

Manual Testing

Manual Testing

Module 1: Testing Concepts(Theory) What?

Module 2: Testing Projects(Practical) How?

Module 3: Agile Process

Jira Tools

Basic QA Topics for Interviews

Module 1

Module 1: Software Testing Concepts:

☒ Module 1: Software Testing Concepts (Topics 1 to 5)

☒ 1. What is Software? Types of Software

- ◇ What is Software?

Software is a **set of instructions, programs, or data** used to operate computers and perform specific tasks.

💡 It tells hardware what to do — without software, hardware is useless.

- ◇ Types of Software:

1. System Software

- Controls hardware and basic system operations
- Example: Operating Systems (Windows, Linux), Drivers, Utilities

• Application Software

- ◇ Used by end-users to perform specific tasks
- ◇ Example: Microsoft Word, Google Chrome, WhatsApp

• Programming Software

- ◇ Helps developers write code
- ◇ Example: IDEs (Visual Studio, PyCharm), compilers, debuggers

• Middleware

- ◇ Acts as a bridge between system software and application software
- ◇ Example: API managers, database connectors

☒ 2. What is Software Testing?

- ◇ Definition:

Software Testing is the **process of evaluating a software application** to detect bugs, check if it meets the requirements, and ensure it works correctly.

- ☒ Goal: **Find defects** and make sure the software is **high quality, reliable, and user-friendly**.

- ◇ Key Objectives:

- ◇ Verify software works as intended
- ◇ Identify and fix bugs early
- ◇ Improve software performance and usability

◇ Ensure customer satisfaction

◇ Real Example:

If you're testing an e-commerce website:

◇ Can users add items to cart?

◇ Does the payment gateway work?

◇ What happens if an invalid card is used?

3. What is Software Quality?

◇ Definition:

Software Quality means how **well the software performs its required functions**, how reliable it is, and how easy it is to use and maintain.

◇ Dimensions of Quality:

Aspect	Description
Functionality	Does it do what it's supposed to?
Reliability	Can it run for long without crashing?
Usability	Is it user-friendly?
Efficiency	Does it use resources wisely (CPU/RAM)?
Maintainability	Can it be easily updated or fixed?
Portability	Can it work on different systems/devices?

🔗 Quality is not just **no bugs** — it's about **overall user satisfaction and performance**.

4. Project vs Product

Feature	Project	Product
Goal	Deliver a solution for a specific client	Build a solution for general users

Feature	Project	Product
Scope	Custom-defined requirements	Market-driven, broader scope
Timeline	Fixed timeline and budget	Evolving, continuous development
Example	Website for a specific business	WhatsApp, MS Word, Adobe Photoshop

☑ A QA in a **project** may focus on client-specific features, while in a **product** team, focus is on mass usage and scalability.

▣ 5. Why Do We Need Testing?

◇ Importance of Testing:

- ◇ To catch **bugs before release**
- ◇ To ensure the **product meets user requirements**
- ◇ To prevent **costly failures in production**
- ◇ To provide a **better user experience**
- ◇ To gain **client/customer trust**

● Without testing, software may crash, give wrong outputs, or leak user data.

◇ Real Case Example:

The **Therac-25 Radiation Machine Bug** (1980s) — a software bug caused patients to receive fatal radiation doses.

▣ 6. Error, Bug & Failure

Term	Definition
Error	A human mistake made by a developer or tester (e.g., wrong logic in code).

Term	Definition
Bug (Defect)	A problem or issue in the code caused by an error that causes incorrect behavior.
Failure	When the system doesn't behave as expected during execution due to a bug.



Flow Example:

Developer makes an **error** → It causes a **bug** in the code → User sees a **failure** when using the software.



Example:

If the login button doesn't work:

- Developer forgot to connect it → **Error**
- That mistake causes a **Bug**
- App crashes when user clicks login → **Failure**



7. Why the Software Has Bugs?

◇ Common Reasons:

1. **Misunderstood Requirements** – Developer/tester misunderstands what user wants
2. **Time Pressure** – Deadlines cause rushed development
3. **Complex Code** – More logic = more chances of mistakes
4. **Human Error** – Developers and testers are human, mistakes happen
5. **Changing Requirements** – Last-minute changes may break existing features
6. **Lack of Testing** – Incomplete or skipped testing leads to undetected issues
7. **Third-party Issues** – External APIs or tools may behave unexpectedly



Real Example:

- ◇ A feature that worked in testing breaks after a library update = **Third-party issue**



8. SDLC & STLC



SDLC: Software Development Life Cycle

SDLC defines the **phases of software development** from idea to release.

Phase	Purpose
Requirements	Understand what the client needs
Design	Create architecture and plan
Development	Write the code
Testing	Test for bugs and issues
Deployment	Release to users
Maintenance	Fix issues and improve over time

◇ STLC: Software Testing Life Cycle

STLC defines the **phases of testing** within SDLC.

Phase	Purpose
Requirement Analysis	Understand what to test
Test Planning	Define test scope, strategy, tools, and schedule
Test Case Design	Write test cases/ scenarios
Test Environment Setup	Prepare hardware, software for testing
Test Execution	Run test cases, log defects
Test Closure	Document results, lessons learned

☒ STLC is **part of SDLC**, but only focuses on testing-related steps.

9. Waterfall Model

◇ What is it?

Waterfall Model is a **linear and sequential SDLC model** where each phase happens one after another (like a waterfall).

◇ Phases:

1. Requirements
2. Design
3. Implementation (Coding)
4. Testing
5. Deployment
6. Maintenance

● Pros:

- ◇ Simple and easy to manage
- ◇ Clear documentation at each step

● Cons:

- ◇ No flexibility for changes
- ◇ Testing happens **only after development** — late bug discovery
- ◇ Not suitable for large or dynamic projects

💡 Use Case:

Best for **small, well-defined projects** with stable requirements.

▣ 10. Spiral Model

◇ What is it?

Spiral Model combines features of **Waterfall** and **Prototyping**. It works in **repeated cycles (spirals)** and focuses on **risk management**.

◇ Phases in Each Spiral:

1. Planning
2. Risk Analysis
3. Development
4. Evaluation

Then repeat the spiral with improvements and additions.

● Pros:

- ◇ Focus on **risk handling**
- ◇ Good for **complex and high-risk projects**
- ◇ Supports **feedback and iteration**

● Cons:

- ◇ Expensive
- ◇ Requires **expert risk analysis**
- ◇ Complex to manage

💡 Use Case:

Used in **military, space, or high-risk systems** where failure is not an option.

11. V-Model (Verification & Validation Model)

◇ What is the V-Model?

The **V-Model** (Validation & Verification model) is an SDLC model where **development and testing activities run in parallel** — forming a "V" shape.

Each development phase has a **corresponding testing phase**.

