Method** | **Action** |
| --- | --- |
| GET | Fetch data (Read) |
| POST | Create data |
| PUT | Full update |
| PATCH | Partial update |
| DELETE | Remove data |

**Database Testing**

**What is Database Testing?**

Database Testing ensures:

* Data is **stored**, **retrieved**, **updated**, and **deleted** correctly.
* The **backend database** works accurately with the **frontend UI**.
* Data is stored in a **structured and organized format**.

**Where is Data Stored?**

Data is stored in **databases** using systems like:

**1. RDBMS – Relational Database Management System**

* Stores data in **tables (rows & columns)**
* Data is **related** using keys
* Uses **SQL (Structured Query Language)**

**Examples of RDBMS:**

* MySQL (used with XAMPP)
* Oracle
* SQLite
* PostgreSQL

**2. NoSQL – Non-relational Databases**

* Stores data in **JSON-like documents**
* Flexible schema (not table-based)
* Used for large, fast-changing data

**Example:**

* MongoDB

**Example: Registration Form Data**

| **Field** | **Data Type** |
| --- | --- |
| Name | Text |
| Email | Text |
| Password | Text |
| Mobile No. | Number |

**Database Structure in MySQL**

**1. Database → Tables → Records (Data)**

In XAMPP, **MySQL** is used to manage database operations.

**Basic SQL Commands with Examples**

| **#** | **SQL Command** | **Purpose** | **Example** |
| --- | --- | --- | --- |
| 1 | CREATE DATABASE | Creates new database | CREATE DATABASE instagram; |
| 2 | DROP DATABASE | Deletes entire database | DROP DATABASE instagram; |
| 3 | CREATE TABLE | Defines a table | CREATE TABLE users (id INT, name VARCHAR(100)); |
| 4 | DROP TABLE | Deletes a table | DROP TABLE users; |
| 5 | SELECT | Retrieves data | SELECT \* FROM users; |
| 6 | INSERT | Adds new data | INSERT INTO users VALUES (1, 'Shahid'); |
| 7 | UPDATE | Modifies data | UPDATE users SET name='John' WHERE id=1; |
| 8 | DELETE | Removes data | DELETE FROM users WHERE id=1; |
| 9 | VIEW | Virtual table based on SELECT | CREATE VIEW user\_view AS SELECT name FROM users; |
| 10 | JOIN | Combines rows from 2+ tables | SELECT users.name, orders.product FROM users JOIN orders ON users.id = orders.user\_id; |

**Why is Database Testing Important?**

* Ensures **data accuracy** and **consistency**
* Avoids data loss or corruption
* Confirms data is stored and fetched **as expected**
* Detects mismatches between UI and DB (e.g., order total showing wrong in invoice)

**How Testers Do Database Testing**

1. Use **phpMyAdmin** in XAMPP or a **MySQL Client**
2. Run SQL queries to check:
   * If data inserted from forms is stored correctly
   * If updates and deletes reflect properly
3. Compare data in UI vs Database

DATABASE – SQL Commands (MySQL/XAMPP)

1. Create Table

Creates a new table named student with 3 columns:

2. Drop Table

Deletes the entire student table permanently:

3. Insert Data

Inserts a new student into the table:

4. Delete Record

Deletes the student whose sid is 3:

5. Select Data (All Records)

Displays all data from the student table:

6. Select Specific Record

Displays only the record where sid = 2:

7. Update Record

Updates the student record with sid = 4 to new values:

**7 Key Principles of Software Testing**

**🔹 1. Testing Shows the Presence of Defects**

**"Testing can show bugs exist, but not that they don’t."**

* The goal of testing is to **identify bugs**, not prove software is perfect.
* Even after 100 tests, we **can’t guarantee** the software is 100% bug-free.
* Testing **reduces defects** but doesn’t **remove all** of them.

*Example:* Even if your login system works 50 times, a bug might still occur in the 51st try.

**🔹 2. Exhaustive Testing is Not Possible**

**"You cannot test everything."**

* **Exhaustive testing** means checking with **every possible input**, which is **impossible** due to time, cost, and effort.
* Instead, we choose the **most important test cases** using techniques like **equivalence partitioning** or **boundary testing**.

*Example:* Testing every mobile number input format manually is not practical — only key cases are tested.

**🔹 3. Early Testing Saves Time & Cost**

**"Test early, fix cheaply."**

* Start testing at the **requirement stage**, not after development.
* **Static testing** (e.g., document reviews) can catch issues early.
* Fixing a bug early (design phase) is **cheaper** than after deployment.

*Analogy:* Like finding a wiring problem before painting a house — it's easier and cheaper to fix.

**🔹 4. Defect Clustering**

**"Most bugs are found in a few places."**

* Based on the **Pareto Principle (80/20 Rule)**:  
  **80% of bugs are in 20% of the code**.
* These clusters are usually in **critical modules** like login, payment, cart.

*Example:* A bug in the Flipkart **checkout page** affects the business more than a minor color mismatch.

**🔹 5. Pesticide Paradox**

**"Same tests = same results."**

* If you repeat the same test cases again and again, they **won’t find new bugs**.
* You must **review and update** test cases regularly to catch **new issues**.

