Complete 100+ Functional Testing Concepts - Comprehensive Interview Guide – Ranjit Appukutti

SECTION 1: FUNDAMENTALS OF FUNCTIONAL TESTING

1. What is Software Testing?

Definition: Software testing is the process of evaluating and verifying that a software application or system meets specified requirements, works as expected, and is free of defects.

Objectives:

- Find defects before users do
- Verify software meets requirements
- Build confidence in quality
- Prevent defects in production
- Reduce maintenance costs
- Ensure customer satisfaction

2. What is Functional Testing?

Definition: Functional testing validates the software system against functional requirements and specifications. It focuses on testing "what" the system does by verifying each function operates correctly.

Key Characteristics:

- Black-box testing approach
- Tests user-facing features
- Based on specifications
- Validates business logic
- Tests input/output behavior
- Independent of code structure

3. What is a Software Testing Life Cycle (STLC)?

Definition: STLC is a sequence of specific activities conducted during the testing process to ensure software quality goals are met.

Phases:

1. **Requirement Analysis:** Understanding testable requirements
2. **Test Planning:** Defining strategy, scope, resources, schedule

3. **Test Case Development:** Creating test cases, test data, test scripts
4. **Test Environment Setup:** Preparing hardware/software environment
5. **Test Execution:** Running test cases and logging defects
6. **Test Closure:** Analyzing results, documenting learnings

4. What is Test Strategy?

Definition: A high-level document defining the testing approach for the software development cycle. It defines what testing activities will be performed and how.

Components:

- Testing objectives
- Scope of testing
- Testing approach
- Resource planning
- Test environment requirements
- Risk assessment
- Defect tracking approach
- Entry/exit criteria

5. What is Test Plan?

Definition: A detailed document describing the scope, approach, resources, and schedule of testing activities. It's a tactical document derived from the test strategy.

Contents:

- Test plan identifier
- Introduction and scope
- Test items
- Features to be tested/not tested
- Testing approach
- Pass/fail criteria
- Suspension/resumption criteria
- Test deliverables
- Testing tasks
- Environmental needs
- Responsibilities
- Staffing and training

- Schedule
- Risks and contingencies
- Approvals

6. What is the difference between Test Plan and Test Strategy?

Test Strategy:

- High-level document
- Created by project manager/test manager
- Defines overall approach
- One strategy for entire organization/project
- Long-term perspective
- Defines what types of testing needed

Test Plan:

- Low-level, detailed document
- Created by test lead/test manager
- Defines specific testing activities
- Multiple plans for different modules
- Short-term perspective
- Defines how testing will be done

7. What is a Test Scenario?

Definition: A high-level statement describing end-to-end functionality to be validated. It's derived from use cases and represents a possible test situation.

Example: Test Scenario: Verify user registration functionality

This includes:

- Register with valid details
- Register with existing email
- Register with invalid data
- Register without mandatory fields

Characteristics:

- One-liner statement
- Covers multiple test cases
- Describes "what" to test

- High-level view
- Helps in test coverage

8. What is a Test Case?

Definition: A detailed set of conditions, inputs, actions, and expected results developed to verify a specific requirement or functionality.

Test Case Template:

Test Case ID: TC_001

Test Case Title: Verify login with valid credentials

Module: Authentication

Priority: High

Severity: Critical

Pre-conditions: User should be registered

Test Steps:

1. Navigate to login page
2. Enter valid username
3. Enter valid password
4. Click login button

Test Data: Username: test@email.com, Password: Test@123

Expected Result: User successfully logged in and redirected to dashboard

Actual Result: [To be filled during execution]

Status: [Pass/Fail/Blocked]

Post-conditions: User session created

Created By: John Doe

Created Date: 01/12/2024

9. What makes a good Test Case?

Characteristics:

- **Accurate:** Tests what it's supposed to test
- **Economical:** No unnecessary steps
- **Traceable:** Linked to requirements
- **Repeatable:** Consistent results
- **Reusable:** Can be used for regression

- **Simple:** Easy to understand and execute
- **Complete:** All necessary information included
- **Independent:** Not dependent on other test cases
- **Specific:** Clear expected results

10. What is Test Data?

Definition: Data created or selected to satisfy test case execution conditions. It includes both valid and invalid data sets.

Types:

- **Valid Data:** Acceptable inputs (positive testing)
- **Invalid Data:** Unacceptable inputs (negative testing)
- **Boundary Data:** Values at edges of input domains
- **Random Data:** Arbitrary values for exploratory testing
- **Production-like Data:** Masked/anonymized real data

Best Practices:

- Maintain data privacy (mask sensitive information)
- Create reusable datasets
- Document data requirements
- Store data separately from test cases
- Version control test data
- Use data generation tools

11. What is Test Execution?

Definition: The process of running test cases on the application under test and comparing actual results with expected results.

Activities:

- Execute test cases as per schedule
- Document actual results
- Compare with expected results
- Log defects for failures
- Update test case status
- Retest fixed defects
- Track test progress
- Report test metrics

12. What is a Test Suite?

Definition: A collection of test cases grouped together for execution. Test cases are typically grouped by feature, functionality, or test type.

Examples:

- Login Test Suite: All login-related test cases
- Regression Test Suite: Critical test cases for regression
- Smoke Test Suite: Basic functionality test cases
- Payment Test Suite: All payment-related test cases

Benefits:

- Organized test execution
- Easy test management
- Selective test execution
- Better tracking and reporting

13. What is Test Coverage?

Definition: A metric measuring the extent to which testing has been performed, expressed as a percentage.

Types:

Requirements Coverage: Formula: (Requirements tested / Total requirements) × 100

Code Coverage:

- Statement Coverage
- Branch Coverage
- Path Coverage
- Function Coverage

Test Case Coverage: Formula: (Test cases executed / Total test cases) × 100

Feature Coverage: Formula: (Features tested / Total features) × 100

14. What is Requirements Traceability Matrix (RTM)?

Definition: A document mapping requirements to test cases, ensuring complete test coverage and bidirectional traceability.

RTM Format:

Req ID | Requirement | Test Case ID | Test Status | Defect ID | Comments

R001 | User Login | TC_001 | Pass | - | -

| | TC_002 | Fail | BUG_045 | Invalid data

| | TC_003 | Pass | - | -

R002 | Registration | TC_004 | Pass | - | -

Types:

- **Forward Traceability:** Requirements → Test Cases
- **Backward Traceability:** Test Cases → Requirements
- **Bidirectional Traceability:** Both directions

Benefits:

- Ensures 100% test coverage
- Impact analysis for requirement changes
- Tracks test progress
- Identifies missing test cases
- Supports compliance and audit

15. What is Entry Criteria?

Definition: Conditions that must be satisfied before testing can begin.

Common Entry Criteria:

- Test plan approved
- Test environment ready and accessible
- Test data prepared
- Build deployed in test environment
- Smoke test passed
- Test cases reviewed and approved
- Resources available
- Tools and access permissions configured
- Defects from previous cycle fixed
- Requirements clearly defined

16. What is Exit Criteria?

Definition: Conditions that must be met to conclude testing phase and move forward.

Common Exit Criteria:

- All planned test cases executed
- Critical and high-priority defects fixed and verified
- Required test coverage achieved (e.g., 95%)

- No showstopper defects open
- Test summary report completed
- Regression testing completed
- Performance benchmarks met
- Sign-off from stakeholders
- All test deliverables submitted
- Risk assessment acceptable

17. What is Suspension Criteria?

Definition: Conditions under which testing will be temporarily halted.

Examples:

- Critical defects blocking testing
- Test environment unavailable
- Build unstable or not deployed properly
- Majority of test cases failing
- Critical hardware/software failure
- Key resources unavailable
- Major requirement changes

18. What is Resumption Criteria?

Definition: Conditions under which suspended testing can resume.

Examples:

- Critical defects fixed
- Environment restored and stable
- New stable build deployed
- Required resources available
- Hardware/software issues resolved
- Updated requirements provided

SECTION 2: TYPES OF FUNCTIONAL TESTING

19. What is Unit Testing?

Definition: Testing individual components or units of code in isolation to verify they work correctly. Smallest testable parts of an application.

Characteristics:

- Done by developers
- Tests individual functions/methods
- Uses white-box technique
- Performed during coding phase
- Automated using frameworks
- Fast execution
- First level of testing

Popular Tools:

- JUnit (Java)
- NUnit (.NET)
- PyTest (Python)
- Jasmine (JavaScript)
- Mocha (Node.js)

Example: Testing a function that calculates discount:

Function: calculateDiscount(price, percentage)

Test Cases:

- Valid inputs: calculateDiscount(100, 10) = 90

- Zero discount: calculateDiscount(100, 0) = 100

- Boundary: calculateDiscount(100, 100) = 0

- Invalid: calculateDiscount(-100, 10) = error

20. What is Integration Testing?

Definition: Testing the interfaces and interaction between integrated units/modules to identify interface defects.

Types:

Big Bang Integration:

- All modules integrated simultaneously
- Tested as a whole
- Difficult to isolate defects
- Used for small systems

Incremental Integration:

1. **Top-Down Approach:**
    - Testing from top to bottom
    - Main module tested first
    - Uses stubs for lower modules
    - Stubs: Dummy modules simulating called modules
2. **Bottom-Up Approach:**
    - Testing from bottom to top
    - Lower modules tested first
    - Uses drivers for higher modules
    - Drivers: Dummy modules that call tested modules
3. **Sandwich/Hybrid Approach:**
    - Combination of top-down and bottom-up
    - Middle-level modules tested with both stubs and drivers

Example: E-commerce system:

- User Module ↔ Shopping Cart Module
- Shopping Cart ↔ Payment Module
- Payment Module ↔ Inventory Module

21. What is System Testing?

Definition: Testing the complete integrated system to verify it meets specified requirements. Tests end-to-end system specifications.

