Software Testing

📘 Session 1 – Introduction to Software and Software Testing

🔹 Software Overview

A **software** is a collection of computer programs designed to perform specific tasks or functions.

Types of Software

1. **System Software**
    - Examples: Device drivers, Operating Systems, Servers, Utilities.
    - Purpose: Manages computer hardware and provides a platform for other software.
2. **Programming Software**
    - Examples: Compilers, Debuggers, Interpreters.
    - Purpose: Helps developers create, test, and debug applications.
3. **Application Software**
    - Examples: Web Applications, Mobile Apps, Desktop Applications.
    - Purpose: Directly serves end-users by performing specific tasks or services.

🔹 Software Testing

Definition

- A process that is part of the software development lifecycle.
- Activity to **detect and identify defects** in the software.
- Objective: To release a **quality product** to the client.

Software Quality

- **Quality** = Justification of all the requirements of a customer in a product.
- Quality is defined **in the customer's mind**, not just in the product.

Characteristics of Quality Software:

- Bug-free
- Delivered on time
- Within budget
- Meets requirements and/or expectations
- Maintainable

🔹 **Why Do We Need Software Testing?**

- To ensure that software is **bug free**.
- To confirm that the system meets **customer requirements** and specifications.
- To confirm that the system meets **end-user expectations**.
- Because fixing bugs **after release** is **more expensive** than during development/testing.

🔹 **Project vs. Product**

- **Project** → Developed for a **specific customer** based on their requirements.
- **Product** → Developed for **multiple customers** based on market requirements.

🔹 **Error, Bug/Defect, and Failure**

- **Error** → Human action that produces an incorrect result (e.g., coding mistake).
- **Bug/Defect** → Deviation between expected and actual behavior of the system (found during testing).
- **Failure** → Defect encountered by the **end-user** while using the system.

🔹 **Reasons for Software Bugs**

- Miscommunication or no communication.
- Software complexity.
- Programming errors.
- Frequently changing requirements.
- Lack of skilled testers.

🎯 **Interview Q&A**

Q1. What are the different types of software?
👉 System software, programming software, and application software.

Q2. What is software testing and why is it needed?
👉 It's the process of identifying defects to ensure software meets requirements, expectations, and quality standards. It reduces the cost of fixing bugs later.

Q3. Define Quality in software.
👉 Quality means delivering a bug-free, reliable, maintainable, and customer-satisfying product within budget and time.

Q4. Difference between Project and Product?

- **Project** → Developed for a single customer (custom solution).

- **Product** → Developed for many users (market-driven).

Q5. What is the difference between Error, Defect, and Failure?

- **Error** → Human mistake in coding or design.
- **Defect/Bug** → Mismatch between expected and actual result found by testers.
- **Failure** → When the end-user encounters a defect in real usage.

Q6. What are common reasons for software bugs?

👉 Miscommunication, complexity, coding errors, changing requirements, lack of skilled testers.

📘 Session 2

SDLC (Software Development Life Cycle)

A process used by the software industry to **design, develop, and test software**.

Waterfall Model

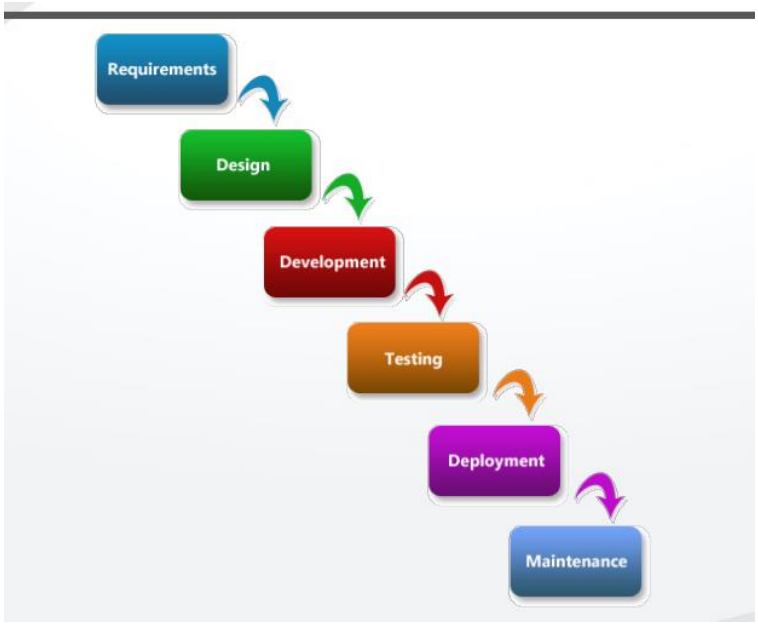
Advantages

- Quality of the product will be good.
- Since requirement changes are not allowed, chances of finding bugs will be less.
- Initial investment is less since testers are hired at later stages.

- Preferred for small projects where requirements are frozen.

Disadvantages

- Requirement changes are not allowed.
- If there is a defect in requirement, it will continue in later phases.
- Total investment is more because rework is time-consuming.
- Testing will start only after coding.



Spiral Model

- Iterative model.
- Overcomes drawbacks of Waterfall model.
- Used when there is dependency on modules.
- Each cycle releases new software to customer.
- Software is released in multiple versions → also called **Version Control Model**.

Advantages

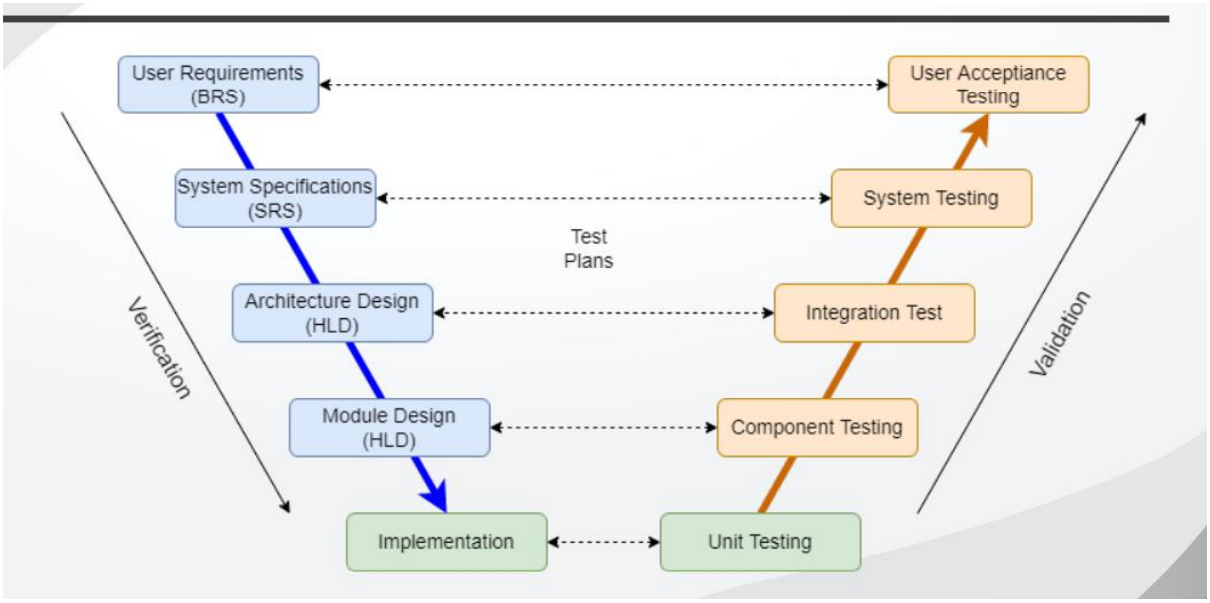
- Testing is done in every cycle before going to the next cycle.
- Customer gets to use the software for every module.
- Requirement changes are allowed after every cycle.

Disadvantages

- Requirement changes are NOT allowed in between the cycle.
- Every cycle of Spiral looks like Waterfall model.

- No testing in requirement & design phase.

V-Model



Advantages

- Testing is involved in every phase.

Disadvantages

- Documentation is more.

- Initial investment is more.

Verification vs Validation

- **Verification** → Are we building the right product?
    - Focus on documentation.
    - Techniques: Reviews, Walkthroughs, Inspections.
- **Validation** → Are we building the product right?
    - Focus on software.
    - Techniques: Actual testing (Unit, Integration, System, UAT).

Static vs Dynamic Testing

- **Static Testing** → Testing project documents (no code execution).
    - Examples: Reviews, Walkthroughs, Inspections.
- **Dynamic Testing** → Testing actual software by giving inputs and checking results.

Static Testing Techniques

- **Reviews** → Ensure correctness & completeness of documents.
    - Examples: Requirement reviews, design reviews, code reviews, test plan reviews.
- **Walkthroughs** → Informal review, no documentation, conducted as needed.
- **Inspections** → Most formal review.
    - 3–8 participants (reader, writer, moderator).
    - Proper schedule communicated to team.

Process:
The author presents the document or code to peers.
Participants ask questions and suggest improvements.
No formal roles or preparation required.

Dynamic Testing Techniques

- **Unit Testing** → Testing individual components in isolation.
- **Integration Testing** → Testing interaction between modules.
- **System Testing** → Testing complete system against requirements.
- **UAT (User Acceptance Testing)** → Performed by real users to check if software meets needs.

Interview Q&A

Q1. What is SDLC?

What is UAT?
User Acceptance Testing is where end users or clients test the software to ensure it meets their business requirements and is ready for deployment.

Key Features of UAT:
Performed by: Actual users or stakeholders, not developers or testers.
Goal: Validate that the system works as expected in real-world scenarios.
Focus: Business functionality, usability, and compliance with requirements.
Environment: Typically done in a staging environment that mimics production.

Ans: SDLC (Software Development Life Cycle) is a structured process used to design, develop, test, and deliver software with high quality.

Q2. Explain Waterfall Model and its drawbacks.

Ans:

- Waterfall is a sequential model where each phase must be completed before the next begins.
- Drawbacks: Requirement changes not allowed, defects in requirements carry forward, testing starts late, high rework cost.

Q3. What is the Spiral Model and why is it called Version Control Model?

Ans:

- Spiral is an iterative model with multiple cycles, and each cycle delivers a working software version to the customer.
- Since multiple versions are released, it is called a Version Control Model.

Q4. What is the V-Model?

Ans:

- V-Model is a verification and validation model where testing is done in parallel with each development phase.
- Advantage: Testing in every phase.
- Disadvantage: High documentation and investment.

Q5. Difference between Verification and Validation.

Ans:

- **Verification**: Process of checking documents, ensures “Are we building the right product?” (reviews, inspections).
- **Validation**: Process of actual testing, ensures “Are we building the product right?” (unit, integration, system, UAT).

Q6. What is the difference between Static and Dynamic Testing?