📁 V-Model Structure:

Development Phase	Corresponding Testing Phase
Requirement Analysis	Acceptance Testing
System Design	System Testing
Architectural Design	Integration Testing
Module Design	Unit Testing
Coding (Middle of the V)	Execution (Bottom-up Testing)

🟡 Pros:

- Testing starts early (parallel with development)
- Easy defect tracking at every stage

🟡 Cons:

- ◇ Not flexible for changes
- ◇ High initial planning needed

12. QA, QC & QE

◇ QA (Quality Assurance)

◇ **Process-focused**

◇ Ensures the right process is followed during development

◇ **Prevention-based**

◇ Example: Audits, process improvement

◇ QC (Quality Control)

◇ **Product-focused**

◇ Involves identifying bugs in the final product

◇ **Detection-based**

◇ Example: Testing, bug reporting

◇ QE (Quality Engineering)

◇ Combines QA + DevOps + Automation

◇ Ensures **quality is built throughout development**

- ◇ Involves **automated testing, CI/CD, and quality mindset**

Role	Focus	Example
QA	Process Quality	Process audits, documentation
QC	Product Quality	Finding bugs in app
QE	Whole Lifecycle	Automation, CI/CD, testing mindset

13. Different Levels of Software Testing

There are **4 major levels of testing**:

1. Unit Testing

- ◇ Tests individual units/modules of code
- ◇ Performed by developers
- ◇ Example: Test if a login function works independently

2. Integration Testing

- ◇ Tests how modules interact
- ◇ Example: Login page + dashboard integration

3. System Testing

- ◇ Tests the complete software as a whole
- ◇ End-to-end functionality
- ◇ Example: Complete e-commerce flow

4. Acceptance Testing

- ◇ Done to check if software meets business requirements
- ◇ Done by clients or QA team
- ◇ Example: Client tests if software works as expected

14. White Box & Black Box Testing

- ◇ White Box Testing:
 - ◇ Tester knows internal code, logic, and structure
 - ◇ Mainly done by developers
 - ◇ Also called **glass-box testing**

- ◇ Black Box Testing:
 - ◇ Tester doesn't know internal code

- ◇ Focuses on **inputs and outputs**
- ◇ Done by QA testers
- ◇ Also called **behavioral testing**

Testing Type	Access to Code	Done By	Focus
White Box	Yes	Developer	Code logic, paths
Black Box	No	QA Tester	Functional behavior/output

15. Static Testing & Dynamic Testing

- ◇ Static Testing:
 - ◇ Testing without executing code
 - ◇ Involves reviews, walkthroughs, and inspections
 - ◇ Performed in early stages
 - ◇ Example: Reviewing test cases or requirements documents
- ◇ Dynamic Testing:
 - ◇ Testing by executing code
 - ◇ Checks how the software behaves in real-time
 - ◇ Example: Manual or automated test execution

Type	Execution	Examples
Static Testing	No	Code review, document analysis
Dynamic Testing	Yes	Functional/Regression Testing

16. Verification & Validation

- ◇ Verification (Are we building the product right?)
 - Process of checking **documents, plans, requirements**
 - Done without executing code
 - Focused on **process quality**
 - Example: Reviewing design documents, checking FRS
- ◇ Validation (Are we building the right product?)
 - ◇ Testing the actual product/software
 - ◇ Executing code to find bugs
 - ◇ Focused on **product quality**

- ◇ Example: Running test cases on app features

Stage	Focus	Type	Example
Verification	Process correctness	Static Test	Requirement review
Validation	Product correctness	Dynamic Test	Functionality testing

■ 17. System Testing Types

System testing is a **complete end-to-end test** of the fully developed software. Below are **types of system testing**:

Type	Description
<input checked="" type="checkbox"/> Functional Testing	Tests if features work as expected
<input checked="" type="checkbox"/> Performance Testing	Tests speed and responsiveness
<input checked="" type="checkbox"/> Security Testing	Tests for vulnerabilities, access control
<input checked="" type="checkbox"/> Usability Testing	Tests if UI/UX is user-friendly
<input checked="" type="checkbox"/> Compatibility Testing	Tests on different OS/devices/browsers
<input checked="" type="checkbox"/> Recovery Testing	Tests app recovery after crash or failure
<input checked="" type="checkbox"/> Installation Testing	Checks installation/uninstallation
<input checked="" type="checkbox"/> Load Testing	Tests behavior under heavy load
<input checked="" type="checkbox"/> Stress Testing	Tests behavior under extreme conditions

18. GUI Testing (Graphical User Interface Testing)

- ◇ What is GUI Testing?
- ◇ Verifies that the **user interface** looks and behaves correctly.
- ◇ Ensures buttons, menus, icons, colors, fonts, layout, etc., are working as intended.

Checklist:

- ◇ Is the logo visible?
- ◇ Are all buttons clickable?
- ◇ Is alignment consistent?
- ◇ Is font readable?
- ◇ Does UI behave correctly on all screen sizes?

 Tools: Selenium, TestComplete, Ranorex

19. Functional & Non-Functional Testing

Type	Description	Examples
◇ Functional Testing	Tests what the system does	Login, signup, cart functionality
◇ Non-Functional Testing	Tests how the system behaves (performance, security, usability)	Load time, speed, UI look, etc.

- ◇ Functional Testing Includes:
 - ◇ Smoke Testing
 - ◇ Regression Testing
 - ◇ Integration Testing
 - ◇ Sanity Testing
- ◇ Non-Functional Testing Includes:
 - ◇ Load Testing
 - ◇ Stress Testing
 - ◇ Volume Testing
 - ◇ Usability Testing
 - ◇ Security Testing

20. Test Design Techniques

- ◇ What is it?
- These are techniques used to **create effective test cases** based on the requirements.

◇ Common Techniques:

1. Equivalence Partitioning

- ◇ Divide input into **valid and invalid groups**
- ◇ Test only one value from each group

• Boundary Value Analysis

- ◇ Test values **at the boundaries** (min, max, just inside/outside)

• Decision Table Testing

- ◇ Use tables to show combinations of inputs & outputs

• State Transition Testing

- ◇ Test system behavior when state changes (e.g., login → dashboard)

• Error Guessing

- ◇ Tester guesses possible areas where bugs can occur based on experience

21. Re-Testing & Regression Testing

Aspect	Re-Testing	Regression Testing
← END Purpose	Check if a specific defect is fixed	Check if new code changes broke old features
← END Based on	Bug fixes	Any change in code (feature or bug)
← END Scope	Narrow – specific test cases	Wide – entire app can be tested
← END Example	Bug: Login button not working → fix → retest login only	Add profile feature → test login, signup, dashboard, etc.

🔑 Both are important:

• **Re-Testing** = Confirm the fix

• **Regression** = Confirm **nothing else broke**

22. Exploratory Testing

- ◇ What is it?
- ◇ Tester explores the app **without pre-written test cases**.
- ◇ Uses **experience, creativity, and logic** to find bugs.