Characteristics:

- Black-box testing
- Conducted in environment similar to production
- Functional and non-functional testing
- Done by testing team
- After integration testing
- Tests entire application flow

Types of System Testing:

- Functional testing
- Performance testing
- Usability testing

- Security testing
- Compatibility testing
- Recovery testing
- Documentation testing

22. What is Smoke Testing?

Definition: Preliminary testing to check whether critical functionalities work before detailed testing begins. Also called "Build Verification Testing."

Purpose:

- Verify build stability
- Check if build is testable
- Identify showstopper issues early
- Save testing effort on unstable builds

Characteristics:

- Wide and shallow testing
- Tests critical paths
- 20-30 minutes duration
- If failed, build rejected
- First test on new build
- Non-exhaustive testing
- Can be automated

Example Smoke Tests:

- Application launches successfully
- Login functionality works
- Main navigation accessible
- Critical APIs responding
- Database connectivity working

23. What is Sanity Testing?

Definition: Narrow and focused testing to verify specific functionality or bug fixes after minor code changes. Subset of regression testing.

Purpose:

- Verify bug fixes work
- Check specific functionality after minor changes

- Determine if detailed testing should proceed
- Quick validation of new functionality

Characteristics:

- Narrow and deep testing
- Unscripted, not documented
- Focuses on specific area
- After receiving software build with minor changes
- Usually not automated
- Rational check of application

Smoke vs Sanity:

Aspect	**Smoke Testing**	**Sanity Testing**
Scope	Wide and shallow	Narrow and deep
Purpose	Build stability	Specific functionality
Documentation	Documented	Not documented
Execution	Scripted	Unscripted
Subset of	Acceptance Testing	Regression Testing
Done by	Developers or Testers	Testers

24. What is Regression Testing?

Definition: Re-testing to verify that recent code changes haven't adversely affected existing functionality.

When to Perform:

- After bug fixes
- After new feature additions
- After code enhancements
- After configuration changes
- During maintenance releases
- After environment changes

Types:

1. Corrective Regression:

- No changes in specification

- Existing test cases reused
- Tests unaffected functionality

2. Retest-All Regression:

- All test cases in suite executed
- Time-consuming
- Ensures comprehensive testing
- Used when changes impact multiple areas

3. Selective Regression:

- Subset of test cases executed
- Only affected areas tested
- Optimized approach
- Analyzes impact of changes

4. Progressive Regression:

- New test cases for changes
- Modifications in specifications
- Tests new functionality integration

5. Complete Regression:

- Major changes in code
- Entire application tested
- Most comprehensive

6. Partial Regression:

- Code changes impact limited areas
- Tests affected modules only

Best Practices:

- Automate regression suite
- Prioritize test cases
- Maintain regression suite
- Select critical test cases
- Execute regularly
- Track regression metrics

25. What is Retesting?

Definition: Testing a specific defect that has been fixed to verify the fix works correctly. Also called "Confirmation Testing."

Process:

1. Defect logged by tester
2. Developer fixes defect
3. Build deployed with fix
4. Tester retests exact scenario
5. Verifies fix successful
6. Closes defect if passed

Retesting vs Regression:

Aspect	Retesting	Regression Testing
Purpose	Verify bug fix	Ensure no new defects
Scope	Specific defect	Entire application
Test Cases	Failed test cases	All or selected cases
Automation	Not initially	Can be automated
Priority	High	Medium to High
Execution	Defect-specific	Broader coverage

26. What is User Acceptance Testing (UAT)?

Definition: Final testing phase where actual business users validate the software meets business requirements and is ready for production.

Types:

1. Alpha Testing:

- Performed at developer's site
- Done by internal staff (not dev team)
- Simulated environment
- Early feedback
- Before beta testing

2. Beta Testing:

- Performed at client's site
- Done by end-users
- Real environment

- Limited user group
- Before final release
- Example: Beta versions of software

3. Contract Acceptance Testing:

- Based on contract acceptance criteria
- Verifies deliverables per contract
- Formal testing
- Legal implications

4. Regulation Acceptance Testing:

- Tests regulatory compliance
- Industry standards validation
- Legal/safety requirements
- Example: HIPAA, GDPR compliance

5. Operational Acceptance Testing:

- Tests operational readiness
- Backup/restore procedures
- Disaster recovery
- Maintenance procedures
- Security checks

UAT Process:

1. Business requirements analysis
2. UAT test plan creation
3. UAT test case design
4. Test data preparation
5. UAT execution
6. Defect logging and tracking
7. Sign-off

27. What is Exploratory Testing?

Definition: Simultaneous test design, execution, and learning where testers actively explore the application without predefined test cases.

Key Elements:

- Learning about the application
- Designing tests
- Executing tests
- Interpreting results

Characteristics:

- No predefined test cases
- Relies on tester's creativity
- Time-boxed sessions
- Freestyle testing
- Requires experienced testers
- Complements scripted testing
- Documents findings

Types:

1. Freestyle Exploratory:

- Complete freedom
- No structure
- Tester decides approach

2. Scenario-based Exploratory:

- Based on user scenarios
- Structured approach
- Real-world use cases

3. Strategy-based Exploratory:

- Follows specific strategy
- Boundary testing
- Risk-based testing

Session-Based Testing:

- Time-boxed sessions (60-90 minutes)
- Charter defines focus
- Notes taken during session
- Debriefing after session

Benefits:

- Finds unexpected defects
- Complements automated testing
- Quick feedback
- Tests real user scenarios
- Adapts to changes quickly

28. What is Ad-hoc Testing?

Definition: Informal, random testing without planning, documentation, or expected results. Goal is to break the system.

Characteristics:

- No formal process
- No test cases
- No documentation
- Random approach
- Cannot be repeated easily
- Based on intuition
- Also called "Monkey Testing"

Types:

1. Buddy Testing:

- Developer and tester work together
- Mutual understanding
- Immediate feedback

2. Pair Testing:

- Two testers work together
- One tests, one observes
- Knowledge sharing

3. Monkey Testing:

- Random inputs
- No rules
- Break the system
- Unusual scenarios

Ad-hoc vs Exploratory:

Aspect	Ad-hoc	Exploratory
Structure	No structure	Some structure
Approach	Random	Systematic
Documentation	None	Session notes
Knowledge	Any tester	Experienced tester
Purpose	Break system	Find defects strategically

29. What is End-to-End Testing?

Definition: Testing complete application flow from start to finish in real-world scenarios, including all integrated components, databases, networks, and external interfaces.

Objectives:

- Validate system flow
- Test data integrity
- Verify system dependencies
- Ensure sub-systems work together
- Test user experience
- Identify system-level issues

Example - E-commerce Application:

Complete Flow:

1. User registers on website
2. Browses product catalog
3. Searches for specific product
4. Adds product to cart
5. Updates cart quantities
6. Proceeds to checkout
7. Enters shipping address
8. Selects payment method
9. Makes payment
10. Receives order confirmation
11. Gets email notification
12. Tracks order status

13. Receives product

Components Tested:

- Frontend (UI)
- Backend (Application logic)
- Database
- APIs
- Third-party integrations
- Network
- Servers

Best Practices:

- Test most common user workflows
- Include both happy and unhappy paths
- Test across different environments
- Validate data flow
- Check error handling
- Verify notifications
- Test from user perspective

30. What is Black Box Testing?

Definition: Testing technique examining functionality without knowledge of internal code structure, implementation, or internal paths.

Characteristics:

- No code knowledge required
- Tests external behavior
- Based on requirements
- Functional testing approach
- Tests what system does
- User perspective

Techniques:

1. Equivalence Partitioning 2. Boundary Value Analysis 3. Decision Table Testing 4. State Transition Testing 5. Use Case Testing 6. Error Guessing

Levels:

- Unit testing (external behavior)

- Integration testing
- System testing
- Acceptance testing

Advantages:

- No programming knowledge needed
- Unbiased testing
- Test from user perspective
- Efficient for large systems
- Tests high-level functionality

Disadvantages:

- Limited coverage
- Cannot test code logic
- Difficult to identify hidden errors
- Test cases can be redundant
- Complex scenarios hard to test

31. What is White Box Testing?

Definition: Testing technique examining internal structure, design, and code. Tester has full knowledge of implementation.

Also Known As:

- Glass Box Testing
- Clear Box Testing
- Structural Testing
- Code-Based Testing
- Open Box Testing

Techniques:

1. Statement Coverage:

- Every statement executed at least once
- Formula: (Statements executed / Total statements) × 100

2. Branch Coverage:

- Every branch (if/else) tested
- Formula: (Branches executed / Total branches) × 100

3. Path Coverage:

- Every possible path tested
- Most comprehensive

4. Condition Coverage:

- Every condition tested for true/false
- Boolean expressions evaluated

5. Multiple Condition Coverage:

- All combinations of conditions

Example:

Code:

```
if (age > 18 && hasLicense) {
   allowDriving();
} else {
   denyDriving();
}
```

Test Cases Needed:

1. age > 18 && hasLicense = true (true branch)
2. age <= 18 && hasLicense = true (false branch)
3. age > 18 && hasLicense = false (false branch)
4. age <= 18 && hasLicense = false (false branch)

Advantages:

- Thorough testing
- Finds hidden errors
- Optimizes code
- Easy automation
- Early defect detection

Disadvantages:

- Requires programming knowledge
- Time-consuming
- Expensive

- Cannot find missing functionality
- Code changes require test updates

32. What is Grey Box Testing?

Definition: Combination of black box and white box testing where tester has partial knowledge of internal structure.