Ans:

- **Static Testing**: Testing documents without executing code (reviews, walkthroughs, inspections).

- **Dynamic Testing**: Testing actual software by executing it with inputs and verifying outputs.

Q7. Give examples of Static Testing Techniques.

Ans: Reviews, Walkthroughs, Inspections.

Q8. Give examples of Dynamic Testing Techniques.

Ans: Unit testing, Integration testing, System testing, UAT testing.

📘 Session 3

QA, QC, Testing Methodologies, and Levels of Testing

QA Vs QC

- QA is Process related.
- QC is the actual testing of the software.
- QA focuses on building in quality.
- QC focuses on testing for quality.
- QA is preventing defects.
- QC is detecting defects.
- QA is process oriented.
- QC is Product oriented.
- QA for entire life cycle.
- QC for testing part in SDLC

Example: Implementing coding standards and conducting process reviews to avoid bugs before coding begins.
Example: Running test cases to find and fix software bugs before deployment.

QE stands for Quality Engineering, a modern approach that blends Quality Assurance (QA) and Quality Control (QC) with engineering practices to ensure software quality throughout the development lifecycle.

Testing Methodologies

- White Box Testing
- Black Box Testing
- Grey Box Testing

White Box Testing (Structural Testing)
Definition: Testing based on internal logic and structure of the code.
Who performs it: Developers or testers with programming knowledge.
Focus: Code paths, conditions, loops, and branches.
Example:
Testing a login function by checking if the if condition correctly validates username and password.

White Box Testing

- White Box Testing conducts on internal logic of the programs.
- Programming Skills are required.

- Example: Unit Testing & Integration Testing

Black Box Testing

- Testing conducts on functionality of the application whether it is working according to customer requirements or not.
- Example: System Testing & UAT Testing

Grey Box Testing

- Both combination of white box and black box testing.
- Example: Database Testing

Levels Of Testing

1. Unit Testing
2. Integration Testing
3. System Testing
4. UAT Testing

Unit Testing

- A unit is a single component or module of a software.
- Unit testing conducts on a single program or single module.
- Unit Testing is white box testing technique.
- Unit testing is conducted by the developers.
- Unit testing techniques:
    - Basis path testing
    - Control structure testing
    - Conditional coverage
    - Loops Coverage
    - Mutation Testing

Integration Testing

- Integration testing performed between 2 or more modules.

Black Box Testing (Behavioral Testing)
Definition: Testing based on external behavior without knowing internal code.
Who performs it: QA testers or end users.
Focus: Inputs and expected outputs.
Example:
Entering valid and invalid credentials on a login page to see if access is granted or denied.

Grey Box Testing (Hybrid Testing)
Definition: Combines both internal knowledge and external testing.
Who performs it: Testers with partial knowledge of the system.
Focus: Integration points, data flow, and security.
Example:
Testing a web application where the tester knows the database schema and checks how form inputs affect backend data.

- Integration testing focuses on checking data communication between multiple modules.
- Integration Testing is white box testing technique.

Integration Testing Types

- Incremental Integration Testing
- Non-Incremental Testing

Incremental Integration

- Incremental Integration: Incrementally adding the modules and testing the data flow between the modules.
- There are 3 Approaches:
    - Top Down
    - Bottom Up
    - Hybrid (Sandwich)

Incremental Integration (Top-Down Integration)

- Incrementally adding the modules and testing the data flow between the modules.
- Ensure the module added is the child of previous module.
- Takes help of stubs for testing.

Incremental Integration (Bottom-Up Integration)

- Incrementally adding the modules and testing the data flow between the modules.
- Ensure the module added is the parent of the previous module.

Incremental Integration (Sandwich/Hybrid Approach)

- Combination of Top-Down & Bottom-Up approach is called Sandwich Approach.

Non-Incremental Integration Testing / Big Bang Testing

- Adding all the modules in a single shot and test the data flow between modules.
- Drawbacks:
    - We might miss data flow between some of the modules.
    - If you find any defect we can’t understand the root cause of defect.

Integration Testing – Stubs & Drivers

- Stubs and Drivers are the dummy programs in Integration testing used to facilitate the software testing activity.
- These programs act as substitutes for the missing modules in the testing.
- They do not implement the entire programming logic of the software module but simulate data communication with the calling module while testing.
- **Stub:** Is called by the Module under Test.
- **Driver:** Calls the Module to be tested.

System Testing

- Testing over all functionality of the application with respective client requirements.
- It is a black box testing technique.
- Conducted by testing team.
- After completion of component and integration level testing, we start System testing.
- Before conducting system testing we should know the customer requirements.
- System Testing focuses on:
    - User Interface Testing (GUI)
    - Functional Testing
    - Non-Functional Testing
    - Usability Testing

Alpha Testing
Purpose: To catch bugs and usability issues before releasing the software to external users.
Key Features:
Performed by: Internal teams—developers, QA testers, or product managers.
Environment: Controlled lab-like setting, not real-world usage.
Timing: Happens before beta testing, after system testing.
Focus: Stability, core functionality, and major issues
Techniques Used: Both white-box and black-box testing.

User Acceptance Testing (UAT)

- After completion of system testing, UAT team conducts acceptance testing in two levels:
    - Alpha testing
    - Beta testing

Beta Testing
Purpose: To validate the software in real-world conditions and gather user feedback.
Key Features:
Performed by: External users—customers, industry experts, or public testers.
Environment: Real-world usage scenarios.
Timing: Happens after alpha testing, just before official release.
Focus: Minor bugs, user experience, compatibility, and performance.
Techniques Used: Mostly black-box testing.

QA vs QC

Q1. What is the difference between QA and QC?
Answer:

Feature	QA	QC
Focus	Process-oriented	Product-oriented

Feature	QA	QC
Purpose	Prevent defects	Detect defects
Lifecycle	Entire SDLC	Testing part of SDLC
Approach	Building in quality	Testing for quality

Testing Methodologies

Q2. What are the main testing methodologies?
Answer:

1. **White Box Testing** – Testing internal logic and code, requires programming knowledge. Example: Unit Testing, Integration Testing.
2. **Black Box Testing** – Testing functionality without knowing the code. Example: System Testing, UAT.
3. **Grey Box Testing** – Combination of white box and black box. Example: Database Testing.

Levels of Testing

Q3. What are the levels of software testing?
Answer:

1. **Unit Testing** – Test individual modules. White box. Conducted by developers.
2. **Integration Testing** – Test interaction between modules. White box.
3. **System Testing** – Test complete system against requirements. Black box.
4. **UAT** – User Acceptance Testing. Conducted by end users.

Unit Testing

Q4. Who performs unit testing and what techniques are used?
Answer:

- Performed by developers.
- Techniques: Basis Path Testing, Control Structure Testing, Conditional Coverage, Loops Coverage, Mutation Testing.

Integration Testing

Q5. What is integration testing and what are its types?
Answer:

- Testing between multiple modules to verify data flow and interactions.

- Types:
    1. Incremental Integration: Top-Down, Bottom-Up, Sandwich/Hybrid
    2. Non-Incremental / Big Bang Testing

Q6. What is the difference between Top-Down and Bottom-Up integration?
Answer:

- **Top-Down:** Add child modules incrementally, parent modules tested after. Uses stubs.
- **Bottom-Up:** Add parent modules incrementally, child modules tested first. Uses drivers.

Q7. What is Big Bang integration testing?
Answer:

- All modules are integrated at once and tested together.
- Drawbacks: Hard to identify defect source and may miss module data flows.

Q8. What are Stubs and Drivers?
Answer:

- **Stub:** Dummy module called by module under test (used in Top-Down).
- **Driver:** Dummy module calling the module under test (used in Bottom-Up).

System Testing

Q9. What is system testing and what does it cover?
Answer:

- Black box testing of complete software to ensure it meets requirements.
- Covers:
    - GUI Testing
    - Functional Testing
    - Non-Functional Testing
    - Usability Testing

User Acceptance Testing (UAT)

Q10. What is UAT and its types?
Answer:

- Performed by end users to validate software meets business requirements.
- Types:
    1. Alpha Testing – Performed in-house by client or users.
    2. Beta Testing – Performed by actual end users in real environment.

📘 Session 4

System Testing Types

System Testing

- GUI Testing
- Usability Testing
- Functional Testing
- Non-Functional Testing

System Testing Types Explained

1. **GUI Testing:**
    - **What It Checks:** Makes sure all buttons, icons, and screens in the software look and work as they should.
    - **Example:** Clicking buttons, checking if the layout is correct.
2. **Usability Testing:**
    - **What It Checks:** Tests if the software is easy for people to use and understand.
    - **Example:** Checking if the menus are clear, and tasks are straightforward.
3. **Functional Testing:**
    - **What It Checks:** Verifies that each part of the software performs its job correctly.
    - **Example:** Testing if login, search, and other functions work as intended.
4. **Non-Functional Testing:**
    - **What It Checks:** Looks at aspects beyond specific functions, like performance, security, and how easy it is to recover from errors.
    - **Example:** Checking how many users the system can handle at once (performance testing), or testing how secure the system is against unauthorized access (security testing).

GUI Testing

GUI testing, or Graphical User Interface testing, is a type of software testing that focuses on verifying the functionality and usability of the graphical elements of a software application. It involves testing the user interface components such as buttons, menus, icons, and any other visual elements to ensure they work as intended and provide a positive user experience.

GUI Testing Types:

- System Testing
- GUI Testing
- Usability Testing
- Functional Testing
- Non-Functional Testing

GUI Testing Checklist

- Testing the size, position, width, height of the elements.
- Testing of the error messages that are getting displayed.
- Testing the different sections of the screen.
- Testing of the font whether it is readable or not.
- Testing of the screen in different resolutions with the help of zooming in and zooming out.
- Testing the alignment of the texts and other elements like icons, buttons, etc., are in proper place or not.
- Testing the colours of the fonts.
- Testing whether the image has good clarity or not.
- Testing the alignment of the images.

Additional GUI Testing Checklist:

- Testing of the spelling.
- The user must not get frustrated while using the system interface.
- Testing whether the interface is attractive or not.
- Testing of the scrollbars according to the size of the page if any.
- Testing of the disabled fields if any.
- Testing of the size of the images.
- Testing of the headings whether it is properly aligned or not.
- Testing of the colour of the hyperlink.
- Testing UI Elements like button, textbox, text area, check box, radio buttons, drop downs, links etc.

Usability Testing

- During this testing, it validates if the application provides context-sensitive help to the user.
- Checks how easily the end users are able to understand and operate the application.

Types of Testing:

- System Testing
- GUI Testing
- Usability Testing
- Functional Testing
- Non-Functional Testing

Functional Testing

- Functionality is the behavior of the application.
- Functional testing talks about how your feature should work.