🔍 Key Features:

- ◇ No documentation needed
- ◇ Done **on the fly**
- ◇ Helps find **unexpected issues**

🧠 Example:

You start testing a shopping cart. While doing that, you suddenly decide to check what happens if you:

- ◇ Add 100 items
 - ◇ Apply invalid promo codes
 - ◇ Leave the payment page idle for 20 minutes
- => These are **exploratory ideas**

📦 23. Adhoc Testing

- ◇ What is it?
- ◇ **Informal** testing done without planning or documentation.
- ◇ Aim is to **break the system** using unconventional or random inputs.

✂️ **Tester tries to "act like a user" who may not follow rules.**

📋 Difference from Exploratory:

- ◇ Adhoc = Random, unstructured
- ◇ Exploratory = Based on experience + logic (more strategic)

🔧 Example:

Entering special characters in every field, pressing buttons randomly, refreshing in middle of payment → **Adhoc testing behavior**

📦 24. Sanity & Smoke Testing

Test Type	Sanity Testing	Smoke Testing
🔍 Purpose	Verify specific bug fixes or minor features	Basic check to see if app is stable
📋 Scope	Narrow – focused areas	Broad – high-level testing
🔧 When Used	After regression / bug fix	After new build deployment
📄 Test Cases	Not scripted	Usually scripted

🧠 Examples:

- ◇ **Sanity:** Check if newly added "edit profile" works correctly

◇ **Smoke:** Open app → login → navigate → no crash = ☒ passed

25. End-To-End (E2E) Testing

◇ What is it?

◇ Testing the **entire application flow** from start to finish, simulating **real-world usage**.

✎ It includes:

◇ Frontend + Backend

◇ APIs + Databases

◇ Third-party integrations

📋 Goal:

Verify that **all components work together** properly.

🧠 Example:

For an e-commerce app:

1. Login →
2. Search product →
3. Add to cart →
4. Apply discount →
5. Checkout →
6. Payment →
7. Order confirmation

☒ All steps = **One End-to-End test**

26. STLC (Software Testing Life Cycle)

◇ What is STLC?

STLC stands for **Software Testing Life Cycle**, which includes all **steps followed during testing** a software product.

📋 STLC Phases:

Phase	Purpose
1 Requirement Analysis	Understand what needs to be tested
2 Test Planning	Create test plan, estimate effort, assign roles

Phase	Purpose
3 Test Case Design	Write test cases & test data
4 Test Environment Setup	Prepare software/hardware for testing
5 Test Execution	Execute test cases & report bugs
6 Test Closure	Final summary of testing: lessons learned, metrics, what went well ✓

🧠 It's different from SDLC – STLC only focuses on **testing**, not entire development.

27. Use Case, Test Scenario & Test Case

Term	Description	Example (Login Page)
Use Case	Real-world functionality the system provides	"User logs into their account"
Test Scenario	High-level idea of what to test	"Verify login with valid credentials"
Test Case	Detailed steps with input, output, expected result, etc.	Step 1: Enter username → Step 2: Click login

📝 Relationship:

- 1 Use Case → Multiple Scenarios
- 1 Scenario → Multiple Test Cases

28. Test Environment and Execution

🔧 Test Environment:

◇ The **setup of hardware, software, network** and tools used for testing.

📁 Includes:

- ◇ Server setup (e.g., staging)
- ◇ OS version
- ◇ Database setup
- ◇ Browser/device compatibility
- ◇ Tools like Selenium, JIRA, Postman, etc.

←
END Test Execution:

- ◇ Running test cases in the prepared environment.
- ◇ **Result = Pass / Fail**
- ◇ Bugs reported during execution → tracked using tools like **JIRA** or **Bugzilla**
- 🧠 Tip: A well-configured test environment is critical for **reliable** and **repeatable** testing.

30: Test Closure

◇ What is Test Closure?

Test Closure is the **final phase** of the **Software Testing Life Cycle (STLC)**. It marks the **formal end of testing** once all planned tests have been executed and defects are addressed.

🎯 Goals of Test Closure:

- ☒ Confirm that testing is complete.
- ☒ Verify all planned test cases are executed.
- ☒ Ensure all critical bugs are fixed/closed.
- ☒ Prepare final documentation and reports.
- ☒ Identify lessons learned for future projects.

📄 Activities During Test Closure:

Step	Description
1. Test Summary Report	A document showing testing activities and outcomes
2. Test Case Status Report	List of how many test cases passed, failed, skipped

Step	Description
3. Bug Report Summary	Number of bugs found, fixed, open, deferred
4. Lessons Learned	What went well, what didn't, how to improve next time
5. Sign-Off	Final approval by stakeholders that testing is complete

Example:

You test a login page:

- ◇ Planned 10 test cases → 9 Passed, 1 Failed and fixed
- ◇ Created 5 bugs → All fixed
- ◇ You write a **Test Closure Report** with these stats
- ◇ You note that test environment setup took longer – improvement area



Topic 31: **Test Metrics**

- ◇ What are Test Metrics?

Test Metrics are **measurable values** used to **track the quality, progress, and efficiency** of the software testing process.

They help stakeholders make **data-driven decisions** about the release and quality.

Types of Test Metrics:

Metric Name	Description	Example
 Test Case Execution Rate	% of test cases executed vs. total planned	90% executed = Good Progress
 Defect Density	Number of defects per module/ lines of code	5 bugs in 1000 LOC = 0.005

Metric Name	Description	Example
⚠ Defect Leakage	Defects missed during testing but found in production	2 critical bugs found after release
🕒 Test Effort Variance	Actual vs. estimated time/effort for testing	Planned 30 hrs, took 40 hrs → +10
✅ Pass/Fail Rate	% of passed vs. failed test cases	80% Pass – 20% Fail

📝 Why Test Metrics Matter?

- ◇ Measure **testing effectiveness**
- ◇ Identify **weak areas** or modules
- ◇ Support decisions like **release readiness**
- ◇ Improve **future planning and estimation**

🧠 Real-World Use:

Let's say you're testing an e-commerce site:

- ◇ 100 test cases planned
 - ◇ 95 executed → Execution Rate = 95%
 - ◇ 10 bugs found → Defect Density
 - ◇ After release, 3 more bugs found → Defect Leakage
- This data goes into **Test Metrics Report**

Basic Topics

Basics Topics of QA Enginner

◇ **1. Test Scenario**

Definition:

Test scenario aik high-level idea ya condition hoti hai jo check karni hoti hai. Yeh batata hai ke hum kis functionality ya feature ko test karne wale hain.

Purpose:

Yeh help karta hai testers ko samajhnay mein ke system ka kaunsa part check karna hai. Bohat basic level per hota hai, steps detailed nahi hotay.

Example:

"Verify that user can login with valid credentials."

Yeh ek test scenario hai. Is mein abhi steps nahi diye, sirf functionality batayi gayi hai.

Use Case:

Jab aap planning kar rahay ho test coverage ka, to test scenarios banate ho taake overall idea mile ke kya-kya test karna hai.