Characteristics:

- Limited code access
- Tests from user + developer view
- Focuses on integration points
- Database testing
- Architecture knowledge

When Used:

- Integration testing
- Web applications
- Database testing
- API testing
- Client-server applications

Techniques:

- Matrix testing
- Regression testing
- Pattern testing
- Orthogonal array testing

Advantages:

- Balanced approach
- Better quality
- Non-intrusive
- Intelligent test scenarios
- Developer-tester bridge

SECTION 3: BLACK BOX TESTING TECHNIQUES

33. What is Equivalence Partitioning?

Definition: Dividing input data into partitions where all values are expected to behave similarly. Test one value from each partition.

Concept: If one value in partition works, all values work. If one fails, all fail.

Steps:

1. Identify input domains
2. Divide into valid and invalid partitions
3. Select one representative value from each
4. Create test cases

Example 1 - Age Field (18-60):

Partitions:

- Invalid: age < 18 (select 10)

- Valid: 18 ≤ age ≤ 60 (select 30)

- Invalid: age > 60 (select 65)

Test Cases:

TC1: age = 10 (Expected: Rejected)

TC2: age = 30 (Expected: Accepted)

TC3: age = 65 (Expected: Rejected)

Example 2 - Month Field (1-12):

Partitions:

- Invalid: month < 1

- Valid: 1 ≤ month ≤ 12

- Invalid: month > 12

Test Cases:

TC1: month = 0 (Invalid)

TC2: month = 6 (Valid)

TC3: month = 13 (Invalid)

Example 3 - File Upload (Max 5MB):

Partitions:

- Invalid: size < 0

- Valid: 0 ≤ size ≤ 5MB

- Invalid: size > 5MB

Test Cases:

TC1: size = -1KB (Invalid)

TC2: size = 2MB (Valid)

TC3: size = 6MB (Invalid)

Benefits:

- Reduces number of test cases
- Comprehensive coverage
- Time efficient
- Systematic approach

34. What is Boundary Value Analysis (BVA)?

Definition: Testing at boundaries between partitions as defects tend to occur at edges of input ranges.

Concept: Errors occur at boundary values more frequently than in the middle of ranges.

Rules: For range [min, max]:

- Test: min-1, min, min+1
- Test: max-1, max, max+1

Example 1 - Valid Range (10-100):

Boundary Values:

- 9 (just below min - Invalid)

- 10 (minimum - Valid)

- 11 (just above min - Valid)

- 99 (just below max - Valid)

- 100 (maximum - Valid)

- 101 (just above max - Invalid)

Test Cases: 6 test cases

Example 2 - Password Length (8-20 characters):

Boundary Values:

TC1: 7 characters (Invalid)

TC2: 8 characters (Valid)

TC3: 9 characters (Valid)

TC4: 19 characters (Valid)

TC5: 20 characters (Valid)

TC6: 21 characters (Invalid)

Example 3 - Temperature Control (-10°C to 50°C):

Boundary Values:

TC1: -11°C (Invalid)

TC2: -10°C (Valid)

TC3: -9°C (Valid)

TC4: 49°C (Valid)

TC5: 50°C (Valid)

TC6: 51°C (Invalid)

Types:

1. Two-Value BVA:

- Test min and max only

2. Three-Value BVA:

- Test min, mid, max

3. Robust BVA:

- Include invalid values (min-1, max+1)

BVA vs Equivalence Partitioning:

Aspect	**BVA**	**Equivalence Partitioning**
Focus	Boundaries	Partitions
Values	Edge values	Any value in partition
Test Cases	More	Fewer
Defect Detection	High	Moderate
Usage	Numeric ranges	All data types

35. What is Decision Table Testing?

Definition: Technique representing combinations of inputs and their corresponding outputs in tabular form. Tests business logic with multiple conditions.

Components:

- **Conditions:** Input conditions
- **Actions:** Expected outputs/results
- **Rules:** Combinations of conditions

Format:

Conditions / Rules	Rule 1	Rule 2	Rule 3	Rule 4
Condition 1	T	T	F	F
Condition 2	T	F	T	F
Action 1	X			X
Action 2		X	X	

Example 1 - Login Functionality:

Conditions	R1	R2	R3	R4
Valid Username	Y	Y	N	N
Valid Password	Y	N	Y	N
Login Success	X			
Invalid Password		X		
Invalid Username			X	
Both Invalid				X

Example 2 - Discount Calculation:

Conditions	R1	R2	R3	R4
Member Type: Gold	Y	N	N	N
Purchase > $1000	Y	Y	N	N
First Time Purchase	Y	Y	Y	N
20% Discount	X			
15% Discount		X		
10% Discount			X	
No Discount				X

Steps:

1. Identify conditions (inputs)
2. Identify actions (outputs)
3. Create decision table
4. Define rules (combinations)
5. Simplify table (combine similar rules)
6. Create test cases from rules

Benefits:

- Complete coverage of combinations
- Clear documentation
- Easy to understand
- Identifies missing requirements
- Good for complex business logic

Limitations:

- Too many conditions = too many rules
- Difficult for large systems
- Time-consuming for many combinations

36. What is State Transition Testing?

Definition: Testing technique where system behavior changes based on current state and input events. Tests state changes and transitions.

Components:

- **States:** Different conditions of system
- **Transitions:** Change from one state to another
- **Events:** Triggers causing transitions
- **Actions:** Operations performed during transition

State Transition Diagram:

[State A] --Event--> [State B] --Event--> [State C]

↑ |

|----------Event----------|

Example 1 - ATM Machine:

States: Idle, Card Inserted, PIN Verified, Transaction

Transitions:

Idle → Card Inserted (Insert Card)

Card Inserted → PIN Verified (Correct PIN)

Card Inserted → Idle (Incorrect PIN 3 times)

PIN Verified → Transaction (Select Transaction)

Transaction → Idle (Complete/Cancel)

PIN Verified → Idle (Timeout)

Example 2 - Order Status:

States: Pending, Processing, Shipped, Delivered, Cancelled

Transitions:

Pending → Processing (Payment confirmed)

Processing → Shipped (Order dispatched)

Shipped → Delivered (Delivery confirmed)

Pending → Cancelled (Cancel by customer)

Processing → Cancelled (Cancel by admin)

State Transition Table:

Current State | Event | Next State | Action

---|---|---|---

Idle | Insert Card | Card In | Read card

Card In | Correct PIN | Verified | Display menu

Card In | Wrong PIN (3x) | Idle | Retain card

Verified | Select Trans | Transaction | Process

Transaction | Complete | Idle | Eject card

Testing Approach:

1. N-Switch Coverage:

- Test all transitions once
- Basic coverage

2. N+1 Switch Coverage:

- Test transition pairs

- Better coverage

Coverage Types:

- All states coverage
- Valid transitions coverage
- Invalid transitions coverage

Example Test Cases - ATM:

TC1: Idle → Insert Card → Card Inserted

TC2: Card Inserted → Correct PIN → PIN Verified

TC3: PIN Verified → Select Trans → Transaction

TC4: Transaction → Complete → Idle

TC5: Card Inserted → Wrong PIN (3x) → Idle (Invalid)

TC6: PIN Verified → Timeout → Idle (Invalid)

Benefits:

- Good for dynamic systems
- Tests state changes
- Identifies invalid transitions
- Clear visualization
- Tests sequence of events

When to Use:

- Workflow applications
- Embedded systems
- Banking applications
- Gaming applications
- Any state-dependent system

37. What is Use Case Testing?

Definition: Testing technique derived from use cases. Tests complete scenarios from user perspective.

Use Case Components:

- **Actor:** User or system interacting
- **Preconditions:** Required state before execution
- **Main Flow:** Primary success scenario

- **Alternative Flows:** Other success paths
- **Exception Flows:** Error scenarios
- **Postconditions:** System state after execution

Example - Online Shopping:

Use Case: Purchase Product

Actor: Customer

Precondition: User logged in

Main Flow:

1. User searches product
2. System displays results
3. User selects product
4. System shows product details
5. User adds to cart
6. User proceeds to checkout
7. User enters shipping address
8. User selects payment method
9. System processes payment
10. System confirms order

Alternative Flow:

3a. User applies coupon code

7a. User selects saved address

Exception Flow:

9a. Payment fails → Display error, retry

Postcondition: Order placed, inventory updated

Test Scenarios Derived:

TC1: Main flow - Complete purchase successfully

TC2: Alternative - Purchase with coupon code

TC3: Alternative - Use saved address

TC4: Exception - Payment failure handling

TC5: Exception - Out of stock scenario

TC6: Exception

- Invalid shipping address

Benefits:

- User-centric testing

- Complete scenario coverage

- Based on requirements

- Easy stakeholder understanding

- Tests business workflows

38. What is Error Guessing?

Definition: Testing technique based on tester's experience, intuition, and knowledge of common failure patterns to guess potential defects.