Functional Testing Types:

1. Object Properties Testing
2. Database Testing
3. Error Handling Testing
4. Calculations/Manipulations Testing
5. Links Testing
6. Cookies & Sessions Testing

Object Properties Testing

- Ensures that visual elements (like buttons, text boxes) in the application have the right characteristics.
- Checks if these objects behave as expected based on their defined properties.
- **Example:** Verifying that a "Submit" button is visible, clickable, and has the correct color according to the application's design.

Database Testing

- Ensures that the application interacts correctly with its database.
- Checks if data is stored, retrieved, and manipulated accurately.
- **Example:** Confirming that user details entered through a registration form are correctly saved and can be retrieved when logging in.

Error Handling Testing

- Focuses on functional aspects related to how the application deals with errors.
- Checks if the system shows appropriate error messages and handles unexpected situations gracefully without crashing.
- **Example:** Testing system response when a user tries to submit a form with missing information, ensuring it displays a helpful error message.

Calculations/Manipulations Testing

- Assesses the accuracy of numeric operations and data manipulations performed by the application.
- **Example:** Testing an e-commerce application to ensure that the total price of items in a shopping cart is accurately calculated.

Links Testing

- Focuses on the existence and execution of links.
- Verifies that hyperlinks within the application work as expected.
- Checks if users are directed to the correct pages for smooth navigation.
- **Example:** Clicking menu links and confirming each link leads to the intended page.
- **Types of links:** Internal Links, External Links, Broken Links

---

Cookies & Sessions Testing

- Evaluates how well the application manages user-specific information.
- Checks if session-related functionalities like login persistence work as intended.
- **Example:** Logging into an online account, closing the browser, reopening it, and confirming that the user remains logged in due to correct handling of cookies or session data.

---

1. What is System Testing?

Answer:
System Testing is a type of testing where the complete and integrated software is tested as a whole to verify that it meets specified requirements. It ensures the application works correctly and meets customer expectations.

---

2. What are the types of System Testing?

Answer:

1. GUI Testing (Graphical User Interface Testing)

2. Usability Testing
3. Functional Testing
4. Non-Functional Testing

---

3. What is GUI Testing?

Answer:
GUI Testing focuses on testing the visual elements of an application like buttons, menus, icons, images, links, and forms. It ensures that the interface is user-friendly, elements are aligned, readable, and functioning as intended.

---

4. What do you check in GUI Testing?

Answer:

- Alignment, size, position of elements
- Font readability and colors
- Screen resolution and zoom compatibility
- Spelling, error messages, disabled fields
- Scrollbars, hyperlinks, and images
- UI elements like buttons, checkboxes, radio buttons, text boxes

---

5. What is Usability Testing?

Answer:
Usability Testing checks how easy and user-friendly the software is. It ensures end users can understand and operate the application without frustration and the interface is attractive and simple to use.

---

6. What is Functional Testing?

Answer:
Functional Testing verifies that each feature of the software works as per the requirements. It focuses on the behavior of the system and validates that outputs are correct for given inputs.

---

7. Types of Functional Testing?

Answer:

1. Object Properties Testing
2. Database Testing

3. Error Handling Testing
4. Calculations/Manipulations Testing
5. Links Testing
6. Cookies & Sessions Testing

---

8. Explain Object Properties Testing.

Answer:
It ensures that GUI elements like buttons, textboxes, and links behave as expected according to their properties such as visibility, clickability, and color.
Example: A "Submit" button should be visible, clickable, and have correct color.

---

9. Explain Database Testing.

Answer:
Database Testing ensures that data is correctly stored, retrieved, and manipulated in the database.
Example: User details entered in a registration form are correctly saved and can be retrieved during login.

---

10. Explain Error Handling Testing.

Answer:
It validates how the system responds to incorrect or unexpected inputs. Checks if appropriate error messages are shown and the system does not crash.
Example: Submitting a form with missing fields displays an error message.

---

11. Explain Calculations/Manipulations Testing.

Answer:
It ensures that all mathematical operations or data manipulations in the application are correct.
Example: Total price in a shopping cart is accurately calculated in an e-commerce website.

---

12. Explain Links Testing.

Answer:
It verifies that hyperlinks work correctly and direct users to the intended pages.
Example: Clicking a menu link redirects to the correct section of the website.
Types of Links: Internal, External, Broken

---

13. Explain Cookies & Sessions Testing.

Answer:
Tests whether the application correctly manages user-specific data and session information.
Example: Logging in, closing the browser, reopening it, and confirming the user remains logged in due to correct handling of cookies.

---

14. What is Non-Functional Testing?

Answer:
Non-Functional Testing evaluates aspects of software not related to specific functions, like performance, security, scalability, and recoverability.
Example: Performance testing checks how many users the system can handle simultaneously.

---

15. Difference between Functional and Non-Functional Testing?

Functional Testing	Non-Functional Testing
Tests specific functions/features	Tests overall performance and quality aspects
Example: Login, search, payment	Example: Load testing, security testing
Checks "what the system does"	Checks "how the system behaves"

---

16. Difference between GUI and Usability Testing?

GUI Testing	Usability Testing
Focuses on layout, alignment, font, buttons, icons	Focuses on ease of use and user experience
Checks correctness of UI elements	Checks if user can interact with the system easily
Example: Button clickable and visible	Example: Task is easy to complete without confusion

📘 Session 5

Non-functional Testing

1. **Performance Testing**
    - Load Testing
    - Stress Testing
    - Endurance Testing
    - Spike Testing
    - Volume Testing

2. **Security Testing**
3. **Recovery Testing**
4. **Compatibility Testing**
5. **Configuration Testing**
6. **Installation Testing**
7. **Sanitation/Garbage Testing**

System Testing

- GUI Testing
- Usability Testing

Types:

- Functional Testing
- Non-Functional Testing

Load Testing

Load testing is a type of performance testing which involves evaluating the performance of the system under the expected workload.

A typical load test includes determining the response time, throughput, error rate, etc., during the course of the load test.

Example:
For a newly developed application with an anticipated load of around 1000 concurrent users, we will create a load test script and configure it with 1000 virtual users and run it for say 1-hour duration. After the load test completion, we can analyse the test result to determine how the application will behave at the expected peak load.

Stress Testing

Stress testing is a type of performance testing where we evaluate the application's performance at a load much higher than the expected load.

Another aspect of stress testing is to determine the break-point of the application, the point at which the application fails to respond in the correct manner.

Example:
For an application with an anticipated load of 1000 users, we will run the test with 1200 users and check if the application is robust enough to not crash.

Endurance Testing (or Soak Testing)

Endurance testing is also known as Soak Testing.
It is done to determine if the system can sustain the continuous expected load for a long duration.

Issues like memory leakage are found with endurance testing.

Example:
For an application like Income Tax filing, the application is used continuously for a very long duration by different users. In this type of application, memory management is very critical. For an application like these, we can run the test for 24 hours to 2 days duration and monitor the memory utilization during the whole test execution.

Spike Testing

In spike testing, we analyse the behaviour of the system on suddenly increasing the number of users.

It also involves checking if the application is able to recover after the sudden burst of users.

Example:
For an e-commerce application running an advertisement campaign, the number of users can increase suddenly in a very short duration. Spike testing is done to analyse these types of scenarios.

Volume Testing

The volume testing is performed by feeding the application with a high volume of data.

The application can be tested with a large amount of data inserted in the database or by providing a large file to the application for processing.

Using volume testing, we can identify the bottleneck in the application with a high volume of data.

Example:
For a newly developed e-commerce application, we can perform volume testing by inserting millions of rows in the database and then carry out the performance test execution.

Performance Testing Types - Summary

Test Type	Goal	Method
Load Testing	Understand performance under expected load	Gradual load increase – Expected Load
Stress Testing	Identify breaking points	Extreme/unrealistic load – Beyond Expected Load
Endurance Testing	Assess long-term stability	Sustained load over time – Expected Load for long time

Test Type	Goal	Method
Spike Testing	Evaluate response to sudden traffic bursts	Short-duration load spikes – Sudden change in load
Volume Testing	Handle large data volumes	Large datasets/data transfer – High volumes of data

---

Security Testing

- Security Testing checks if the software is secure and protects sensitive information.
- It aims to find and fix vulnerabilities that could be exploited by hackers.

Focus on:

- User Authentication
- User Authorization / Access Control
- Data Encryption & Decryption
- Network Security
- System Software Security
- Client-side Application Security
- Server-side Application Security

Example:
Testing an online banking application to make sure that user account information is protected from unauthorized access.

Recovery Testing

Recovery Testing assesses how well a system can recover after a failure or crash. It checks if the software can resume normal operation without losing data.

Software should be recovery tested for failures like:

- Power supply failure
- External server unreachable
- Wireless network signal loss
- Physical conditions
- Database server down
- API's response failed

Example:
Simulating a sudden power outage and checking if a word processing software can recover the document being edited when power is restored.

Compatibility Testing

Compatibility Testing ensures that the software works well on different devices, browsers, and operating systems. It verifies that the application is compatible with a variety of environments.

Focus on:

- Operating System compatibility
- Browser compatibility (Cross browser testing)
- Hardware compatibility (Configuration testing)
- Backward compatibility
- Forward compatibility

Example:
Testing a mobile app on different smartphones and ensuring it functions correctly on various screen sizes and resolutions.

Configuration Testing

Configuration Testing checks if the software works correctly with different configurations or settings. It ensures that the application adapts well to various setups.

Example:
Testing a video game on different computers with varied hardware configurations to ensure smooth gameplay on various setups.

Installation Testing

Installation Testing assesses the process of installing and uninstalling the software. It checks if the installation is smooth and if the software can be removed without causing issues.

Example:
Installing a new software version and checking if it sets up properly without errors or conflicts with existing installations.

Sanitation/Garbage Testing

Sanitation/Garbage Testing involves checking for unnecessary or leftover data in the system. It ensures that the software cleans up after itself and doesn't leave unused or "garbage" data behind.

Example:
Testing a messaging app to make sure that deleted messages are completely removed from the system and don't linger in the background.

Functional Testing vs Non-Functional Testing

Functional Testing:

- Focus: Ensures that the software functions as expected, performing its intended tasks.
- What It Checks: Specific features, actions, and behaviors outlined in the requirements.

Non-Functional Testing:

- Focus: Evaluates how well the software performs under various conditions and assesses aspects beyond specific functionalities.
- What It Checks: Performance, security, usability, and other aspects related to the user experience.

Major Difference:

- Functional Testing is about **WHAT** the software does (ensuring individual features work correctly).
- Non-Functional Testing is about **HOW WELL** the software performs (speed, security, user experience, etc.).

✅ **Interview Q&A – Non-Functional Testing (Session 5)**

Q1. What is Non-Functional Testing?

Answer:
Non-Functional Testing verifies **how well** the software works rather than what it does.
It checks aspects like performance, security, compatibility, usability, recovery, and installation.