◇ **2. Test Case**

Definition:

Test case woh detailed instructions hoti hain jinke through hum ek specific functionality test karte hain. Ismein steps, input data, expected result sab hota hai.

Purpose:

Ensure karna ke system correct tareeke se kaam kar raha hai. Agar kuch galat hota hai, to test case fail ho jata hai.

Example:

Test Case for Login:

- Step 1: Open login page
- Step 2: Enter valid username
- Step 3: Enter valid password
- Step 4: Click login
- Expected Result: User should be redirected to dashboard

Use Case:

Har functionality ke multiple test cases banaye jaate hain taake different angles se test ho sake (positive, negative, edge cases, etc.)

◇ **3. Test Suite**

Definition:

Test suite ka matlab hota hai aik group ya collection of test cases jo kisi specific module ya feature ko test karne ke liye banaye gaye hoon.

Purpose:

Organize karna multiple test cases ko taake asani se execution aur tracking ho sake.

Example:

Login Module Test Suite may include:

- ◇ Login with valid credentials
- ◇ Login with invalid password
- ◇ Login with blank username
- ◇ Forgot password link working

Use Case:

Automation mein test suite chalaya jaata hai taake ek baar mein saaray related test cases run ho jaayein.

◇ **4. Test Data**

Definition:

Test data woh inputs hotay hain jo hum test cases run karte waqt system mein dalte hain.

Purpose:

System ki functionality test karne ke liye zaroori hota hai sahi aur relevant data use karna.

Example:

Test data for login:

Username: `uzair123`

Password: `Test@123`

Use Case:

Different types of test data use kiya jaata hai: valid, invalid, boundary values, etc. Yeh test case ke success ya failure ko determine karta hai.

◇ 5. Test Plan

Definition:

Test Plan aik official document hota hai jisme testing ki strategy, scope, tools, schedule, test cases, aur responsibilities likhe hotay hain.

Purpose:

Testing ko plan karna aur team ko direction dena ke testing kaise perform hogi, kab hogi, kis area ko test karna hai.

Example (Headings in a test plan):

- ◇ Test objectives
- ◇ Test environment
- ◇ Scope in/out
- ◇ Test deliverables
- ◇ Schedule
- ◇ Roles & responsibilities

Use Case:

Testing start karne se pehle Test Plan ready hota hai, jis par project manager aur QA lead dono kaam karte hain.

◇ 6. RTM – Requirement Traceability Matrix

Definition:

RTM ka full form hota hai **Requirement Traceability Matrix**. Yeh aik table ya sheet hoti hai jisme hum **requirements** ko unke related **test cases** se map karte hain. Matlab: har requirement k test case banay hain ya nahi — yeh RTM se pata chalta hai.

Purpose:

RTM ensure karta hai ke koi bhi requirement miss na ho testing mein. Har requirement ka test case hona chahiye — yeh verify karne ke liye RTM use hota hai.

Example:

Requirement ID	Description	Test Case ID	Status
REQ-001	User Login functionality	TC-001, TC-002	Covered
REQ-002	Forgot Password feature	TC-003	Covered

Use Case:

Jab client ya manager check karta hai ke saari requirements properly test hui hain ya nahi — tab RTM use hoti hai. Also helpful in audits.

◇ 7. Smoke Testing

Definition:

Smoke testing aik short set hota hai basic test cases ka jo software ke **core features** ko check karta hai. Isay **build verification testing** bhi kehte hain.

Purpose:

Dekha jata hai ke software build stable hai ya nahi. Agar smoke test pass ho jaye to hum aagay ki testing start karte hain.

Example:

- Application open ho rahi hai?

- Login page load ho raha hai?
- Dashboard aa raha hai after login?

Use Case:

Daily build milta hai developers se, QA pehle smoke test karta hai — agar fail ho jaye to aagay testing nahi hoti. Sirf major functions check kiye jaate hain.

◇ 8. Sanity Testing

Definition:

Sanity testing hoti hai **narrow and deep testing**. Jab koi specific functionality mein change hota hai, to usi functionality ko test karte hain — yehi sanity testing hoti hai.

Purpose:

Verify karna ke naye change ke baad functionality sahi kaam kar rahi hai ya nahi — bina pura system test kiye.

Example:

Developer ne password reset ka code fix kiya. Ab QA sirf password reset functionality ko test karega (na ke pura login module).

Use Case:

Fast feedback ke liye hota hai. Jab frequent bug fixes ho rahe hoon, to har dafa full regression test karna mushkil hota hai — us waqt sanity testing hoti hai.

◇ 9. Regression Testing

Definition:

Regression testing ka matlab hota hai purani functionalities ko dobara test karna jab bhi naya feature add ho ya bug fix ho — taake ensure ho ke naye change ne koi purani cheez to nahi tor di.

Purpose:

Guarantee karna ke naye changes ki wajah se existing system affect nahi hua.

Example:

Agar developer ne profile update ka feature add kiya hai — QA team dobara login, dashboard, logout functions ko bhi test karegi to make sure sab kaam theek hai.

Use Case:

Large scale projects mein jab multiple modules interlinked hoon — har choti change ke baad regression testing zaroori hoti hai.

◇ 10. Positive & Negative Testing

Definition:

◇ **Positive testing:** Jab hum correct inputs ke sath test karte hain to verify ke system expected output deta hai.

◇ **Negative testing:** Jab hum galat ya unexpected input dete hain taake dekhein system error handle karta hai ya nahi.

Purpose:

System ki **reliability aur robustness** test karna — har situation mein sahi kaam kare.

Example:

Login Test:

◇ Positive: Username = uzair, Password = 123 → should login

◇ Negative: Username = blank, Password = 123 → should show error

Use Case:

Real-world users hamesha sahi input nahi daalte — is liye negative testing help karti hai ensure karne mein ke

app crash ya misbehave na kare.

◇ 11. Test Environment

Definition:

Test Environment aik combination hota hai hardware, software, server settings, aur configurations ka jahan pe QA team application ko test karti hai. Ye same hota hai ya similar hota hai production (real user) environment jesa.

Purpose:

Ensure karna ke app sahi tarah chal rahi hai real-world jesa environment mein. Bugs identify karne ke liye same setup chahiye hota hai jesa production mein hoga.

Example:

- Operating System: Windows 10
- Browser: Chrome 120
- Server: Apache
- Database: MySQL
- Application build version: 2.3.4

Use Case:

Jab kisi feature ka load ya compatibility test karna ho, to correct test environment zaroori hota hai. Misconfigured environment = misleading results.

◇ 12. Test Execution

Definition:

Test Execution ka matlab hai ke jo test cases banaye gaye hain unko actually run karna (manually ya automation se), aur check karna ke expected result milta hai ya nahi.

Purpose:

Dekha jata hai ke software sahi kaam kar raha hai ya nahi. Aur jo bugs milte hain unko defect report mein document kiya jata hai.