Based On:

- Past experience

- Similar application defects

- Common error-prone areas

- Intuition and creativity

- Domain knowledge

Common Areas to Guess Errors:

1. Input Fields:

- Empty fields

- Special characters

- Very long strings
- SQL injection attempts
- Script injection
- Null values
- Negative numbers

2. Boundaries:
- Maximum limits
- Minimum limits
- Overflow conditions

3. Typical User Mistakes:
- Wrong data format
- Wrong sequence
- Incomplete data

4. Technical Issues:
- Network failures
- Database connection loss
- Memory overflow
- Concurrent access

Example Scenarios:

Login Page:

- Blank username/password
- SQL injection: ' OR '1'='1
- Very long password (>1000 chars)
- Special chars: <script>alert('XSS')</script>
- Copy-paste password with spaces
- Case sensitivity
- Browser back button after logout

File Upload:

- Zero-byte file
- Extremely large file
- Wrong format
- Corrupted file
- Duplicate filename
- Special chars in filename
- Path traversal: ../../etc/passwd

Error Guessing Checklist:

□ Null/blank inputs □ Special characters □ Boundary values □ Invalid data types □ Concurrent operations □ Permission issues □ Network failures □ Browser compatibility □ Mobile responsiveness □ Session timeout □ Memory leaks □ Race conditions

Benefits:

- Finds defects missed by formal techniques

- Quick to execute

- No documentation needed

- Experience-based

- Complements other techniques

Limitations:

- Depends on tester skill

- Not systematic

- Not repeatable

- No guarantee of coverage

- Difficult to track

39. What is Pairwise Testing?

Definition: Combinatorial testing technique ensuring all possible discrete combinations of parameters are tested in pairs.

Concept:

Research shows most defects are triggered by interaction of two parameters. Testing all pairs provides good coverage with fewer test cases.

Example - Login Page:

Parameters: Browser: Chrome, Firefox, Edge OS: Windows, Mac, Linux User Type: Admin, Regular User

Total Combinations: 3 × 3 × 2 = 18

Pairwise Test Cases: ~9 test cases TC1: Chrome, Windows, Admin TC2: Chrome, Mac, Regular TC3: Firefox, Windows, Regular TC4: Firefox, Linux, Admin TC5: Edge, Mac, Admin TC6: Edge, Linux, Regular TC7: Chrome, Linux, Regular TC8: Firefox, Mac, Admin TC9: Edge, Windows, Regular

Benefits:

- Reduces test cases significantly
- Good defect detection
- Efficient coverage
- Systematic approach

Tools:

- PICT (Microsoft)
- AllPairs
- ACTS (NIST)

SECTION 4: DEFECT MANAGEMENT

40. What is a Defect/Bug?

Definition: Deviation of actual result from expected result. A flaw in software causing incorrect or unexpected behavior.

Other Terms:

- Bug
- Issue
- Fault
- Error
- Problem
- Incident

Types Based on Origin:

1. Functional Defects:

- Feature not working as expected
- Incorrect calculations
- Wrong data displayed

2. Logical Defects:

- Algorithm issues
- Incorrect logic implementation
- Wrong conditions

3. Performance Defects:

- Slow response time
- Memory leaks
- High CPU usage

4. UI Defects:

- Alignment issues
- Color mismatches
- Broken layouts
- Typos

5. Security Defects:

- Unauthorized access
- Data leaks
- Injection vulnerabilities

6. Compatibility Defects:

- Browser issues
- OS-specific problems
- Device-specific issues

41. What is Defect Life Cycle?

Definition: Journey of a defect from discovery through resolution to closure.

States:

1. New:

- Defect identified and logged
- Awaiting review
- Initial state

2. Assigned:

- Reviewed by lead/manager
- Assigned to developer
- Ready for fixing

3. Open:

- Developer acknowledges
- Starts working on fix
- In progress

4. Fixed:

- Developer fixed the issue
- Build ready for testing
- Awaiting verification

5. Retest:

- Tester retesting the fix
- Verification in progress

6. Verified:

- Fix confirmed working
- Defect resolved successfully
- Ready for closure

7. Closed:

- Final state
- No further action needed
- Defect successfully resolved

8. Reopened:

- Fix didn't work
- Issue still exists
- Back to developer

9. Rejected:

- Not a valid defect
- Working as designed
- Unable to reproduce

10. Deferred:

- Fix postponed

- Scheduled for future release

- Low priority

11. Duplicate:

- Same defect already logged

- References original defect

- Closed as duplicate

Flow Diagram:

New → Assigned → Open → Fixed → Retest → Verified → Closed ↓ ↓ Rejected Reopened ↓ ↓ Closed ← Back to Assigned

42. What is Severity?

Definition: Impact of defect on application functionality and business. Indicates how serious the defect is.

Severity Levels:

Critical/Blocker (S1):

- Application crash

- Data loss/corruption

- Security breach

- Complete functionality blocked

- No workaround available

- Production down

Example: Payment gateway not working

High/Major (S2):

- Major functionality failure

- Significant impact on business

- Workaround difficult or complex

- Affects multiple users

Example: Unable to add items to cart

Medium/Moderate (S3):

- Minor functionality issue

- Workaround available

- Affects specific scenario

- Limited user impact

Example: Search filter not working for one category

Low/Minor (S4):

- Cosmetic issues

- UI/UX problems

- Typos or grammatical errors

- No functionality impact

- Easy workaround

Example: Button color mismatch, text alignment issue

43. What is Priority?

Definition: Urgency of fixing the defect. Indicates when the defect should be fixed.

Priority Levels:

P1 - Immediate/Critical:

- Fix immediately

- Blocks testing/release

- Top priority

- Usually fixed within hours

P2 - High:

- Fix in current release

- Important but not blocking

- Fix within days

P3 - Medium:

- Fix in upcoming releases

- Can wait

- Fix within weeks

P4 - Low:

- Fix when time permits

- Nice to have

- May defer to future versions

44. Severity vs Priority - Combinations

High Severity, High Priority:

Example: Payment processing fails for all users Impact: Critical business function down Action: Fix immediately

High Severity, Low Priority:

Example: Company logo misspelled on homepage Impact: Embarrassing but not functional Action: Can wait till next deployment

Low Severity, High Priority:

Example: CEO's name typo on about page Impact: Minor but highly visible Action: Fix immediately before launch

Low Severity, Low Priority:

Example: Footer text alignment off by 2px Impact: Minimal visual issue Action: Fix when convenient

Real-World Examples:

| Defect | Severity | Priority | Reason |

|--------|----------|----------|--------|

| App crashes on launch | Critical | High | Blocks all users |

| Spelling error in terms of service | Low | Low | Minor, not urgent |

| Wrong CEO name on site | Low | High | Embarrassing, visible |

| Feature works but has memory leak | High | Medium | Serious but workaround exists |

| Crash in rarely used admin feature | High | Low | Serious but limited users |

45. What is a Defect Report?

Definition: Detailed document describing a defect found during testing. Provides all information needed to reproduce and fix.

Complete Defect Report Template:

DEFECT ID: BUG-2024-001

SUMMARY: Unable to login with valid credentials

DESCRIPTION: When user enters correct username and password and clicks login button, system shows "Invalid credentials" error instead of logging in.

SEVERITY: High PRIORITY: High STATUS: New

REPORTED BY: John Doe REPORTED DATE: 07-Dec-2024 ASSIGNED TO: Development Team MODULE: Authentication

ENVIRONMENT:

- Application: MyApp v2.5
- OS: Windows 11
- Browser: Chrome 120
- Database: MySQL 8.0
- Test Environment: QA Server

STEPS TO REPRODUCE:

1. Navigate to https://myapp.com/login
2. Enter username: testuser@email.com
3. Enter password: Test@123

4. Click "Login" button

EXPECTED RESULT: User should be logged in and redirected to dashboard

ACTUAL RESULT: Error message displays: "Invalid credentials. Please try again" User remains on login page

TEST DATA: Username: testuser@email.com Password: Test@123

ATTACHMENTS:

- Screenshot: login_error.png
- Video: login_attempt.mp4
- Logs: application.log

ADDITIONAL INFORMATION:

- Issue started after v2.5 deployment
- Works fine in v2.4
- Affects all user accounts tested
- Console shows 401 error

REPRODUCIBILITY: 100% (10/10 attempts)

WORKAROUND: None available

Essential Fields:

1. **Summary:** One-line description
2. **Description:** Detailed explanation
3. **Steps to Reproduce:** Exact steps
4. **Expected vs Actual:** Clear comparison
5. **Environment:** Complete context
6. **Attachments:** Screenshots, logs, videos
7. **Severity/Priority:** Impact assessment
8. **Reproducibility:** How often it occurs

46. What makes a Good Defect Report?

Qualities:

1. Clear and Concise:

- One defect per report
- Short, descriptive title
- No ambiguity

2. Reproducible:

- Exact steps provided
- Consistent results
- Can be replicated

3. Complete Information:

- All relevant details
- Environment specifics
- Test data included

4. Specific:

- Not vague or general
- Precise location
- Exact behavior described

5. Objective:

- Facts, not opinions
- No emotional language
- Professional tone

Good vs Bad Examples:

Bad:

Title: Login not working Description: I tried to login but couldn't. Fix it.

Good:

Title: Login fails with valid credentials showing "Invalid credentials" error Description: When entering valid username (test@email.com) and password (Test@123), system displays error "Invalid credentials" instead of logging in. Issue occurs 100% of the time across Chrome, Firefox browsers on Windows 11. Steps: [Detailed steps] Environment: [Complete environment] Attachments: [Screenshots, logs]

47. What is Defect Triage?

Definition: Process of reviewing, prioritizing, and assigning defects. Meeting where team decides which defects to fix and when.