Q2. What is the difference between Functional and Non-Functional Testing?

Answer:

- **Functional Testing** → Focuses on *WHAT* the system does (features, requirements).
- **Non-Functional Testing** → Focuses on *HOW WELL* the system performs (speed, security, user experience).

Q3. What are the different types of Non-Functional Testing?

Answer:

1. Performance Testing
    - Load Testing

    - Stress Testing
    - Endurance/Soak Testing
    - Spike Testing
    - Volume Testing
2. Security Testing
3. Recovery Testing
4. Compatibility Testing
5. Configuration Testing
6. Installation Testing
7. Sanitation/Garbage Testing

Q4. What is Load Testing? Can you give an example?

Answer:

- Load testing checks system performance under the expected workload.
- It measures response time, throughput, error rate, etc.

Example: Running an e-commerce site with 1000 virtual users for 1 hour to ensure it handles the expected load.

Q5. What is Stress Testing?

Answer:

- Stress Testing checks application behavior under **beyond expected load** conditions.
- It helps find the **break-point** where the system fails.

Example: Testing an app designed for 1000 users with 1200+ users.

Q6. What is Endurance/Soak Testing?

Answer:

- Endurance (Soak) testing checks whether the system can handle **continuous load for a long period**.
- Helps find issues like memory leaks.

Example: Running a tax-filing app continuously for 24–48 hours.

Q7. What is Spike Testing?

Answer:

- Spike Testing checks system behavior when the number of users **suddenly increases**.
- Also checks recovery after the sudden spike.

Example: An e-commerce app during a flash sale or ad campaign.

Q8. What is Volume Testing?

Answer:

- Volume testing checks application performance with **large amounts of data**.

Example: Inserting millions of rows into a database and then running performance tests.

Q9. Can you summarize Performance Testing types in one line?

Answer:

- **Load Testing** → Performance under expected load.
- **Stress Testing** → Breaking point under extreme load.
- **Endurance Testing** → Stability under continuous load.
- **Spike Testing** → Sudden load increase.
- **Volume Testing** → Performance with huge data volumes.

Q10. What is Security Testing?

Answer:

- Security Testing checks if the application is secure against threats.
- Focuses on authentication, authorization, encryption, network, client-side, and server-side security.

Example: Testing an online banking app to ensure user data is safe from unauthorized access.

Q11. What is Recovery Testing?

Answer:

- Recovery Testing ensures the system can recover after failures like power outage, server crash, or network loss.

Example: Simulating a power failure and verifying a document editor restores the file after restart.

Q12. What is Compatibility Testing?

Answer:

- Ensures software works across **different devices, OS, browsers, and hardware**.

Example: Testing a mobile app on Android and iOS with different screen resolutions.

Q13. What is Configuration Testing?

Answer:

- Checks software performance with different hardware or software configurations.

Example: Testing a game on different computers with varied CPU, RAM, and GPU.

Q14. What is Installation Testing?

Answer:

- Ensures that the software installs and uninstalls properly without errors.

Example: Installing a new version of an app and checking if it runs without issues.

Q15. What is Sanitation/Garbage Testing?

Answer:

- Ensures software does not leave unnecessary data after operations.

Example: Checking that deleted chat messages in a messaging app are completely removed.

📘 Session 6

Software Testing Terminology

Regression Testing

Regression Testing
Regression testing is the process of testing a software application to ensure that new changes (like updates, bug fixes, or new features) do not negatively impact the existing functionality of the software.

Example:
Imagine you have a mobile app that allows users to log in, view their profile, and send messages. If a developer adds a new feature, such as the ability to upload profile pictures, regression testing ensures that after this change, users can still log in, view their profiles, and send messages without any issues.

Types of Regression Testing

- **Unit Regression Testing:**
  We test only the specific changes made by the developer.
- **Regional Regression Testing:**
  We test the part that was changed along with the connected parts.
  Impact Analysis Meeting will be conducted to figure out which parts will be affected by the changes, involving both the testing team and the developers.
- **Full Regression:**
  We test the main part that was changed and also check the rest of the software to be thorough.
  For example, if the developer made changes in many areas, instead of checking each one separately, we test everything together in one full round.

Re-Testing

Whenever the developer fixed a bug, tester will test the bug fix is called Re-testing.

- Tester close the bug if it worked otherwise re-open and send to developer.
- To ensure that the defects which were found and posted in the earlier build were fixed or not in the current build.

Example:
Build 1.0 was released. Test team found some defects (Defect Id 1.0.1, 1.0.2) and posted.
Build 1.1 was released, now testing the defects 1.0.1 and 1.0.2 in this build is retesting.

Re-Testing Vs Regression Testing

- An Application Under Test has three modules namely Admin, Purchase and Finance.
- Finance module depends on Purchase module.
- If a tester found a bug on Purchase module and posted. Once the bug is fixed, the tester needs to do Retesting to verify whether the bug related to the Purchase is fixed or not and also tester needs to do Regression Testing to test the Finance module which depends on the Purchase module.

In summary:
Re-testing is specifically testing the fixed bug to confirm it's resolved, while Regression Testing is making sure that the fix hasn't caused new issues or disruptions, especially in modules that are interconnected.

Smoke Testing

Smoke Testing

- Also called BVT(Build Verification Test).
- Focus on stability of the build and we check whether it is ready for further testing or not.
- During smoke test we check build installation, appearance of basic screens and navigations etc.
- The term "smoke" comes from the idea that if there's a major issue, it would generate enough smoke to stop further testing.

Examples:

- Application is able to install or not
- Once installed, all the screens are working properly or not
- Backend database is up and running or not.
- All the API's related to application are invoked or not etc.

Sanity Testing

Sanity Testing

- Also called Basic Functional Test (BFT).
- Focus on basic functionality and we check whether basic functionality is working or not.
- During sanity test we check basic functionalities like login, user registration and logout etc.

Examples:

- Sign up option is available on login page.
- Clicking "Sign up" redirects to proper , sign up form.
- Clicking Sign in does not re-direct to "Sign up" form.
- Submitting "Sign up" form goes successful, with out crash.
- User signed up, is able to login

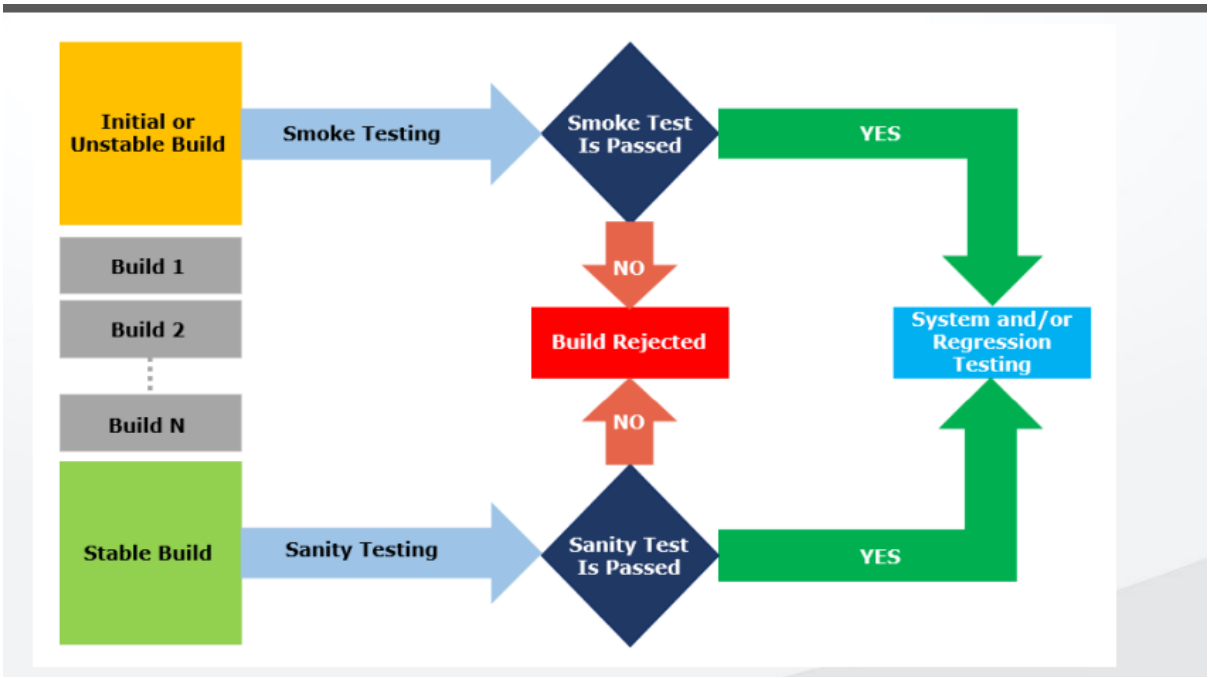
Sanity Vs Smoke Testing

Smoke Testing

- Smoke Test is done to make sure the build we received from the development team is testable/stable or not.
- Smoke Testing is performed by both developers and testers.
- Smoke Testing, build may be either stable or unstable
- It is done on initial builds.
- Usually it is done every time there is a new build release.

Sanity Testing

- Sanity Test is done to check for the main functionalities of the application without going deeper.
- Sanity Testing is performed by testers alone.
- Sanity Testing, build is relatively stable
- It is done on stable builds.
- It is planned when there is no enough time to do in-depth testing.



Exploratory Testing

- We have to explore the application ,understand completely and test it.
- Understand the application , identify all possible scenarios , document it then use it for testing.
- We do exploratory testing when the Application ready but there is no requirement.
- Test Engineer will do exploratory testing when there is no requirement.

Drawbacks:

- You might misunderstand any feature as a bug (or) any bug as a feature since you do not have requirement.
- Time consuming
- If there is any bug in application , you will never know about it.

Adhoc Testing

- Testing application randomly without any test cases or any business requirement document.
- Adhoc testing is an informal testing type with an aim to break the system.
- Tester should have knowledge of application even thou he doesn't have requirements/test cases.
- This testing is usually an unplanned activity.

Monkey Testing

- Testing application randomly without any test cases or any business requirement document.
- Monkey testing is an informal testing type with an aim to break the system.
- Tester do not have knowledge of application
- Suitable for gaming applications.

Exploratory Vs Adhoc Vs Monkey Testing

Testing Type	Documentation	Plan	Testing Style	Tester's Knowledge	Testing Approach	Purpose	Application Type
Exploratory Testing	None	No formal plan	Informal	Testers don't know much about the application	Random testing	Intention is to learn or explore the functionality of the application	Any application that is new to the tester
Adhoc Testing	None	No formal plan	Informal	Testers should have some knowledge of the application's functionality	Random testing	Intention is to break the application or find out corner defects	Any application
Monkey Testing	None	No formal plan	Informal	Testers don't know much about the application	Random testing	Intention is to break the application or find out corner defects	Typically used for gaming applications

---

Positive Testing

- Positive testing focuses on ensuring that a system behaves as expected when provided with valid inputs.
- The goal is to confirm that the software functions correctly under normal or expected conditions. In positive testing, the tester checks if the application does what it is supposed to do when everything is right.