Example:

- ◇ Test Case: Login with valid credentials
- ◇ Action: Username = uzair, Password = 123456
- ◇ Expected: Dashboard load ho
- ◇ Actual: Dashboard load hogaya ☒ → Pass
Ya nahi hua **✗** → Fail

Use Case:

Internships mein jab kaha jata hai "run the test cases", to yehi Test Execution hoti hai.

◇ 13. Defect / Bug

Definition:

Jab software expected behavior se different kaam karta hai, ya koi error produce hoti hai, usay hum **defect** ya **bug** kehte hain. Developer ki coding mistake ki wajah se hota hai.

Purpose:

Bug report karna zaroori hota hai taake developer usay fix kare aur system reliable ho.

Example:

◇ Expected: Password field should accept only 8+ characters

◇ Actual: It accepts 3 characters too → 🐞 **Bug found**

Use Case:

Internship ya job mein QA engineers bug ko JIRA ya kisi bug tracking tool mein log karte hain.

◇ 14. Defect Life Cycle (Bug Life Cycle)

Definition:

Defect Life Cycle ya Bug Life Cycle aik **process** hoti hai jo batati hai ke jab bug report hota hai to uska kya-kya status hota hai jab tak wo fix ho kar close na ho jaye.

Main Stages:

1. **New** – Bug report kiya
2. **Assigned** – Developer ko diya
3. **Open** – Developer kaam start karta hai
4. **Fixed** – Developer ne fix kar diya
5. **Retest** – QA dobara test karta hai
6. **Closed** – Bug fix ho gaya
7. **Reopen** – Bug wapis aa gaya
8. **Deferred** – Fix baad mein hoga
9. **Rejected** – Developer mana kar diya

Use Case:

Bug tracking process ko standard banane ke liye defect life cycle follow ki jati hai. Interviews mein yeh important concept hota hai.

◇ 15. Severity vs Priority

Definition:

◇ **Severity** = Kitna serious hai bug system ke liye (Technical impact)

◇ **Priority** = Kitni jaldi fix karna chahiye (Business importance)

Example:

Bug Type	Severity	Priority
Crash on login	High	High
Typo in Terms & Conditions	Low	High

Bug Type	Severity	Priority
Error in admin panel only	High	Low

Use Case:

Bug report karte waqt QA ko dono fields fill karni hoti hain — severity developer ke liye aur priority product owner ke liye hoti hai.

Shabash Uzair! Ab chaltay hain aglay **QA Basic Terms (16 to 20)** ki taraf — full **Urdu-style English** explanation ke sath jaise pehlay kiya tha — taake aapki **notes aur concept dono strong** ho jaayein.

◇ 16. Smoke Testing

Definition:

Smoke Testing aik **basic level testing** hoti hai jisme QA team verify karti hai ke application ka main functionality chal raha hai ya nahi — taake pata lag sake ke system testing ke laayak hai ya nahi.

Purpose:

Check karna ke build install hone ke baad **basic features** sahi chal rahe hain. Agar smoke fail ho gaya to aagay testing ka faida nahi.

Example:

• App launch ho rahi hai? ☒

• Login kaam kar raha hai? ☒

• Dashboard khul raha hai? ☒

Ye sab pass → build is stable

Agar login crash kare → **✗** build reject

Use Case:

Naya software build milte hi QA pehle smoke test karta hai — isay **"build verification testing"** bhi kehte hain.

◇ 17. Sanity Testing

Definition:

Sanity Testing hoti hai **narrow and deep testing** jab koi choti fix ya update hui ho, aur usi specific feature ko check kiya jata hai ke wo ab sahi kaam kar raha hai ya nahi.

Purpose:

Ensure karna ke koi naya bug introduce nahi hua aur jo fix hua hai wo theek hai — without doing full regression testing.

Example:

Developer ne "Forgot Password" bug fix kiya → QA sirf yehi functionality test karega (login, reset, email link)

Agar sab sahi ho gaya ☒ → build is good

Use Case:

Agar short time ho aur quick check karna ho ke bug fix hua ya nahi → Sanity Test used.

◇ 18. Regression Testing

Definition:

Regression Testing ka matlab hai ke jab koi bug fix hota hai ya feature add hota hai to **purani functionality** ko dobara test karna taake pata chale ke kahin aur koi bug to introduce nahi ho gaya.

Purpose:

Ensure karna ke naye changes puranay features ko **toot** na dein.

Example:

Aapne checkout page ka button style change kiya

→ Ab test karna ho ga ke

- ◇ Payment kaam kar raha hai?
- ◇ Order place ho raha hai?
- ◇ Confirmation email aa rahi hai?

Use Case:

Har naye build mein regression test cases chalayе jaate hain — automation tools bhi use hotay hain ismein.

◇ 19. Static Testing

Definition:

Static Testing ka matlab hai software ko **run kiye bagair hi test karna**. Sirf requirements, code, design documents, test cases ko review karna aur issues identify karna.

Purpose:

Errors ko early detect karna before software chalayе — cost aur time dono save karne ke liye.

Example:

- ◇ Requirement document read karke unclear ya missing part identify karna
- ◇ Test cases review karna
- ◇ Code review (QA + developer milkar)

Use Case:

Software life cycle ke early phase mein static testing hoti hai — jisme meetings aur walkthroughs hoti hain.

◇ 20. Dynamic Testing

Definition:

Dynamic Testing ka matlab hai ke actual software ko **run karke test karna** — aur dekha jata hai ke expected result mil raha hai ya nahi.

Purpose:

Application ki behavior aur output ko real time mein verify karna by executing code.

Example:

Login page pe invalid credentials daal ke dekhna

→ Error aata hai ya nahi?

→ Password validation kaam karta hai?

Use Case:

Manual aur automation testing dono dynamic testing ke examples hain. Internship aur job mein mostly dynamic testing hoti hai.

☒ Recap in Easy Words:

Term	Real Meaning
Smoke Testing	Build stable hai ya nahi (basic check)
Sanity Testing	Specific fix test (quick deep check)

Term	Real Meaning
Regression Testing	Old features break to nahi huay?
Static Testing	Test without running code
Dynamic Testing	Run karke output check karna

◇ 21. Black Box Testing

Definition:

Black Box Testing mein tester ko **internal code ya structure** ka koi idea nahi hota. Sirf input dete hain aur output check karte hain — andar system kaise kaam kar raha hai, wo nahi dekhte.

Purpose:

End-user ki tarah test karna — sirf functionality ko verify karna.

Example:

Agar login form test karna hai to:

- Correct username/password do
- Wrong password try karo
- Empty fields check karo

Lekin **code kaise likha hai**, us ka pata nahi.

Use Case:

Manual testing, UI testing, and functional testing mostly black box hoti hain.

◇ 22. White Box Testing

Definition:

White Box Testing mein tester ko **poora access hota hai code ka**. Wo logic, loops, conditions sab check karta hai.