Participants:

- Test Lead/Manager
- Development Lead
- Project Manager
- Business Analyst
- Product Owner

Activities:

1. Review:

- Validate defect is genuine
- Check for duplicates
- Verify reproducibility

2. Prioritize:

- Assign severity
- Assign priority
- Consider business impact

3. Assign:

- Assign to right developer

- Set target fix date
- Allocate resources

4. Defer/Reject:
- Identify non-issues
- Mark duplicates
- Defer low-priority defects

Triage Meeting:

Agenda:

1. Review new defects
2. Discuss critical defects
3. Prioritize defect fixes
4. Assign defects to developers
5. Review deferred defects
6. Decide on deferments
7. Set target dates

Triage Metrics:
- Defects triaged per session
- Time to triage
- Defect acceptance rate
- Defect rejection rate
- Defer rate

48. What is Defect Density?

Definition: Metric measuring number of defects per unit size of software (per KLOC - thousand lines of code, per function point, or per module).

Formula:

Defect Density = Total Defects / Size

Where Size can be:

- Lines of Code (LOC)
- Function Points
- Number of Modules
- Number of Features

Example:

Application: 50,000 lines of code Defects Found: 150

Defect Density = 150 / 50 = 3 defects per KLOC

Industry Standards:

- 1-25 defects per KLOC (varies by industry)

- Lower is better (but 0 is suspicious)

- Banking: 10-15 defects/KLOC

- Web apps: 15-20 defects/KLOC

Usage:

- Compare modules

- Predict defects

- Quality indicator

- Focus testing effort

49. What is Defect Removal Efficiency (DRE)?

Definition: Metric measuring percentage of defects found before release compared to total defects.

Formula:

DRE = (Defects Found Before Release / Total Defects) × 100

Where: Total Defects = Defects Before Release + Defects After Release

Example:

Defects found in testing: 95 Defects found in production: 5 Total defects: 100

DRE = (95 / 100) × 100 = 95%

Interpretation:

- DRE > 95%: Excellent
- DRE 85-95%: Good
- DRE 70-85%: Average
- DRE < 70%: Poor

Goal:

Higher DRE = Better testing = Fewer production issues

50. What is Defect Leakage?

Definition: Defects found in production that should have been found during testing phases. Measures testing effectiveness.

Formula:

Defect Leakage = (Defects in Production / Total Defects) × 100

Example:

Defects found in UAT: 80 Defects found in Production: 20 Total: 100

Defect Leakage = (20 / 100) × 100 = 20%

Target:

- Defect Leakage < 5%: Excellent
- Defect Leakage 5-10%: Good
- Defect Leakage > 10%: Needs improvement

Causes:

- Inadequate test coverage
- Missed test scenarios
- Environment differences
- Time pressure
- Lack of resources
- Poor requirement understanding

51. What is Defect Age?

Definition: Time elapsed from defect detection to defect closure. Measures how quickly defects are resolved.

Formula:

Defect Age = Defect Closed Date - Defect Reported Date

Example:

Reported: Dec 1, 2024 Closed: Dec 7, 2024 Defect Age: 6 days

Categories:

- Fresh: < 7 days
- Medium: 7-30 days
- Aged: > 30 days

Why Important:

- Tracks resolution speed
- Identifies bottlenecks
- Measures team efficiency
- Customer satisfaction indicator

Goals:

- Critical: < 1 day

- High: < 3 days

- Medium: < 7 days

- Low: < 30 days

SECTION 5: TEST DOCUMENTATION

52. What is a Test Plan Document?

Definition: Comprehensive document outlining testing scope, approach, resources, schedule, and activities.

IEEE 829 Standard Sections:

1. Test Plan Identifier:

Unique ID for the test plan

2. Introduction:

- Purpose of document

- Project background

- Scope of testing

3. Test Items:

- Features to be tested

- Application modules

- Version information

4. Features to be Tested:

Detailed list of functionalities

5. Features Not to be Tested:

Out of scope items with rationale

6. Approach:

- Testing strategy
- Test levels
- Test types
- Test techniques

7. Item Pass/Fail Criteria:

- Definition of success
- Acceptance criteria
- Quality gates

8. Suspension and Resumption Criteria:

When to pause/restart testing

9. Test Deliverables:

- Test cases
- Test scripts
- Defect reports
- Test summary report

10. Test Environment:

- Hardware requirements
- Software requirements
- Network configuration
- Test data needs

11. Responsibilities:

- Roles and responsibilities

- Test team structure

- Resource allocation

12. Staffing and Training:

- Team members

- Training needs

- Skill requirements

13. Schedule:

- Testing timeline

- Milestones

- Dependencies

14. Risks and Contingencies:

- Identified risks

- Mitigation strategies

- Backup plans

15. Approvals:

Sign-offs from stakeholders

53. What is a Test Strategy Document?

Definition: High-level document defining overall testing approach for organization or project. Long-term testing methodology.

Components:

1. Testing Objectives:

- Ensure quality standards
- Meet requirements
- Minimize defects
- Reduce risks

2. Testing Scope:

In Scope:

- Functional testing
- Integration testing
- System testing
- UAT

Out of Scope:

- Performance testing (separate project)
- Localization testing

3. Test Approach:

- Black box testing
- Risk-based testing
- Automated regression
- Manual exploratory

4. Test Levels:

1. Unit Testing (Developers)
2. Integration Testing (Developers + QA)
3. System Testing (QA Team)
4. UAT (Business Users)

5. Test Types:

- Functional
- Security
- Compatibility
- Usability

6. Testing Tools:

- Test Management: JIRA

- Automation: Selenium
- Performance: JMeter
- Defect Tracking: Bugzilla

7. Defect Management:

- Severity/Priority definitions
- Defect lifecycle
- Triage process
- Reporting frequency

8. Test Environment Strategy:

- Dev environment
- QA environment
- Staging environment
- Production environment

9. Risk Analysis:

Risk: Tight timeline Mitigation: Prioritize critical features

Risk: Limited resources Mitigation: Automate regression tests

10. Entry/Exit Criteria:

Defined for each test level

54. What is Test Scenario Document?

Definition: Document listing high-level test conditions or situations to be validated.

Format:

Test Scenario ID: TS_LOGIN_001 Module: User Authentication Test Scenario: Verify login functionality Priority: High Status: In Progress

Test Scenarios:

1. Verify login with valid credentials
2. Verify login with invalid credentials
3. Verify login with blank fields
4. Verify forgot password functionality
5. Verify remember me functionality
6. Verify account lockout after failed attempts
7. Verify session timeout
8. Verify logout functionality

Each scenario expands to multiple test cases

55. What is Test Case Document?

Definition: Detailed document with step-by-step instructions, test data, and expected results.

Standard Format:

TEST CASE SPECIFICATION

Test Case ID: TC_LOGIN_001 Test Scenario ID: TS_LOGIN_001 Module: User Authentication Test Case Title: Verify login with valid credentials Test Case Description: Validate that user can successfully login with correct username and password

Priority: High Severity: Critical Test Type: Functional Test Environment: QA Browser: Chrome 120

Preconditions:

1. Application is accessible
2. User is registered with credentials: Username: testuser@email.com Password: Test@123
3. User is not already logged in

TEST STEPS:

Step | Action | Expected Result

1 | Navigate to login page | Login page displayed 2 | Enter valid username | Username accepted 3 | Enter valid password | Password masked, accepted 4 | Click "Login" button | Processing indicator shown 5 | Verify redirection | Redirected to dashboard 6 | Verify user session | User name displayed in header

Test Data: Username: testuser@email.com Password: Test@123

Expected Result:

- User successfully logged in
- Dashboard displayed
- Welcome message shown
- User profile icon visible

Actual Result: [To be filled during execution]

Status: [Pass/Fail/Blocked/Not Executed] Execution Date: Executed By:

Postconditions:

- User session created
- User logged in state maintained

Comments/Notes: [Additional observations]

Attachments: [Screenshots if test fails]

56. What is Test Summary Report?

Definition: Document summarizing test activities, results, and quality assessment at end of testing cycle.

Contents:

1. Introduction:

- Purpose

- Scope

- Audience

2. Test Objectives:

What testing aimed to achieve

3. Test Summary:

Total Test Cases: 500 Executed: 485 Passed: 450 Failed: 30 Blocked: 5 Not Executed: 15

Pass Percentage: 92.78%

4. Test Environment:

Configuration details

5. Defect Summary:

Total Defects: 125

By Severity: Critical: 5 (Fixed: 5) High: 20 (Fixed: 18, Open: 2) Medium: 50 (Fixed: 45, Open: 5) Low: 50 (Fixed: 40, Deferred: 10)

By Status: Fixed: 108 Open: 7 Deferred: 10

6. Test Coverage:

Requirements Coverage: 98% Code Coverage: 85% Feature Coverage: 100%

7. Test Execution Summary:

Start Date: Nov 1, 2024 End Date: Dec 7, 2024 Duration: 37 days Planned: 40 days

8. Risks and Issues:

Risk: 7 high-priority defects open Mitigation: Extended testing by 3 days

Issue: Environment downtime Impact: 2 days testing delay

9. Recommendations:

- Fix 7 open defects before release
- Conduct additional security testing
- Improve error handling in payment module

10. Sign-off:

Approval from stakeholders

57. What is Defect Report/Bug Report?

Definition: Document describing a defect found during testing with all necessary details for reproduction and fixing.