Examples of Positive Tests:

- Login Test: Entering valid credentials and checking if the user can successfully log in.
- Calculator Addition: Adding two positive numbers to ensure the calculator gives the correct sum.
- Form Submission: Filling out a form with valid data and verifying that it is successfully submitted.

Negative Testing

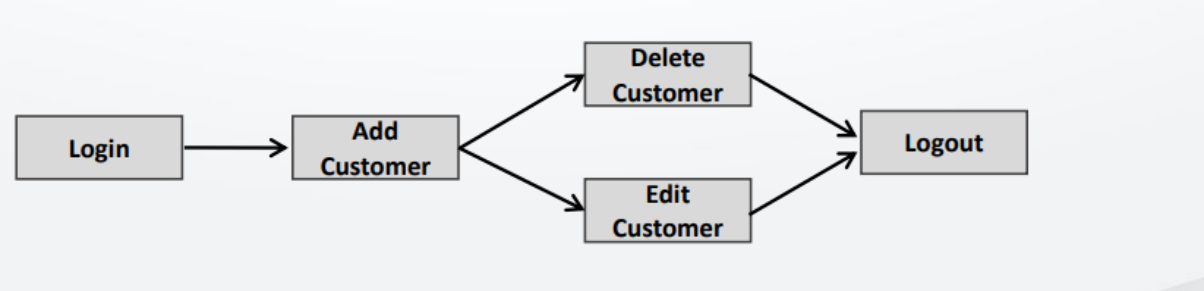
- Negative testing is about examining how well a system handles invalid or unexpected inputs.
- The goal is to identify potential weaknesses or vulnerabilities in the software by deliberately providing it with incorrect or inappropriate data.

Examples of Negative Tests:

- Login Test (Negative): Attempting to log in with an incorrect password to check if the system rejects invalid credentials.
- File Upload (Negative): Trying to upload a file in a format not supported by the application and verifying if the system handles it gracefully.
- Credit Card Payment (Negative): Entering an expired credit card date during a payment process to see if the system detects and handles this scenario correctly.

End-To-End Testing

- End-to-End Testing is a comprehensive testing approach that evaluates the entire application flow from start to finish.
- It involves testing the interactions between various components and systems to ensure they work seamlessly together.



Example of End-to-End Testing for an E-commerce Application:
Scenario: User Makes a Purchase

Steps:

1. The user logs into the e-commerce website.
2. The user browses products, adds items to the cart, and proceeds to checkout.
3. The user provides shipping and payment information.
4. The system processes the payment and generates an order confirmation.
5. The user receives an email confirmation.

End-to-End Testing Checks:

1. Confirm that users can successfully log in.
2. Ensure the shopping cart calculates the correct total.
3. Verify that the checkout process collects and processes shipping and payment information accurately.
4. Check that the system generates a valid order confirmation.
5. Confirm that the user receives the expected email confirmation.

Globalization & Localization Testing

Globalization Testing

- Globalization Testing checks if an application can function seamlessly across different regions and cultures, accommodating various languages, date formats, and currencies.

Localization Testing

- Localization Testing ensures that an application is adapted for a specific locale or target audience by verifying language translations, cultural preferences, and regional requirements.
- In simple terms, globalization testing makes sure your software can work anywhere in the world, and localization testing ensures it's a good fit for the specific cultural and linguistic needs of a particular region.

Great ✅ Here's a set of **Interview Q&A for Session 6 (Testing Terminologies):**

Regression Testing

Q1. What is regression testing?
👉 Regression testing is testing performed to ensure that new changes (bug fixes, enhancements, or new features) do not break existing functionality.

Q2. Can you give an example of regression testing?

👉 If a new feature like profile picture upload is added to an app, regression testing ensures that login, profile view, and messaging still work correctly.

Q3. What are the types of regression testing?

👉 Unit Regression, Regional Regression (with impact analysis), and Full Regression.

Re-Testing

Q4. What is re-testing?

👉 Re-testing is testing a specific defect again after the developer has fixed it.

Q5. Difference between Regression Testing and Re-testing?

👉 Re-testing checks **whether a specific defect is fixed**.

👉 Regression testing checks **whether the fix has caused new defects in related areas**.

Smoke Testing

Q6. What is smoke testing?

👉 Smoke testing (Build Verification Testing) ensures that the build is stable and testable.

Q7. Can you give examples of smoke testing?

👉 Checking if the application installs properly, screens load correctly, database is running, and APIs respond.

Sanity Testing

Q8. What is sanity testing?

👉 Sanity testing ensures basic functionality works after a new build or bug fix (e.g., login, registration, logout).

Q9. Difference between Smoke Testing and Sanity Testing?

👉 Smoke testing = checks **build stability** (is it ready for testing?).

👉 Sanity testing = checks **basic functionality** (does it work correctly?).

Exploratory, Adhoc & Monkey Testing

Q10. What is exploratory testing?

👉 Testing done by exploring the application without formal requirements or test cases, usually to learn and discover functionality.

Q11. What is adhoc testing?

👉 Random testing done without documentation but with some knowledge of the application, mainly to break the system.

Q12. What is monkey testing?

👉 Random testing without any knowledge of the application, often used for gaming apps to check system robustness.

Q13. Difference between Exploratory, Adhoc, and Monkey Testing?

👉 Exploratory = To **learn and explore** the app.

👉 Adhoc = To **break the app** with tester's knowledge.

👉 Monkey = To **break the app randomly** without knowledge.

Positive & Negative Testing

Q14. What is positive testing?

👉 Verifying the system with valid inputs (e.g., login with correct username & password).

Q15. What is negative testing?

👉 Verifying the system with invalid inputs (e.g., login with wrong password, upload invalid file format).

End-to-End Testing

Q16. What is end-to-end testing?

👉 Testing the complete flow of an application from start to finish, covering integration of all modules.

Q17. Can you give an example of end-to-end testing?

👉 In e-commerce: login → browse items → add to cart → checkout → payment → order confirmation email.

Globalization & Localization Testing

Q18. What is globalization testing?

👉 Ensures the application works across multiple regions, languages, currencies, and date formats.

Q19. What is localization testing?

👉 Ensures the application is customized for a specific locale (correct translations, formats, cultural aspects).

Q20. What's the difference between globalization and localization testing?

👉 Globalization = Can the app work worldwide?

👉 Localization = Can the app work correctly for a **specific region**?

📘 Session 7

Test Design Techniques

Test Design Techniques

Test design techniques helps to design better cases. It reduce the number of test cases to be executed.

1. Equivalence Class Partitioning
2. Boundary Value Analysis (BVA)
3. Decision Table based testing
4. State Transition
5. Error Guessing

Equivalence Class Partitioning (ECP)

- Partition data into various classes and we can select data according to class then test.
- It reduce the number of test-cases and saves time for testing.

Example:
Name: *Allow only alphabets*

Equivalence Classes	Valid/Invalid	Test Data
A..Z	Valid	XYZ
a..z	Valid	zyz
Special Characters	Invalid	@#$%
Spaces	Invalid	Xy z
Numbers	Invalid	1234

Boundary Value Analysis (BVA)

- Boundary Value Analysis (BVA) is a testing technique where we focus on testing the boundaries or edges of valid and invalid input values.

Example:
Enter Age: *Allow digits from 18–35*

Parameter	Data	Valid/Invalid
Min	18	Valid
Min-1	17	Invalid
Min+1	19	Valid

Parameter	Data	Valid/Invalid
Max	35	Valid
Max-1	34	Valid
Max+1	36	Invalid

Decision Table

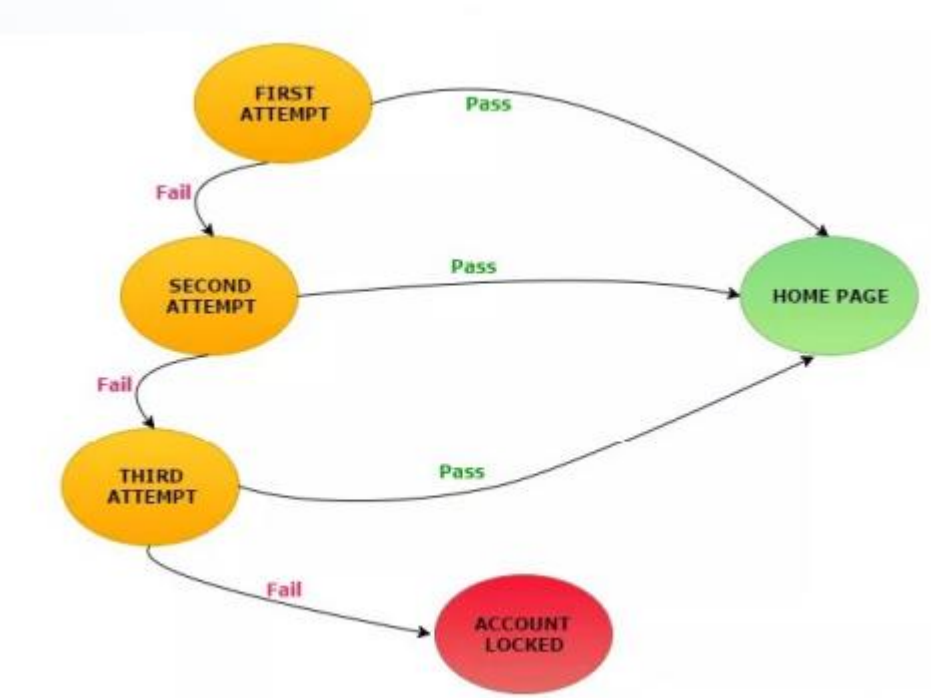
- Decision Table is also called as *Cause-Effect Table*.
- This technique will be used if we have more conditions and corresponding actions.
- In Decision table technique, we deal with combinations of inputs.
- To identify the test cases with decision table, we consider conditions and actions.
- Refer the below document for Decision Table.

State Transition

- In State Transition technique changes in input conditions change the state of the Application.
- This testing technique allows the tester to test the behavior of an AUT.
- The tester can perform this action by entering various input conditions in a sequence.
- In State transition technique, the testing team provides positive as well as negative input test values for evaluating the system behavior.

State Transition Diagram and State Transition Table

Example: *Login page of an application which locks the user after three wrong password attempts*



State	Login Attempt	Correct Password	Incorrect Password
S1	First attempt	S4	S2
S2	Second attempt	S4	S3
S3	Third attempt	S4	S5
S4	→ Home Page		
S5	→ Display message "Account locked. Please consult admin"		

Error Guessing

- Error guessing is a software testing technique where testers use their experience and intuition to identify potential areas of a system where errors or defects might occur.
- This technique relies on the tester's knowledge of the system.