*Analogy:* Just like pests become immune to one pesticide, your tests may become ineffective if not updated.

**🔹 6. Testing is Context-Dependent**

**"One size doesn't fit all."**

* The type of testing depends on the **type of software**:
  + Mobile App? → Battery & screen size testing.
  + E-commerce site? → Payment, performance, UX testing.
* There is no universal testing method.

*Example:* Testing an ATM system ≠ Testing a video game.

**🔹 7. Absence of Errors Fallacy**

**"Bug-free ≠ Useful."**

* Even if software is **99% error-free**, if it doesn’t meet **user requirements**, it’s **useless**.
* Testing should also verify whether the software does **what the user actually wants**.

*Example:* A flight booking app that is bug-free but doesn't allow you to choose a seat = failed product.

**Summary Table**

| **Principle** | **Key Idea** |
| --- | --- |
| 1. Presence of Defects | Testing shows bugs exist |
| 2. Exhaustive Testing | Testing everything is impossible |
| 3. Early Testing | Saves cost and time |
| 4. Defect Clustering | Bugs occur in specific areas |
| 5. Pesticide Paradox | Change test cases regularly |
| 6. Context Dependent | Different software = different tests |
| 7. Absence of Errors Fallacy | Bug-free ≠ correct product |

**Levels of Software Testing**

There are **4 main levels** of software testing:

| **Level** | **Description** |
| --- | --- |
| **1. Unit Testing** | Tests individual functions/modules. Performed by **developers**. |
| **2. Integration Testing** | Tests how modules work together (e.g., login → dashboard). |
| **3. System Testing** | Testing the entire system as a whole. Performed by **testers**. |
| **4. UAT (User Acceptance Testing)** | Final stage. Testing from **client’s point of view** to check if software meets requirements. |

**Project Testing Workflow**

1. **Client** gives requirements.
2. **Developer** writes the code and builds the website.
3. **Tester** creates test cases and starts testing.
4. **Bugs** found → sent back to **developer**.
5. Developer **fixes** bugs → Tester **retests**.
6. Once **bug-free, user-friendly**, and meets client expectations → UAT → **Go Live**.

**GUI Testing (Graphical User Interface)**

GUI testing is done to ensure the **look and feel**, alignment, buttons, font, and behavior of the application are correct.

**Includes:**

* Button placement & colors
* Font consistency
* Form alignment
* Input field validation
* Error/success message placement
* Icons & images

**Responsive Testing**

Checks whether the website layout **adjusts properly** on different devices.

**White Box, Black Box, Grey Box Testing**

| **Type** | **Who Performs?** | **Focus** | **Example** |
| --- | --- | --- | --- |
| **White Box** | Developer | **Internal Code Structure** | Login function code, logic flow |
| **Black Box** | Tester | **Functionality (without code knowledge)** | User login, cart, add product |
| **Grey Box** | Both | Mix of code logic and UI functionality | Check how login logic works behind frontend |

**Retesting vs Regression Testing**

| **Concept** | **Meaning** | **When?** |
| --- | --- | --- |
| **Retesting** | Testing a **specific bug** after it’s fixed | After developer solves a known bug |
| **Regression** | Testing the **overall application** to ensure the fix didn’t affect other parts | After every code change or update |

**Test Case Flow (Real-world Flow)**

1. Developer writes **code**.
2. Tester creates **test cases**.
3. Test is executed → **Bug found**.
4. Developer **fixes** the bug.
5. Tester does **retesting** of that fix.
6. Then does **regression testing** to check full app.
7. New features added (e.g., voice call) → fresh test cases made.

**Static vs Dynamic Testing**

| **Static Testing** | **Dynamic Testing** |
| --- | --- |
| No code execution | Involves executing code |
| Done on **documents** | Done on **running application** |
| Examples: SRS, test plan, test cases, bug reports | Examples: Login, Cart, Payment flow tests |

**Static Testing Activities**

1. **Review** –
   * Tester/team reviews test cases
   * Example: 1 Sr. tester reviews 200 test cases
2. **Walkthrough** –
   * Team member **explains** documents to others
3. **Inspection** –
   * Formal, detailed review with **entire team**
   * Used for **critical projects** (e.g., payment module)

**Critical Functionality Testing in Projects**

* **PM (Project Manager)** arranges review meetings on **WhatsApp/Zoom** or internal tools.
* For **50+ critical functionalities** (like payments), multiple rounds of review and test planning are done.

**Performance Testing Concepts**

Performance testing is done to check **how well** a software or website performs under various **load conditions**, focusing on speed, stability, scalability, and resource usage.

**🔹 Types of Performance Testing**

| **Type** | **Description** |
| --- | --- |
| **1. Load Testing** | Test how the system performs under **expected number of users** (e.g., 1k, 5k, 10k users). |
| **2. Stress Testing** | Test the **breaking point** by increasing load **beyond limits** (e.g., 1 lakh users) to check how the system fails and recovers. |
| **3. Volume Testing** | Test the system’s ability to handle **large volumes of data** (e.g., 50 GB database). |
| **4. Spike Testing** | Sudden **increase/decrease in user load** to test how the system responds. |