Covered in detail in Section 4 (Defect #45)

58. What is Test Execution Report?

Definition: Daily/weekly report showing test execution progress and status.

Format:

TEST EXECUTION REPORT Date: December 7, 2024 Project: E-commerce Application Release: v2.5

EXECUTION SUMMARY:

Total Test Cases: 500 Planned Today: 50 Executed Today: 48 Pending: 2

OVERALL PROGRESS:

Total Executed: 485/500 (97%) Passed: 450 (92.78%) Failed: 30 (6.19%) Blocked: 5 (1.03%)

TODAY'S EXECUTION:

Passed: 44 Failed: 3 Blocked: 1

NEW DEFECTS:

Critical: 0 High: 1 Medium: 2 Low: 0

DEFECT STATUS:

Open: 7 Fixed Pending Retest: 12 Closed: 108

TEST ENVIRONMENT:

Status: Stable Uptime: 100% Issues: None

RISKS/BLOCKERS:

- Payment gateway intermittent issues
- Test data refresh pending

NEXT DAY PLAN:

- Complete remaining 15 test cases
- Retest 12 fixed defects
- Execute regression suite

59. What is Requirements Traceability Matrix (RTM)?

Covered in detail earlier (Concept #14)

Additional points:

RTM Benefits:

- 100% coverage verification
- Impact analysis tool
- Compliance documentation
- Audit trail
- Project tracking
- Gap identification

RTM Maintenance:

- Update with requirement changes
- Track throughout project
- Version control
- Regular reviews
- Tool support (e.g., JIRA, ALM)

SECTION 6: TESTING METHODOLOGIES & APPROACHES

60. What is Agile Testing?

Definition: Testing approach aligned with Agile methodology. Continuous testing throughout development with frequent feedback.

Agile Testing Principles:

1. Testing is continuous activity
2. Continuous feedback
3. Whole team responsibility
4. Face-to-face communication
5. Simplicity

6. Customer satisfaction

7. Respond to change

8. Sustainable pace

Agile Testing Quadrants:

Q1 - Technology Facing, Support Programming:

- Unit tests
- Component tests
- Automated tests
- Purpose: Guide development

Q2 - Business Facing, Support Team:

- Functional tests
- Story tests
- Prototypes
- Purpose: Verify features

Q3 - Business Facing, Critique Product:

- Exploratory testing
- Usability testing
- UAT
- Purpose: Find issues

Q4 - Technology Facing, Critique Product:

- Performance testing
- Security testing
- Load testing
- Purpose: Check quality attributes

Agile Testing Practices:

- Test-Driven Development (TDD)
- Behavior-Driven Development (BDD)
- Acceptance Test-Driven Development (ATDD)
- Continuous Integration
- Continuous Testing
- Automation
- Pair programming with testers

Tester Role in Agile:

- Part of development team
- Participate in sprint planning
- Write acceptance criteria
- Automate tests
- Exploratory testing
- Collaborate continuously
- Attend daily standups

Sprint Testing Cycle:

Day 1-2: Sprint Planning, Story analysis Day 3-7: Test case creation, Automation Day 8-12: Test execution, Defect logging Day 13-14: Regression, Sprint review, Retrospective

61. What is Test-Driven Development (TDD)?

Definition: Development approach where tests are written before writing code. Code is written to make tests pass.

TDD Cycle (Red-Green-Refactor):

1. Red - Write Failing Test:

Write test for new functionality Test fails (no code yet)

2. Green - Write Minimum Code:

Write simplest code to pass test Test passes

3. Refactor - Improve Code:

Refactor without changing behavior Tests still pass

Example:

// Step 1: Write test (Red) @Test public void testAddition() { Calculator calc = new Calculator(); assertEquals(5, calc.add(2, 3)); } // Test fails - Calculator.add() doesn't exist

// Step 2: Write code (Green) public class Calculator { public int add(int a, int b) { return a + b; } } // Test passes

// Step 3: Refactor (if needed) // Improve code structure, tests still pass

Benefits:

- Better code quality

- High test coverage

- Less debugging

- Living documentation

- Confidence in changes

Challenges:

- Initial time investment

- Requires discipline

- Learning curve

- Not for all scenarios

62. What is Behavior-Driven Development (BDD)?

Definition: Extension of TDD focusing on behavior from user perspective. Tests written in natural language (Given-When-Then).

BDD Format:

Feature: User Login

Scenario: Successful login with valid credentials Given user is on login page When user enters valid username "test@email.com" And user enters valid password "Test@123" And user clicks login button Then user should be redirected to dashboard And user should see welcome message

Gherkin Syntax:

- **Feature:** High-level description

- **Scenario:** Specific test case

- **Given:** Preconditions

- **When:** Action/event

- **Then:** Expected outcome

- **And:** Additional steps

Tools:

- Cucumber (Java, Ruby)

- SpecFlow (.NET)

- Behave (Python)

- JBehave (Java)

Benefits:

- Non-technical can understand

- Living documentation

- Collaboration improves

- Focus on behavior

- Bridges communication gap

Example - E-commerce:

Feature: Shopping Cart

Scenario: Add product to cart Given user is logged in And user is on product page When user clicks "Add to Cart" button Then product should be added to cart And cart count should increase by 1 And success message should display

Scenario: Remove product from cart Given user has 2 items in cart When user clicks remove on first item Then item should be removed And cart count should be 1

63. What is Acceptance Test-Driven Development (ATDD)?

Definition: Collaborative approach where acceptance criteria are defined before development. Similar to BDD but focuses on acceptance tests.

Process:

1. **Discuss:** Team discusses user story
2. **Distill:** Define acceptance criteria
3. **Develop:** Write tests then code
4. **Demo:** Show working software

Participants:

- Customer/Product Owner
- Developer
- Tester

Example:

User Story: As a customer, I want to search products by name so that I can quickly find what I need.

Acceptance Criteria:

1. Search box visible on all pages
2. Search accepts text input
3. Results display within 2 seconds
4. Relevant products shown first
5. "No results" message if nothing found
6. Minimum 3 characters required

Tests written before code to verify each criterion

ATDD vs BDD vs TDD:

Aspect	TDD	BDD	ATDD

| Focus | Unit level | Behavior | Acceptance |

| Language | Technical | Natural | Natural |

| Participants | Developers | Whole team | Whole team |

| Level | Unit tests | Any level | Acceptance tests |

| Tools | JUnit, NUnit | Cucumber | FitNesse, Robot |

64. What is Risk-Based Testing?

Definition: Testing approach prioritizing features based on risk. High-risk areas tested more thoroughly.

Risk Factors:

1. Probability:

- Likelihood of failure

- Historical data

- Complexity

- Technology maturity

2. Impact:

- Business impact

- User impact

- Financial loss

- Reputation damage

Risk Calculation:

Risk = Probability × Impact

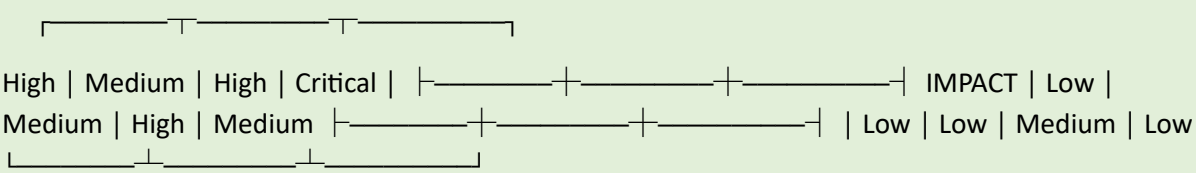
Example: Feature: Payment Processing Probability: High (8/10) Impact: Critical (10/10) Risk Score: 80 (Very High)

Feature: Footer Links Probability: Low (2/10) Impact: Low (2/10) Risk Score: 4 (Very Low)

Risk Matrix:

PROBABILITY

Low Medium High



Testing Priority:

1. Critical Risk: Maximum testing

2. High Risk: Thorough testing

3. Medium Risk: Moderate testing

4. Low Risk: Minimal testing

Example - Banking App:

Critical Risk:

- Money transfer
- Account balance
- Login security

High Risk:

- Bill payment
- Statement generation

Medium Risk:

- Profile update
- Notification settings

Low Risk:

- UI themes
- Help text

Benefits:

- Optimizes testing effort

- Focuses on important areas

- Better resource utilization

- Risk mitigation
- Stakeholder confidence

65. What is Positive Testing?

Definition: Testing with valid inputs to verify system accepts them and produces expected results. Tests "what system should do."

Also Called:

- Happy path testing
- Expected behavior testing

Purpose:

- Verify normal operation
- Test valid scenarios
- Ensure requirements met
- Validate business logic

Examples:

Login:

Valid Credentials:

- Username: validuser@email.com

- Password: Valid@123

Expected: Successful login

Age Field (18-60):

Valid inputs: 18, 25, 45, 60

Expected: Accepted

File Upload (Max 5MB):

Valid files: 1MB, 3MB, 5MB PDF

Expected: Upload successful

Search Functionality:

Valid search term: "laptop"

Expected: Relevant results displayed

66. What is Negative Testing?

Definition: Testing with invalid, unexpected, or incorrect inputs to verify system handles errors gracefully. Tests "what system should NOT do."