Interview Q&A – Test Design Techniques

1. What are Test Design Techniques?

Answer:
Test design techniques are systematic approaches used to create effective test cases. They help in reducing the number of test cases while ensuring maximum test coverage and quality.

---

2. Name the main Test Design Techniques.

Answer:
The major test design techniques are:

1. Equivalence Class Partitioning (ECP)
2. Boundary Value Analysis (BVA)
3. Decision Table Testing
4. State Transition Testing
5. Error Guessing

---

3. Explain Equivalence Class Partitioning (ECP) with an example.

Answer:

- ECP divides the input data into groups (classes) where test cases can be designed for one representative value from each group.
- Example: For a field "Name" that only accepts alphabets:
    - Valid: A–Z, a–z
    - Invalid: Numbers (123), Special characters (@#$), Spaces (Xy z)

---

4. What is Boundary Value Analysis (BVA)?

Answer:

- BVA focuses on testing the boundaries of input values, as defects are often found at edges.
- Example: Age field allows 18–35.
    - Valid: 18, 19, 34, 35
    - Invalid: 17, 36

---

5. Difference between ECP and BVA?

Answer:

- **ECP** divides data into groups and picks one value from each group.
- **BVA** tests the edge values (min, max, min±1, max±1).
- Both reduce test cases, but BVA focuses on limits, while ECP focuses on representative groups.

---

6. What is a Decision Table in testing?

Answer:

- A decision table (also called Cause-Effect table) is used when there are multiple conditions and actions.
- It helps in identifying test cases for different combinations of inputs.

---

7. When do you use Decision Table Testing?

Answer:

- When the system has complex business rules with multiple conditions and corresponding outcomes.
- Example: Banking loan approval based on salary, age, and credit score.

---

8. Explain State Transition Testing with an example.

Answer:

- State transition testing is used when the output depends on the current state and previous inputs.
- Example: Login system locks the account after 3 failed attempts.
    - Correct password → Home Page
    - 3 wrong attempts → Account Locked

---

9. What is the difference between State Transition and Decision Table Testing?

Answer:

- **State Transition**: Focuses on changes in state due to events (sequential input).
- **Decision Table**: Focuses on combinations of conditions and actions.

---

10. What is Error Guessing in testing?

Answer:

- Error Guessing is based on tester's experience and intuition to find defects in areas where issues are likely.
- Example: Entering special characters in "Username" field, trying invalid file formats for upload, etc.

---

11. Is Error Guessing a formal technique?

Answer:

- No, it is informal and experience-driven. It complements formal techniques like BVA and ECP.

---

12. Which technique would you choose if requirements are not clear?

Answer:

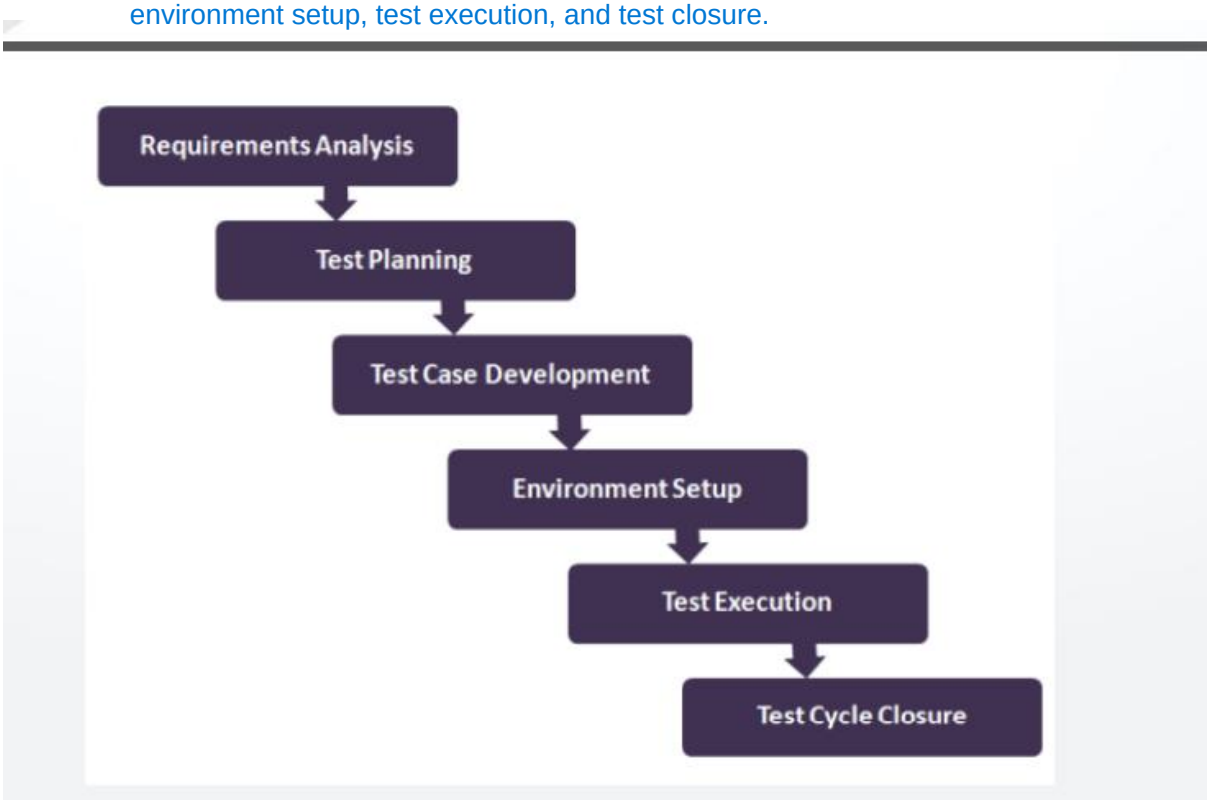
- **Error Guessing** or **Exploratory Testing**, since formal techniques depend on well-defined requirements.

📘 Session 8

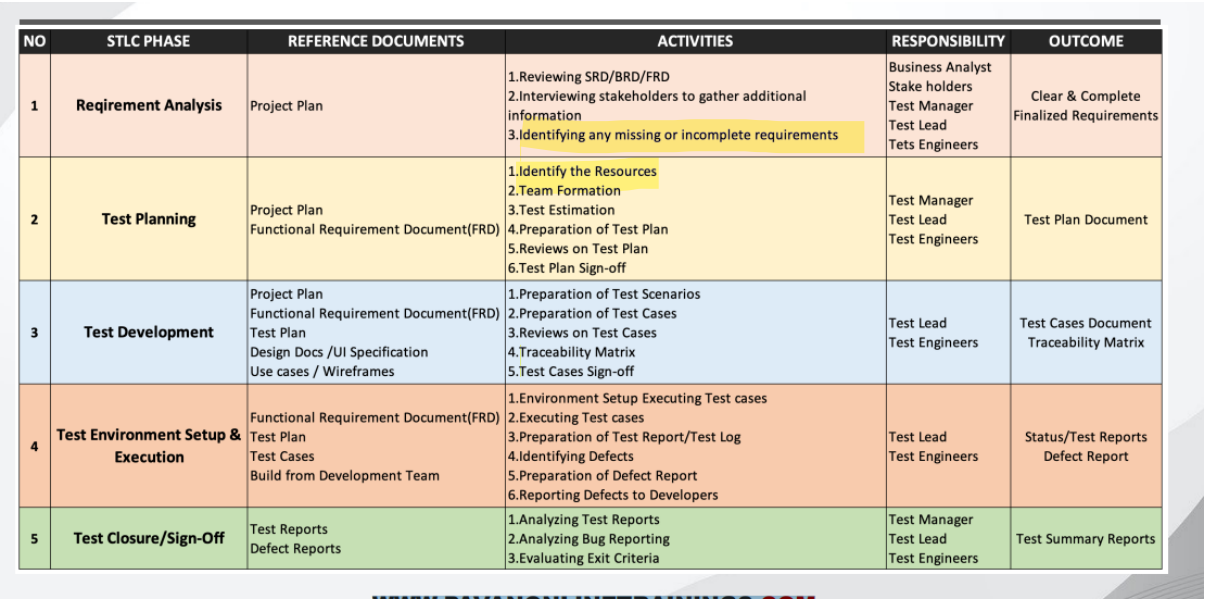
Software Testing Life Cycle (STLC)

STLC

STLC is a sequence of specific activities conducted during the testing process to ensure software quality. It includes phases like requirement analysis, test planning, test case design, test environment setup, test execution, and test closure.



STLC



NO	STLC PHASE	REFERENCE DOCUMENTS	ACTIVITIES	RESPONSIBILITY	OUTCOME
1	**Reqirement Analysis**	Project Plan	1.Reviewing SRD/BRD/FRD 2.Interviewing stakeholders to gather additional information 3.Identifying any missing or incomplete requirements	Business Analyst Stake holders Test Manager Test Lead Tets Engineers	Clear & Complete Finalized Requirements
2	**Test Planning**	Project Plan Functional Requirement Document(FRD)	1.Identify the Resources 2.Team Formation 3.Test Estimation 4.Preparation of Test Plan 5.Reviews on Test Plan 6.Test Plan Sign-off	Test Manager Test Lead Test Engineers	Test Plan Document
3	**Test Development**	Project Plan Functional Requirement Document(FRD) Test Plan Design Docs /UI Specification Use cases / Wireframes	1.Preparation of Test Scenarios 2.Preparation of Test Cases 3.Reviews on Test Cases 4.Traceability Matrix 5.Test Cases Sign-off	Test Lead Test Engineers	Test Cases Document Traceability Matrix
4	**Test Environment Setup & Execution**	Functional Requirement Document(FRD) Test Plan Test Cases Build from Development Team	1.Environment Setup Executing Test cases 2.Executing Test cases 3.Preparation of Test Report/Test Log 4.Identifying Defects 5.Preparation of Defect Report 6.Reporting Defects to Developers	Test Lead Test Engineers	Status/Test Reports Defect Report
5	**Test Closure/Sign-Off**	Test Reports Defect Reports	1.Analyzing Test Reports 2.Analyzing Bug Reporting 3.Evaluating Exit Criteria	Test Manager Test Lead Test Engineers	Test Summary Reports

Test Plan Contents

- Overview
- Scope of testing
- Features to be tested
- Features not to be tested
- Test Environments
- Test Strategy
- Defect Reporting Procedure
- Roles & Responsibilities
- Test Schedule
- Test Deliverables
- Entry and Exit Criteria
- Suspension and Resumption Criteria
- Tools
- Risks and Mitigations
- Approvals

Entry Criteria
Definition: Conditions that must be fulfilled before testing can begin.
Purpose: To ensure the testing team has everything needed to start testing effectively.