Purpose:

Ensure karna ke har line of code properly kaam kar rahi hai.

Example:

- ◇ If-else condition chal rahi hai ya nahi?
- ◇ Loops infinite to nahi chal rahe?
- ◇ Function ka output sahi hai?

Use Case:

Mostly developers karte hain ya automation testers — isay **clear box testing** bhi kehte hain.

◇ 23. Alpha Testing

Definition:

Alpha Testing hoti hai jab product **development environment** mein hi company ke andar QA team ya internal users use karke test karte hain.

Purpose:

Final bugs ya feedback lena before public release.

Example:

Software abhi officially release nahi hua, lekin company ke employees usay test kar rahe hain.

Use Case:

Software release se pehle internal feedback aur bug fix ka stage.

◇ **24. Beta Testing****Definition:**

Beta Testing hoti hai jab product almost ready hota hai aur company **limited real users** ko release karke unka feedback leti hai.

Purpose:

Real world mein software kaise perform karta hai, wo dekhna.

Example:

Facebook ya Instagram ka new feature selected users ko milta hai pehle

→ Ye beta testing hoti hai.

Use Case:

Market release se pehle customer feedback lena, bugs report karwana.

◇ **25. Boundary Value Analysis (BVA)****Definition:**

BVA ek **black box test technique** hai jisme hum input values ke **boundaries pe test karte hain**, kyunki wahaan bug aane ka chance zyada hota hai.

Purpose:

Find edge case errors — jahan system fail hone ka chance hota hai.

Example:

Agar age input 18 to 60 allowed hai:

Test Cases:

- ◇ 17 (invalid)
- ◇ 18 (valid - lower boundary)
- ◇ 60 (valid - upper boundary)
- ◇ 61 (invalid)

Use Case:

Jab bhi range ya limit wale input hoon (salary, age, price), BVA lagbhag har testing mein use hota hai.

☒ **Quick Summary Table:**

Term	Easy Meaning
Black Box	Code nahi dekhna, sirf input/output
White Box	Code ka andar ka logic bhi check karna
Alpha Testing	Company ke andar testing
Beta Testing	Real users ke sath limited testing

Term	Easy Meaning
BVA	Limit values test karna (min/max)

Shabash Uzair bhai! 🙌 Chaltay hain aakhri **QA Basics Terms 26–30** ki taraf — **full details, urdu-style English, with purpose, examples, and use cases** taake aapki notebook full tayar ho jaye! 📖 ✍️

◇ 26. Equivalence Partitioning

Definition:

Ye ek **black box test technique** hai jisme input values ko **groups/partitions** mein divide kiya jata hai — aur har group ka sirf **ek hi value** test ki jati hai.

Purpose:

Jab input ka range bada ho, to har value test karna mushkil hota hai. Is technique se **kam test cases** mein bhi coverage mil jati hai.

Example:

Agar input age 18–60 valid hai:

- Valid Partition: 18–60 (Test: 30)
- Invalid Partition 1: <18 (Test: 16)
- Invalid Partition 2: >60 (Test: 65)

Sirf har partition ka **ek test case** banayenge — sab check nahi karte.

Use Case:

Jab form fields mein **range based** ya **categorical** inputs hoon — jaise: Age, Grade, Salary etc.

◇ 27. Usability Testing

Definition:

Usability Testing mein software ko **real users ki tarah use** karke dekha jata hai ke kya wo asaani se samajh aata hai, use karna simple hai ya nahi.

Purpose:

Ensure karna ke product **user-friendly** ho, navigation easy ho, aur user confuse na ho.

Example:

Agar ek app ka button "Sign Up" ki jagah pe chhupa hua ho ya difficult language likhi ho → User ko problem hogi.

Use Case:

Mobile apps, websites, UI/UX design mein usability testing bohat important hai.

◇ 28. Sanity Testing

Definition:

Sanity Testing hoti hai jab software mein koi **minor change** kiya jata hai aur tester quickly check karta hai ke **ky-a feature kaam kar raha hai ya nahi**.

Purpose:

Verify karna ke fix ya update ke baad software still stable hai — **deep testing nahi karte**.

Example:

Login page ka "Remember Me" option fix kiya gaya — ab tester sirf wo feature aur basic login ko test karega.

Use Case:

Jab development mein choti updates aayein aur full regression test ka time na ho.

◇ 29. Smoke Testing

Definition:

Smoke Testing hoti hai jab software ka **basic functionality** test ki jati hai jaise ke wo **boot ho raha hai ya crash to nahi kar raha**.

Purpose:

Dekhna ke system major functions run karte hain ya nahi — agar fail ho to aage testing ka faida nahi.

Example:

- ◇ App open hoti hai ya crash hoti hai?
- ◇ Home page load ho raha hai?
- ◇ Main menu kaam kar raha hai?

Use Case:

Build deploy hone ke baad pehla test — isay “build verification testing” bhi kehte hain.

◇ **30. Ad-Hoc Testing**

Definition:

Ad-Hoc Testing hoti hai jab tester **bina kisi test case/documentation ke**, bas apne **experience aur gut feeling** se software test karta hai.

Purpose:

Aise bugs dhoondhna jo documents mein define nahi hotay, aur unexpected areas mein ho sakte hain.

Example:

Tester randomly 10 times submit button dabaye ya alag inputs try kare bina test case follow kiye.

Use Case:

Exploratory testing mein use hota hai ya jab formal process ka time na ho.

☑ **Final Summary Table (26–30):**

Term	Easy Meaning
Equivalence Partitioning	Input groups banao, har group ka 1 test
Usability Testing	Dekhna ke user easily app use kar sakta hai ya nahi
Sanity Testing	Minor fix ke baad quick feature test
Smoke Testing	System ki basic check — app chal rahi hai ya crash
Ad-Hoc Testing	Random aur informal testing — no test case follow

- ◇ Manual Testing Lab Exercises
- ◇ Test Case Writing Practice
- ◇ Jira Hands-on Guide
- ◇ Agile Project QA Flow

Module 2

Module 2: Software Testing Project.

1: Project Introduction

- ◇ What is a Software Testing Project?

A **software testing project** is a **real or simulated project** where testers:

- Understand software requirements.
- Write and run test cases.
- Report bugs.
- Ensure the application works as expected.

 Think of this as the **practical side of software testing** — where you apply everything you've learned in Module 1.

Example:

Suppose you're testing an **Online Food Ordering App**.

- ◇ User should be able to sign up, browse food, and place orders.
- ◇ Your project is to test every feature and find bugs before release.

2: Understanding Functional Requirements from FRS

- ◇ What is FRS?

FRS = Functional Requirement Specification

It is a document that describes:

- ◇ What features the software **must do**.
- ◇ How the system should behave for **each functionality**.

What is included in FRS?

Section	Description
Feature Name	e.g., Login page
Input/Output	Email, password → Success or Error
Validations	Email format, required password
User Roles	Admin, customer, etc.
Business Rules	Password must be at least 8 characters

💡 Why FRS is important?