Purpose:

- Verify error handling
- Test system robustness
- Prevent crashes
- Ensure proper error messages
- Security validation

Examples:

Login:

Invalid Scenarios:

- Blank username

- Wrong password

- SQL injection: ' OR '1'='1

- Special characters: <script>

- Very long password (1000+ chars)

- Spaces in password

Expected: Appropriate error messages, no crash

Age Field (18-60):

Invalid inputs:

- Negative: -5

- Below range: 10

- Above range: 70

- Text: "twenty"

- Special chars: @#$

- Decimal: 25.5

Expected: Validation error displayed

Email Field:

Invalid formats:

- Missing @: testgmail.com

- Missing domain: test@

- Special chars: test@@gmail.com

- Spaces: test @gmail.com

Expected: "Invalid email format" error

File Upload:

Invalid scenarios:

- Oversized file (10MB)

- Wrong format (.exe instead of .pdf)

- Corrupted file

- Empty file

- Virus-infected file

Expected: Appropriate error, upload rejected

Positive vs Negative:

Aspect	Positive	Negative
Input	Valid	Invalid
Purpose	Verify function	Verify error handling
Coverage	Expected paths	Unexpected paths
Examples	Valid data	Invalid data
Result	Success	Graceful failure

67. What is Monkey Testing?

Definition: Random testing without test cases or plan. Tester behaves like a monkey randomly clicking/entering data trying to break the system.

Characteristics:

- No predefined test cases
- Completely random
- No documentation
- Unusual inputs
- Unpredictable actions
- Goal: Break the system

Types:

1. Dumb Monkey Testing:

- No product knowledge
- Random actions

- Any user could do
- Example: Random button clicking

2. Smart Monkey Testing:

- Product knowledge
- Informed random testing
- Valid user workflows with random data
- Example: Valid flow with edge case data

3. Brilliant Monkey Testing:

- Deep product knowledge
- Strategic randomness
- Focuses on weak areas
- Example: Targeted random testing on known problem areas

Examples:

E-commerce App:

- Click buttons rapidly 100 times

- Enter special characters everywhere

- Open 50 tabs simultaneously

- Switch between pages rapidly

- Fill forms with random data

- Upload random file types

- Browser back/forward repeatedly

When Used:

- After formal testing
- To find crash scenarios
- Stress testing
- Find unexpected behavior

Benefits:

- Finds unusual bugs
- No test case creation time
- Fresh perspective
- Unbiased testing

Limitations:

- Cannot be reproduced easily
- No coverage guarantee
- Time-consuming
- Depends on tester
- Difficult to track

68. What is Gorilla Testing?

Definition: Testing one module/functionality repeatedly and thoroughly with various inputs to ensure it's robust. Intensive testing of specific area.

Characteristics:

- Focus on single module
- Exhaustive testing
- Different test data
- Multiple scenarios
- Thorough coverage

Example:

Login Module Gorilla Testing:

- Test with 1000 different usernames

- Test with 1000 different passwords

- Test different combinations

- Test special characters

- Test international characters

- Test copy-paste scenarios

- Test auto-fill

- Test remember me

- Test 100 times continuously

Difference from Monkey Testing:

Aspect	**Monkey**	**Gorilla**
Scope	Entire application	Single module
Approach	Random	Focused

Aspect	Monkey	Gorilla
Coverage	Broad	Deep
Repeatability	Difficult	Easier

69. What is Recovery Testing?

Definition: Testing how well system recovers from crashes, hardware failures, or other catastrophic problems.

What is Tested:

- System restart after crash
- Data integrity after failure
- Automatic recovery procedures
- Backup and restore
- Failover mechanisms
- Checkpointing

Scenarios:

1. Application Crash:

- Force terminate application

- Restart application

- Verify: Data not corrupted

- Verify: Session recovered

- Verify: No data loss

2. Database Failure:

- Simulate database crash

- Verify: Application handles gracefully

- Verify: Error message displayed

- Verify: Reconnection attempts

- Verify: No data corruption

3. Network Failure:

- Disconnect network

- Perform operations

- Reconnect network

- Verify: Transactions retry

- Verify: Data syncs

- Verify: No duplicates

4. Power Failure:

- Simulate power loss

- Restart system

- Verify: Unsaved data handling

- Verify: System boots correctly

- Verify: Data consistency

Example - E-commerce:

Scenario: User filling checkout form, system crashes

Test:

1. Fill 80% of checkout form

2. Kill application process

3. Restart application

4. Login again

5. Navigate to checkout

Expected:

- Form data recovered OR

- Clear message about data loss

- No corrupted data

- System stable

70. What is Installation Testing?

Definition: Testing software installation, update, and uninstallation processes.

Test Areas:

1. Installation Testing:

- Fresh installation

- Installation path selection

- Disk space verification

- Prerequisites check

- License agreement
- Installation options
- Progress indication
- Successful completion
- Shortcuts created
- Registry entries

2. Update Testing:

- Upgrade from previous version
- Incremental updates
- Data migration
- Settings preservation
- Rollback capability
- Version verification

3. Uninstallation Testing:

- Complete removal
- Registry cleanup
- File deletion
- Settings removal
- No orphan files
- System stability after uninstall

Test Scenarios:

1. Install on minimum system requirements
2. Install on recommended system requirements
3. Install with insufficient disk space
4. Install on different OS versions
5. Install over existing version
6. Install while other apps running
7. Interrupt installation mid-way
8. Install without admin privileges
9. Silent installation
10. Custom installation

SECTION 7: SPECIALIZED TESTING TYPES

71. What is Compatibility Testing?

Definition: Testing to ensure application works across different environments, platforms, browsers, devices, and configurations.

Types:

1. Browser Compatibility:

Browsers to Test:

- Chrome (latest 3 versions)
- Firefox (latest 3 versions)
- Safari (latest 2 versions)
- Edge (latest 2 versions)
- Opera

Test Aspects:

- UI rendering
- Functionality
- JavaScript execution
- CSS display
- Responsive design

2. Operating System Compatibility:

- Windows (10, 11)
- macOS (Monterey, Ventura, Sonoma)
- Linux (Ubuntu, Fedora)
- Mobile OS (iOS, Android)

Test Aspects:

- Installation
- Functionality
- File operations
- System integration

3. Device Compatibility:

- Desktops
- Laptops
- Tablets
- Smartphones (various screen sizes)
- Different resolutions

Test Aspects:

- Display
- Touch interactions
- Orientation (portrait/landscape)
- Hardware integration

4. Network Compatibility:

- WiFi
- 4G/5G
- 3G
- Low bandwidth
- Offline mode

Test Aspects:

- Loading time
- Data synchronization
- Error handling

5. Database Compatibility:

- MySQL
- PostgreSQL
- Oracle
- SQL Server
- MongoDB

Test Aspects:

- Data operations

- Performance

- Migration

6. Software Compatibility:

- Different software versions

- Third-party integrations

- APIs

- Plugins

Test Aspects:

- Integration

- Data exchange

- Version conflicts

Testing Approach:

Matrix-based testing:

	Chrome	Firefox	Safari	Edge
Windows 10	✓	✓	✗	✓
Windows 11	✓	✓	✗	✓
macOS	✓	✓	✓	✗

72. What is Localization Testing?

Definition: Testing to ensure application is adapted for specific locale/region including language, culture, and regulations.

Test Areas:

1. Language:

- Translation accuracy

- Grammar and spelling

- Text expansion/contraction

- Special characters

- Right-to-left languages (Arabic, Hebrew)

- Double-byte characters (Japanese, Chinese)

2. Date and Time:

- Date format (DD/MM/YYYY vs MM/DD/YYYY)
- Time format (12-hour vs 24-hour)
- Time zones
- Calendar systems
- Week start day

3. Currency:

- Currency symbols ($ £ € ¥)
- Decimal separators (. vs ,)
- Thousand separators
- Currency placement

4. Number Formats:

US: 1,234.56

Europe: 1.234,56

India: 1,23,456.78

5. Cultural:

- Colors meaning
- Images appropriateness
- Icons meaning
- Holidays
- Working days
- Address formats
- Phone number formats

6. Legal/Regulatory:

- Privacy laws (GDPR, CCPA)
- Tax regulations
- Industry standards
- Age restrictions
- Content restrictions

Example Test Cases:

US vs UK:

- Date: 12/07/2024 means different dates

- Spelling: Color vs Colour

- Currency: $ vs £

- Phone: (555) 123-4567 vs +44 20 1234 5678

US vs India:

- Currency: $ vs ₹

- Date: MM/DD/YYYY vs DD/MM/YYYY

- Number: 1,000,000 vs 10,00,000 (10 lakhs)

73. What is Internationalization (i18n) Testing?

Definition: Testing to ensure application can be adapted to various languages and regions without engineering changes.

Key Difference:

- **Internationalization:** Making app adaptable
- **Localization:** Adapting for specific locale

Test Areas:

1. Unicode Support:

- UTF-8 encoding

- Special characters

- Emojis

- Multi-byte characters

2. Text Handling:

- Variable text length

- Text direction (LTR, RTL)

- Text wrapping

- Font support

3. Input Methods:

- Keyboard layouts

- IME (Input Method Editors)

- Voice input

4. Locale-Independent:

- Code doesn't hardcode formats

- Externalized strings

- Configuration-based formatting

Example:

Hard-coded (Bad):

String date = "12/07/2024";

String currency = "$" + amount;

Internationalized (Good):

String date = DateFormat.getDateInstance(locale).format(date);

String currency = NumberFormat.getCurrencyInstance(locale).format(amount);

74. What is Accessibility Testing?

Definition: Testing to ensure application is usable by people with disabilities (visual, auditory, physical, cognitive).