Examples:
Requirements and design documents are approved.
Test environment is set up and stable.
Test data is prepared.
Test cases are written and reviewed.
Code is unit tested and ready for integration/system testing.

Exit Criteria
Definition: Conditions that must be satisfied to conclude testing.
Purpose: To confirm that testing goals have been met and the product is ready for the next phase or release.

Examples:
All planned test cases are executed.
Defects are resolved or deferred with justification.
Test coverage meets the required threshold.
No critical or high-severity bugs remain.
Test summary report is completed and signed off.

Use Case, Test Scenario & Test Case

- Use Case: Describes the requirement. It contains three key elements:
    - Actor: Represents the user, either an individual or a group, interacting with a process.

    - Action: Specifies the steps or activities taken to achieve the desired outcome.
    - Goal/Outcome: Defines the successful result or the expected outcome of the user's interaction with the system.
- Test Scenario: A possible area to be tested (What to test).
- Test Case: Step by step actions to be performed to validate functionality of AUT (How to test). Test case contains test steps, expected result & actual result.

Use Case: Buy a product

- Actor: Customer
- Goal: Purchase a specific product on the website.
- Steps:
    - Browse the product catalog.
    - Select a product and add it to the cart.
    - Proceed to checkout.
    - Enter shipping and payment information.
    - Place the order.
    - Receive confirmation and order tracking information.

Test Scenario & Test Case

- Test Scenario: Successful purchase with valid credit card
    - This scenario covers a happy path where the customer completes the purchase without any issues.
- Test Case: TC01 – Purchase with valid credit card
    - Pre-conditions: Customer has a valid account and a product is added to the cart.
    - Steps:
        - Enter valid shipping address.
        - Enter valid credit card information (number, expiry date, CVV).
        - Click "Place Order".
    - Expected Result:
        - Order confirmation is displayed with order details.
        - Credit card is charged successfully.
        - Order status reflects "Processing".

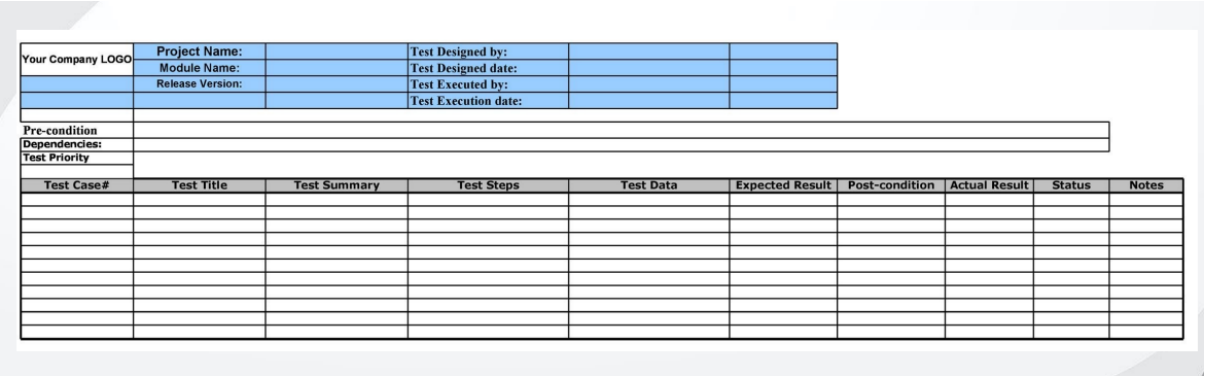
Additional Test Cases

- TC02 – Purchase with invalid credit card number
- TC03 – Purchase with insufficient funds
- TC04 – Purchase with missing shipping address
- TC05 – Purchase with guest checkout

Test Case Contents

- Test Case ID
- Test Case Title
- Description
- Pre-condition
- Priority (P0, P1, P2, P3) – order
- Requirement ID
- Steps/Actions
- Expected Result
- Actual Result
- Test data

Test Case Template



Your Company LOGO	Project Name:		Test Designed by:		
	Module Name:		Test Designed date:		
	Release Version:		Test Executed by:		
			Test Execution date:		

Pre-condition	
Dependencies:	
Test Priority	

Test Case#	Test Title	Test Summary	Test Steps	Test Data	Expected Result	Post-condition	Actual Result	Status	Notes

Requirement Traceability Matrix (RTM)

- The Requirement Traceability Matrix (RTM) is a document that establishes a mapping between requirements and test cases. Its primary purpose is to ensure that all the requirements specified for a system are covered by corresponding test cases.
- Key components of an RTM include:

- Requirement ID: A unique identifier assigned to each requirement, making it easy to reference and track.
- Req Description: A detailed description of each requirement, outlining what functionality or behavior is expected from the system.
- Test Case IDs: A list of test cases associated with each requirement. This establishes a clear link between the requirements and the corresponding test cases that verify or validate them.

Requirement Traceability Matrix (RTM) Example

Requirement ID	Requirement Description	Test Case ID's
REQ001	User should be able to log in	TC001, TC002
REQ002	System should display product details	TC003, TC004
REQ003	User can add items to the shopping cart	TC005, TC006

Here's a simplified example of what an RTM might look like.

Test Environment / Test Bed

- Test Environment is a platform specially built for test case execution on the software product.
- It is created by integrating the required software and hardware along with proper network configurations.
- Test environment simulates production/real time environment.
- Another name of test environment is Test Bed.

Test Execution

- Activities:
  - Test cases are executed based on the test planning.
  - Status of test cases are marked, like Passed, Failed, Blocked, Run, and others.
  - Documentation of test results and logging defects for failed cases is done.
  - All the blocked and failed test cases are assigned bug IDs.
  - Retesting once the defects are fixed.
  - Defects are tracked till closure.

Guidelines for Test Execution

- The build being deployed to the QA environment is the most important part of the test execution cycle.
- Test execution is done in Quality Assurance (QA) environment.
- Test execution happens in multiple cycles.
- Test execution phase consists of executing the test cases + test scripts (if automation).

Interview Q&A on STLC (Session 8)

Q1. What is STLC (Software Testing Life Cycle)?
A1. STLC is a sequence of specific activities conducted during the testing process to ensure software quality. It includes phases like requirement analysis, test planning, test case design, test environment setup, test execution, and test closure.

Q2. What are the main phases of STLC?
A2.

1. Requirement Analysis
2. Test Planning
3. Test Case Design
4. Test Environment Setup
5. Test Execution
6. Defect Reporting & Tracking
7. Test Closure

Q3. What does a Test Plan contain?
A3. A test plan contains:

- Overview
- Scope of testing
- Features to be tested / not tested
- Test environments
- Test strategy
- Defect reporting procedure
- Roles & responsibilities
- Test schedule
- Test deliverables

- Entry & Exit criteria
- Suspension & Resumption criteria
- Tools
- Risks & Mitigations
- Approvals

Q4. What is the difference between Use Case, Test Scenario, and Test Case?
A4.

- **Use Case**: Describes requirement with Actor, Action, and Goal. (What user wants to achieve).
- **Test Scenario**: High-level statement of what to test.
- **Test Case**: Step-by-step actions to validate the functionality of the AUT with test steps, expected and actual results.

Q5. Can you give an example of Use Case, Test Scenario, and Test Case?
A5.

- **Use Case**: Buy a product online.
- **Test Scenario**: Successful purchase with valid credit card.
- **Test Case**: Enter valid shipping address → enter valid credit card → click "Place Order" → expect order confirmation and payment success.

Q6. What is RTM (Requirement Traceability Matrix)?
A6. RTM is a document that maps requirements to corresponding test cases to ensure that all requirements are covered by tests.

Q7. Why is RTM important?
A7.

- Ensures 100% requirement coverage.
- Helps track missing functionality.
- Useful for impact analysis when requirements change.
- Maintains a clear link between requirements and test cases.

Q8. What is a Test Environment (Test Bed)?
A8. A test environment is a setup of hardware, software, and network configuration where testing is executed. It simulates the production environment.

Q9. What activities are done in the Test Execution phase?
A9.

- Execute test cases as per test plan.
- Mark test cases as Pass, Fail, Blocked, or Run.
- Log defects for failed cases.
- Assign bug IDs.
- Retest after fixing.
- Track defects till closure.

Q10. What are the guidelines for Test Execution?
A10.

- Ensure the correct build is deployed in QA.
- Execution happens in QA environment.
- Done in multiple cycles.
- Includes execution of both manual and automation test scripts.

📘 Session 9

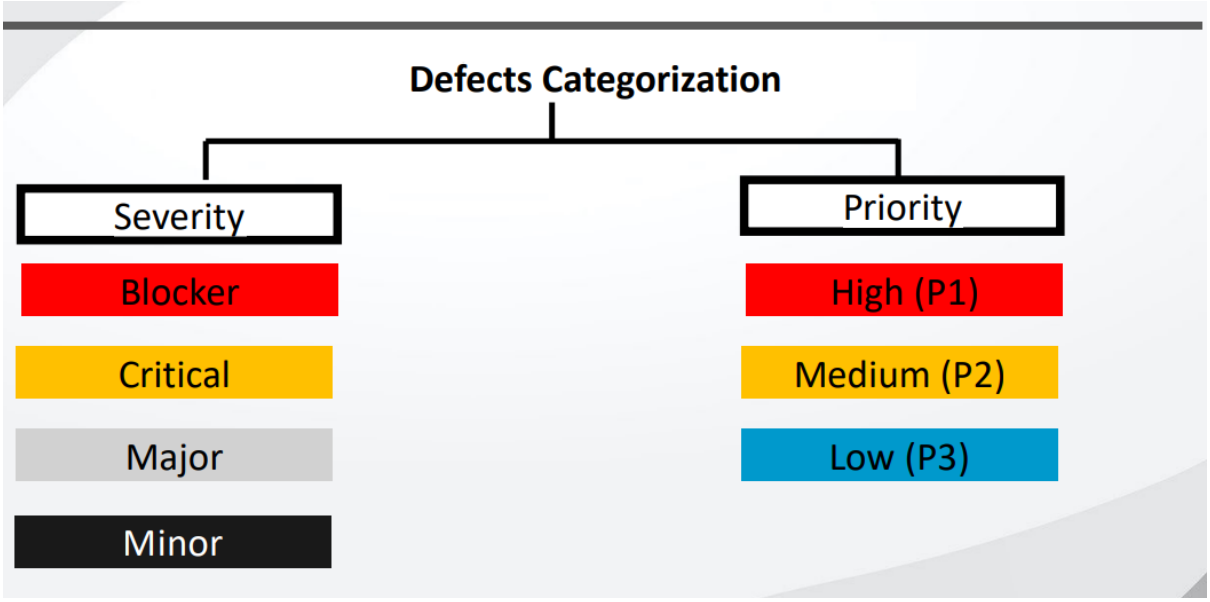
Defects

- Any mismatched functionality found in an application is called a Defect/Bug/Issue.
- During Test Execution, Test engineers report mismatches as defects to developers through templates or using tools.