You cannot test what you don't understand!

FRS helps you design test cases based on **what the system is expected to do**.

✓ 3: Creating Test Scenarios

◇ What is a Test Scenario?

A **Test Scenario** is a **high-level idea** of what to test.

It answers: *"What will I test?"*

📄 Example:

Feature	Test Scenario
Login	Verify login with valid credentials
Login	Verify login with wrong password
Signup	Check signup form field validations

💡 Tips:

- ◇ Scenarios are broader than test cases.
- ◇ Don't add steps or expected results here.

- ◇ Cover **positive** and **negative** conditions.

✓ 4: Creating Test Cases

- ◇ What is a Test Case?

A **Test Case** is a **step-by-step guide** to test a specific functionality.

It includes: What to test, how to test, what to expect.

📄 Format of Test Case:

Field	Example
Test Case ID	TC_Login_01
Description	Verify login with valid email & password
Pre-Conditions	User must be registered
Steps to Execute	1. Go to login page → 2. Enter details → Click
Expected Result	User is logged in successfully
Actual Result	(To be filled during execution)
Status	Pass / Fail

🤔 Why Test Cases Matter?

- ◇ They **ensure complete coverage**.
- ◇ Help testers avoid missing any steps.
- ◇ Serve as **evidence** in case of bugs or disputes.

Perfect, Uzair! Let's continue with the **remaining topics of Module 2: Software Testing Project** in the same detailed and beginner-friendly style.

✓ Topic 5: Test Execution

- ◇ What is Test Execution?

Test execution is the phase where you **run the test cases** on the application to check if it works as expected.

- 🔧 You're now doing real testing on the application — either manually or using automation tools.

⚙️ Steps in Test Execution:

1. Open the application.
2. Perform steps from test cases.
3. Compare actual result with expected result.
4. Mark the result: ☒ Pass / ☒ Fail.
5. Report bugs if any step fails.

Example:

You have this test case:

Step	Action	Expected Result
1	Go to login page	Login page opens
2	Enter valid email/ password	Redirected to dashboard

 If Step 2 fails → ☒ Mark "Fail" →  Report a bug.

☒ Topic 6: **Bug Reporting & Tracking**

◇ What is Bug Reporting?

When you find an issue during testing, you **report it to developers** using a tool like **JIRA** or **Bugzilla**.

Bug Report Template:

Field	Description
Bug ID	Unique ID (e.g., BUG-1234)
Title	Short description (e.g., "Login fails on Chrome")
Steps to Reproduce	Clear steps to see the bug
Expected Result	What should happen
Actual Result	What happened instead

Field	Description
Severity	How serious is the bug? (High, Medium, Low)
Priority	How urgent is it to fix?
Attachments	Screenshots, logs, etc.

🧠 Bug Tracking Tools:

- **JIRA**
- **Bugzilla**
- **Redmine**
- **Mantis**

☑ Topic 7: **Test Sign Off**

◇ What is Test Sign Off?

Test Sign Off is the **final approval** by QA that testing is completed and the product is ready for release. It means: "We have tested everything. No major bugs are left. It's good to go live."

☑ Sign Off Criteria:

- ◇ All planned test cases executed.
- ◇ All critical bugs fixed and retested.
- ◇ Regression testing completed.
- ◇ Test summary prepared.

📊 Test Summary Report Includes:

- ◇ Total test cases run
- ◇ Passed / Failed count
- ◇ Bug status
- ◇ Pending issues (if any)
- ◇ QA team recommendation

🧠 Why is it important?

- ◇ It gives confidence to the product team.
- ◇ Shows that testing was properly done.
- ◇ Acts as a document of record.

Module 3

Module 3: Agile Testing

1: What is Agile?

◇ Definition:

Agile is a software development methodology that focuses on **continuous delivery**, **customer feedback**, and **quick iterations** (called sprints). It breaks down large projects into **small, manageable pieces**.

 Core Ideas of Agile:

- Work is divided into **small increments**.
- Continuous communication with stakeholders.
- Working product delivered at the end of each **sprint**.
- Responds well to **changing requirements**.

 Agile Manifesto (4 Key Values):

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan


2: What is Scrum / Scrum Team?

◇ What is Scrum?

Scrum is the **most popular Agile framework** that defines **how teams should work** together in Agile. It uses **roles, ceremonies, and artifacts** to manage work.

 Scrum Team Roles:

1. **Product Owner (PO)**: Owns the product backlog and defines features.
2. **Scrum Master**: Facilitates the process and removes obstacles.
3. **Development Team**: Developers + QA testers + Designers who build and test the product.

 Scrum is iterative:

Each iteration is called a **Sprint** (usually 1–4 weeks), and at the end of each sprint, a working product is delivered.

3: What is Sprint?

◇ Definition:

A **Sprint** is a **fixed time period** (usually 2 weeks) in which a team works on a defined set of tasks or features and delivers a working software.

 Sprint Cycle Includes:

- ◇ Sprint Planning
- ◇ Daily Standup (15 min daily meeting)
- ◇ Sprint Execution
- ◇ Sprint Review (demo)

- ◇ Sprint Retrospective (feedback)

Benefits:

- ◇ Fast delivery of usable features
- ◇ Regular feedback
- ◇ Encourages team collaboration

☒ 4: What is a User Story?

- ◇ Definition:

A **User Story** is a **short, simple description** of a software feature written from the perspective of the end user.

Format:

As a `<type of user>`, I want `<some goal>` so that `<some reason>`.

- ☒ Example:

As a **registered user**, I want to **reset my password** so that I can **regain access to my account**.

Characteristics of a good user story: (*INVEST*)

- ◇ **I**ndependent
- ◇ **N**egotiable
- ◇ **V**aluable
- ◇ **E**stimable
- ◇ **S**mall
- ◇ **T**estable


☒ 5: How to Give Story Points / Estimate User Story

- ◇ What are Story Points?

Story Points are **units used to estimate the effort** needed to complete a user story (not in hours/days, but complexity-wise).

Example Pointing Scale:

- ◇ 1 → Very Easy
- ◇ 2 → Easy
- ◇ 3 → Medium
- ◇ 5 → Complex
- ◇ 8 → Very Complex
- ◇ 13 → Risky or uncertain

 **Fibonacci series** (1, 2, 3, 5, 8, 13...) is commonly used.

Who gives the estimates?

- ◇ Entire **scrum team**, including **QA testers**, gives points in a meeting called **Planning Poker**.

☒ 6: What is Definition of Done (DoD) and Definition of Ready (DoR)?

◇ Definition of Done (DoD):

It is a **set of criteria** that a **user story must meet** to be considered **complete**.

☑ Example DoD:

- Code is written
- Code is reviewed
- Functionality is tested (unit + QA testing)
- No major bugs remain
- Deployed on staging
- Accepted by Product Owner

➡ Ensures **quality and completeness**.