Defect Reporting Tools:

- Clear Quest
- DevTrack
- Jira
- Quality Center
- Bug Jilla etc.

Defect Classification



Defects Categorization:

- **Severity**
- **Priority**

Severity Levels: Blocker, Critical, Major, Minor
Priority Levels: High (P1), Medium (P2), Low (P3)

Defect Severity

- Severity describes the seriousness of defect and how much impact it has on business workflow.
- Tester decides the Severity of the defect.
- Defect severity can be categorized into four classes:

1. **Blocker (Show stopper):** Nothing can proceed further.
    - Ex: Application crashed, Login not working.
2. **Critical:** Main/basic functionality is not working. Customer business workflow is broken.
    - Ex1: Fund transfer not working in net banking.
    - Ex2: Ordering product in e-commerce application not working.
3. **Major:** Causes undesirable behavior, but feature is still functional.
    - Ex1: After sending email, no confirmation message.
    - Ex2: After booking cab, no confirmation.
4. **Minor:** Won't cause major break-down of the system.

- Ex: Look and feel issues, spellings, alignments.

Defect Priority

- Priority describes the importance of defect.
- Tester decides the Priority of the defect. Developer or product manager can change priority later if needed.
- Defects will be fixed by developer based on priority.

Defect Priority Classes:

- **P1 (High):** Must be resolved immediately as it affects the system severely.
- **P2 (Medium):** Can wait until a new version/build is created.
- **P3 (Low):** Developer can fix it in later releases.

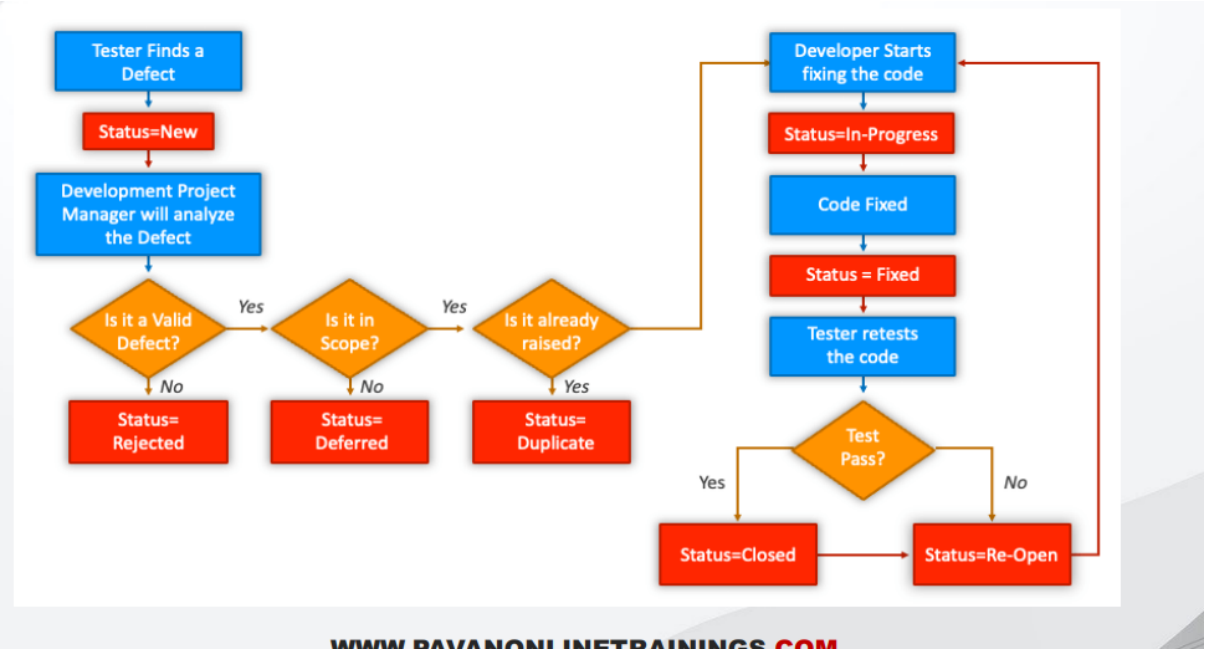
Difference Between Severity and Priority

Severity	Priority
Defined by the impact of a problem on application functionality.	Defined by the impact on business.
Category decided by testers.	Decided by testers, but developers/product owners can change it.
Deals with technical aspects of the application.	Deals with timeframe/order to fix defects.
Value does not change with time (fixed).	Value is subjective and may change over time.

Examples on Priority & Severity

Issue Description	Priority	Severity
A spelling mistake in a page not frequently navigated.	Low	Low
Application crashing in some corner case.	Low	High
Slight change in logo color/spelling mistake in company name.	High	Low
Login functionality issue (cannot login).	High	High
Web page not found when user clicks rare link.	Low	High
Cosmetic/spelling issue in a paragraph/page.	Low	Low

Defect (Bug) Life Cycle



Defect Resolution

After receiving defect report from testing team, development team conducts review meeting to fix defects. Then they send a **Resolution Type** to testing team.

Resolution Types:

- Accept
- Reject
- Duplicate
- Enhancement
- Need more information
- Not Reproducible
- Fixed
- As Designed

Defect Report Contents

- **Defect ID:** Unique identification number.
- **Defect Description:** Detailed info, module where defect found.
- **Version:** Version of application where defect found.
- **Steps:** Detailed steps with screenshots to reproduce defect.

- **Date Raised:** When defect is raised.
- **Reference:** Provide reference to requirements, design, architecture, screenshots.
- **Detected By:** Tester who raised defect.
- **Status:** Status of defect.
- **Fixed By:** Developer who fixed it.
- **Date Closed:** When defect closed.
- **Severity:** Describes impact.
- **Priority:** Related to fixing urgency. (High/Medium/Low).

Test Metrics

SNO – Required Data

1. No. of Requirements
2. Avg. No. of Test Cases written per Requirement
3. Total No. of Test Cases written
4. Total No. of Test Cases Executed
5. No. of Test Cases Passed
6. No. of Test Cases Failed
7. No. of Test Cases Blocked
8. No. of Test Cases Unexecuted
9. Total No. of Defects Identified
10. Critical Defects Count
11. Higher Defects Count
12. Medium Defects Count
13. Low Defects Count
14. Customer Defects
15. No. of defects found in UAT

Test Metrics Calculations

- **% Test cases Executed:**
  (Executed / Total Written) * 100
- **% Test cases Not Executed:**
  (Not Executed / Total Written) * 100

- **% Test cases Passed:**
  (Passed / Executed) * 100
- **% Test cases Failed:**
  (Failed / Executed) * 100
- **% Test cases Blocked:**
  (Blocked / Executed) * 100

Additional Metrics

- **Defect Density:**
  No. of defects found / No. of requirements
- **Defect Removal Efficiency (DRE):**
  (Fixed Defects / (Fixed Defects + Missed Defects)) * 100
    - A = Fixed Defects
    - B = Missed Defects
- **Defect Leakage:**
  (Defects in UAT / Defects in Testing) * 100
- **Defect Rejection Ratio:**
  (Rejected Defects / Total Raised) * 100
- **Defect Age:**
  Fixed Date – Reported Date
- **Customer Satisfaction:**
  No. of complaints per period of time

Defects – Interview Q&A

Q1. What is a defect/bug/issue in software testing?
A defect (or bug/issue) is any mismatch between the expected functionality and the actual functionality of the application.

Q2. How are defects reported?
Defects are reported by test engineers during test execution using templates or defect tracking tools like Jira, Quality Center, Bugzilla, etc.

Q3. Name some popular defect reporting tools.

- Jira
- Quality Center (HP ALM)
- Bugzilla
- Clear Quest
- DevTrack

Severity & Priority – Interview Q&A

Q4. What is defect severity? Who decides it?

- Severity describes how serious the defect is and its impact on business workflow.
- Severity is decided by the **tester**.

Q5. What are the levels of defect severity with examples?

1. **Blocker (Showstopper):** Application crash, login not working.
2. **Critical:** Major functionality broken (e.g., fund transfer not working).
3. **Major:** Functionality works but with undesirable behavior (e.g., no confirmation after booking).
4. **Minor:** Cosmetic issues like spelling, alignment.

Q6. What is defect priority? Who decides it?

- Priority refers to the urgency of fixing a defect.
- Initially decided by the **tester**, but developers/product managers can change it based on business needs.

Q7. What are the levels of defect priority?

- **P1 (High):** Must fix immediately (e.g., login issue).
- **P2 (Medium):** Can wait for next build.
- **P3 (Low):** Fixed in later releases.

Q8. What is the difference between Severity and Priority?

- **Severity** → Technical impact on functionality (decided by tester).
- **Priority** → Business urgency to fix (decided by tester but can be changed by dev/product team).

Examples – Interview Q&A

Q9. Give an example of Low Severity & High Priority defect.
Spelling mistake in company name or wrong logo color.

Q10. Give an example of High Severity & Low Priority defect.
Application crash in a rare scenario (corner case).

Defect Life Cycle – Interview Q&A

Q11. What is a defect life cycle?
It is the journey of a defect from its identification until its closure, covering different states like New → Assigned → Fixed → Retest → Closed.

Q12. What are defect resolution types?

- Accept
- Reject
- Duplicate
- Enhancement
- Need more information
- Not reproducible
- Fixed
- As designed

Defect Report – Interview Q&A

Q13. What are the key contents of a defect report?

- Defect ID
- Description
- Version
- Steps to reproduce (with screenshots)
- Date Raised
- Reference documents
- Detected By
- Status
- Fixed By
- Date Closed
- Severity
- Priority

Test Metrics – Interview Q&A

Q14. What are test metrics? Why are they important?
Test metrics are quantitative measures used to track test progress, quality, and efficiency. They help management in decision-making and process improvement.

Q15. What is Defect Density?
No. of defects identified / No. of requirements.

Q16. What is Defect Removal Efficiency (DRE)?
(Fixed Defects / (Fixed Defects + Missed Defects)) * 100

Q17. What is Defect Leakage?
(No. of defects found in UAT / No. of defects found during testing) * 100

Q18. What is Defect Rejection Ratio?
(No. of defects rejected / Total defects raised) * 100

Q19. What is Defect Age?
Time between defect reported date and defect fixed date.

Q20. Give examples of Test Case Metrics.

- % Test Cases Executed
- % Test Cases Passed
- % Test Cases Failed
- % Test Cases Blocked
- % Test Cases Not Executed