◇ Definition of Ready (DoR):

It is a **checklist** that confirms a user story is **clear and ready to be picked up** by the team in the sprint.

☑ Example DoR:

- ◇ User story is well-written and understood
- ◇ Acceptance criteria is clear
- ◇ No dependency on other tasks
- ◇ Estimated by team

➡ Ensures **clarity and preparedness**.

☑ 7: **Sprint Planning**

◇ What is it?

A **Sprint Planning meeting** is held at the **start of a sprint**, where the team:

- ◇ Selects **what work** will be done
- ◇ Breaks tasks into smaller subtasks
- ◇ Estimates and assigns work

🕒 Timeboxed to:

- ◇ 2 hours for a 1-week sprint
- ◇ 4 hours for a 2-week sprint

◇ Who Attends:

- ◇ Scrum Master
- ◇ Product Owner
- ◇ Scrum Team (Dev + QA)

☑ 8: **Backlog Refinement**

◇ What is it?

A **regular session** where the team **reviews, discusses, and improves user stories** in the **Product Backlog**.

📅 Activities include:

- ◇ Adding new user stories

- ◇ Clarifying stories
- ◇ Estimating effort
- ◇ Splitting large stories into smaller ones

🕒 Usually done **once per week**

☑ 9: **Sprint Review**

- ◇ What is it?

A **demo session** held **at the end of the sprint**, where the team:

- ◇ **Shows what was completed**
- ◇ Gathers **feedback from stakeholders**

🎯 Goal: Present **working features** to the Product Owner and stakeholders.

🧠 Purpose:

- ◇ Align expectations
- ◇ Confirm acceptance or rejection of stories

☑ 10: **Sprint Retrospective**

- ◇ What is it?

A **reflection meeting** held **after the Sprint Review** to discuss:

- ◇ What went well?
- ◇ What didn't go well?
- ◇ What can be improved in the next sprint?

🔗 Goal: Continuous improvement of the team.

JIRA

JIRA Tool.

☑ 1: **How to Install and Configure JIRA Tool**

- ◇ What is JIRA?

JIRA is a popular **project management and issue tracking tool** used for:

- Managing agile projects (Scrum, Kanban)
- Tracking bugs
- Writing test cases (with Zephyr plugin)

- ◇ How to Install:

1. Go to: <https://www.atlassian.com/software/jira>
2. Click on **"Try it free"** (Cloud version)
3. Create an Atlassian account

4. Choose **"JIRA Software"**

5. Set up your project (choose **Scrum** or **Kanban**)

➡ No local installation needed — it's **cloud-based**.

☑ 2: How to Create an EPIC / User Stories in JIRA

◇ EPIC:

An **EPIC** is a large feature or requirement that can be broken down into smaller **User Stories**.

◇ User Story:

A **small, self-contained unit** of work written from the user's perspective.

📄 Steps to Create:

1. Open your Jira project
2. Click "Create"
3. Choose Issue Type: **Epic** or **Story**
4. Enter:

- ◇ Title
- ◇ Description
- ◇ Priority
- ◇ Assignee

• Click "Create"

🗣️ Example User Story:

As a user, I want to **reset my password** so that I can **regain access** to my account.

☑ 3: Creating Sprints in JIRA

◇ Sprint:

A **timeboxed cycle** (usually 1 or 2 weeks) where work is done on selected stories.

📄 How to Create:

1. Go to **Backlog** section
2. Click **"Create Sprint"**
3. Add issues (User Stories) to the sprint
4. Set **Start** and **End Date**
5. Click **Start Sprint**

🔗 Once started, the sprint board shows "To Do", "In Progress", and "Done" columns.

☑ 4: Sprint Life Cycle in JIRA

🌀 Sprint Lifecycle Includes:

1. **Sprint Planning** → Add issues to Sprint
2. **Active Sprint** → Team works on tasks
3. **Daily Standups** → Track progress
4. **Sprint Review** → Demo completed features
5. **Sprint Retrospective** → Improve processes

6. **Close Sprint** → End sprint in JIRA

🔗 All activities are tracked using **boards** and **burndown charts** in JIRA.

✓ 5: Backlogs in JIRA

◇ What is a Backlog?

A **list of all tasks, bugs, and user stories** that need to be worked on in the future.

📖 In JIRA:

- ◇ Found under the **Backlog tab**
- ◇ Can include Epics, Stories, Tasks, and Bugs
- ◇ Prioritized by the **Product Owner**
- ◇ Team pulls from it during **Sprint Planning**

✓ You can reorder tasks via drag-and-drop.

Perfect, Uzair! Let's now complete the final topics of **Module 3: Agile Testing + JIRA Tool** with detailed explanations for your QA notes:

✓ 6: Creating Bugs in JIRA

◇ What is a Bug?

A **bug** (or defect) is an error or flaw in software that causes it to behave unexpectedly.

📖 How to Log a Bug in JIRA:

1. Click on **"Create"** on the top menu
 2. Set **Issue Type** as **Bug**
 3. Fill in these fields:
 - **Summary:** Short title (e.g., *Login fails when credentials are valid*)
 - **Description:** Steps to reproduce, expected vs actual result
 - **Priority:** (High/Medium/Low)
 - **Assignee:** Tester or developer
 - **Environment:** OS, browser, version
- Attach screenshots or logs (optional but recommended)
 - Click **Create**

🔗 You can also link this bug to a specific test case or user story for traceability.

✓ 7: How to Write Test Cases in JIRA with Zephyr Plugin

◇ Zephyr for JIRA:

A test management plugin that allows testers to:

- ◇ Write test cases
- ◇ Organize them in cycles
- ◇ Track test execution

📖 Writing Test Cases:

1. Go to **Tests** → **Create a Test**
2. Fill in:

- ◇ **Summary:** Name of test case
- ◇ **Description:** What this test will verify
- ◇ **Test Steps:** Step-by-step instructions
- ◇ **Test Data:** (optional)
- ◇ **Expected Result**

- Save the test

🔗 You can assign test cases to a user and link them to stories or bugs.

☑ 8: Creating Test Cycles and Executing Test Cases in JIRA

- ◇ What is a Test Cycle?

A **test cycle** is a group of test cases executed together, often mapped to a sprint or release.

📄 Creating Test Cycles:

1. Go to **Tests** → **Plan Test Cycle**
2. Click on **Create New Cycle**
3. Name it (e.g., *Sprint 3 - Regression Cycle*)
4. Add:
 - ◇ Test cases (from Zephyr)
 - ◇ Build version
 - ◇ Environment

- Click **Create**

🔧 Executing Test Cases:

1. Select your **test cycle**
2. Choose a **test case**
3. Click **Execute**
4. Choose status:

- ◇ **PASS** / **FAIL** / **WIP** (Work in Progress)

- ◇ Add comments, attachments

- If failed, you can directly **raise a bug** and link it

🔗 This gives clear traceability between: Test → Execution → Bug