Standards:

- WCAG (Web Content Accessibility Guidelines)
- Section 508 (US)
- ADA (Americans with Disabilities Act)

Categories:

1. Visual Impairment:

Tests:

- Screen reader compatibility

- Alt text for images

- Sufficient color contrast

- Font size adjustability

- No color-only information

- Keyboard navigation

Tools:

- JAWS

- NVDA

- VoiceOver

2. Hearing Impairment:

Tests:

- Captions for videos
- Transcripts for audio
- Visual alerts
- No audio-only information

3. Motor Disability:

Tests:

- Keyboard-only navigation
- Large clickable areas
- No time-limited actions
- Voice commands
- Mouse alternatives

4. Cognitive Disability:

Tests:

- Simple language
- Clear instructions
- Consistent navigation
- Error prevention
- Undo functionality

WCAG Principles (POUR):

1. Perceivable:

- Text alternatives
- Captions
- Adaptable content
- Distinguishable elements

2. Operable:

- Keyboard accessible
- Enough time
- No seizure-inducing content

- Navigable

3. Understandable:

- Readable text

- Predictable behavior

- Input assistance

4. Robust:

- Compatible with assistive technologies

- Valid HTML

- Proper markup

Test Checklist:

□ All images have alt text

□ Form labels properly associated

□ Keyboard navigation works

□ Tab order logical

□ Color contrast ratio ≥ 4.5:1

□ Focus indicators visible

□ Error messages clear

□ Videos have captions

□ Headings properly structured

□ ARIA labels present

□ Screen reader tested

□ Zoom to 200% works

Tools:

- WAVE
- axe DevTools
- Lighthouse
- Color Contrast Analyzer
- Screen readers (JAWS, NVDA)

75. What is Usability Testing?

Definition: Testing focused on user experience, ease of use, and user satisfaction with the application.

What is Evaluated:

1. Learnability
2. Efficiency
3. Memorability
4. Errors
5. Satisfaction

Test Methods:

1. Hallway Testing:

- Grab random people
- Quick informal testing
- Get quick feedback

2. Remote Usability Testing:

- Users test from their location
- Screen sharing
- Task-based testing

3. Expert Review:

- UX experts evaluate
- Heuristic evaluation
- Identify issues

4. A/B Testing:

- Two versions
- Compare metrics
- Data-driven decisions

5. Session Recording:

- Record user sessions
- Analyze behavior
- Identify pain points

Nielsen's 10 Usability Heuristics:

1. Visibility of system status
2. Match between system and real world
3. User control and freedom

4. Consistency and standards

5. Error prevention

6. Recognition over recall

7. Flexibility and efficiency

8. Aesthetic and minimalist design

9. Help users recognize, diagnose errors

10. Help and documentation

Test Scenarios:

Task: Book a flight

Measure:

- Time to complete

- Number of clicks

- Errors made

- User satisfaction (1-10 scale)

- Task completion rate

Observations:

- Where users hesitate

- What confuses them

- What they like

- What frustrates them

Metrics:

- Task Success Rate

- Time on Task

- Error Rate

- Satisfaction Score (SUS - System Usability Scale)

- Navigation efficiency

- Learning curve

76. What is Security Testing?

Definition: Testing to uncover vulnerabilities, threats, and risks in software to protect from malicious attacks.

Objectives:

- Confidentiality
- Integrity
- Availability
- Authentication
- Authorization
- Non-repudiation

Types:

1. Vulnerability Scanning:

- Automated scan for known vulnerabilities

- Security holes identification

- Misconfiguration detection

Tools: Nessus, OpenVAS, Qualys

2. Security Scanning:

- Identify network and system weaknesses

- Find security loopholes

Tools: Nmap, Nikto

3. Penetration Testing:

- Simulate cyber-attack

- Exploit vulnerabilities

- Ethical hacking

Types:

- Black box (no knowledge)

- White box (full knowledge)

- Gray box (partial knowledge)

Tools: Metasploit, Burp Suite, Kali Linux

4. Security Audit:

- Code inspection

- Architecture review

- Compliance check

5. Risk Assessment:

- Identify security risks

- Analyze impact

- Prioritize threats

Common Vulnerabilities (OWASP Top 10):

1. Injection:

SQL Injection:

Test: ' OR '1'='1

Expected: Proper input validation

Example:

Input: admin' OR '1'='1'--

Bad Query: SELECT * FROM users WHERE name='admin' OR '1'='1'--'

Result: Bypass authentication

2. Broken Authentication:

Tests:

- Weak passwords allowed

- Session timeout

- Password reset flaws

- Credential stuffing

3. Sensitive Data Exposure:

Tests:

- Data encryption in transit (HTTPS)

- Data encryption at rest

- Secure transmission

- Proper masking

4. XML External Entities (XXE):

Test: Malicious XML input

Expected: XML parser configured securely

5. Broken Access Control:

Tests:

- Access unauthorized pages
- Vertical privilege escalation
- Horizontal privilege escalation
- Force browsing

6. Security Misconfiguration:

Tests:

- Default credentials
- Directory listing
- Verbose error messages
- Unnecessary services

7. Cross-Site Scripting (XSS):

Test: <script>alert('XSS')</script>

Expected: Input sanitization

Types:

- Reflected XSS
- Stored XSS
- DOM-based XSS

8. Insecure Deserialization:

Test: Manipulated serialized objects

Expected: Proper validation

9. Using Components with Known Vulnerabilities:

Tests:

- Outdated libraries
- Unpatched dependencies
- Known CVEs

10. Insufficient Logging & Monitoring:

Tests:

- Login attempts logged

- Failed attempts monitored

- Audit trail maintained

- Alert mechanisms

Security Test Cases:

Authentication:

□ Brute force attack prevented

□ Account lockout after failed attempts

□ Strong password policy enforced

□ Two-factor authentication works

□ Session timeout implemented

□ Logout invalidates session

Authorization:

□ Users can't access unauthorized data

□ Role-based access control works

□ Horizontal privilege escalation prevented

□ Vertical privilege escalation prevented

Input Validation:

□ SQL injection prevented

□ XSS prevented

□ Command injection prevented

□ Path traversal prevented

□ Buffer overflow prevented

Data Protection:

□ Sensitive data encrypted

□ HTTPS enforced

□ Passwords hashed

□ Credit cards masked

□ Personal data protected

Error Handling:

□ No sensitive info in error messages

□ No stack traces displayed

□ Generic error messages

□ Errors logged securely

SECTION 8: TESTING METRICS

77. What are Testing Metrics?

Definition: Quantitative measures used to track and assess testing status, progress, and quality.

Purpose:

- Track progress
- Measure quality
- Identify bottlenecks
- Make informed decisions
- Improve processes
- Report to stakeholders

Categories:

1. Process Metrics:

- Test case productivity
- Defect detection rate
- Test execution rate

2. Product Metrics:

- Defect density
- Defect severity distribution
- Defect age

3. Project Metrics:

- Schedule variance

- Effort variance
- Cost variance

78. Key Testing Metrics Explained

1. Test Case Coverage:

Formula: (Test Cases Executed / Total Test Cases) × 100

Example:

Total: 500

Executed: 450

Coverage: 90%

2. Requirements Coverage:

Formula: (Requirements Tested / Total Requirements) × 100

Example:

Total Requirements: 100

Tested: 98

Coverage: 98%

3. Defect Density:

Formula: Total Defects / Size (KLOC)

Example:

Defects: 150

Code: 50 KLOC

Density: 3 defects/KLOC

4. Defect Removal Efficiency (DRE):

Formula: (Defects Found Before Release / Total Defects) × 100

Example:

Pre-release: 95

Post-release: 5

DRE: 95%

5. Defect Leakage:

Formula: (Production Defects / Total Defects) × 100

Example:

Production: 5

Total: 100

Leakage: 5%

6. Defect Rejection Ratio:

Formula: (Defects Rejected / Total Defects) × 100

Example:

Rejected: 10

Total: 100

Ratio: 10%

7. Defect Age:

Formula: Closed Date - Reported Date

Average: Sum of all ages / Total defects

8. Test Execution Rate:

Formula: Test Cases Executed / Time Period

Example:

Executed: 50

Time: 1 day

Rate: 50 test cases/day

9. Pass Percentage:

Formula: (Passed Test Cases / Executed Test Cases) × 100

Example:

Passed: 450

Executed: 485

Pass %: 92.78%

10. Defect Detection Percentage (DDP):

Formula: (Defects Found by Testing / Total Defects Injected) × 100

Higher percentage = Better testing

11. Schedule Variance:

Formula: (Actual Time - Planned Time) / Planned Time × 100

Example:

Planned: 30 days

Actual: 33 days

Variance: +10% (delayed)

12. Effort Variance:

Formula: (Actual Effort - Planned Effort) / Planned Effort × 100

13. Test Case Effectiveness:

Formula: (Defects Found / Test Cases Executed) × 100

Measures how effective test cases are in finding defects

14. Defect Distribution:

By Severity:

Critical: 5%

High: 20%

Medium: 45%

Low: 30%

By Module:

Login: 20 defects

Payment: 35 defects

Cart: 15 defects

15. Test Case Productivity:

Formula: Test Cases Written / Person-Hour

Example:

Test Cases: 100

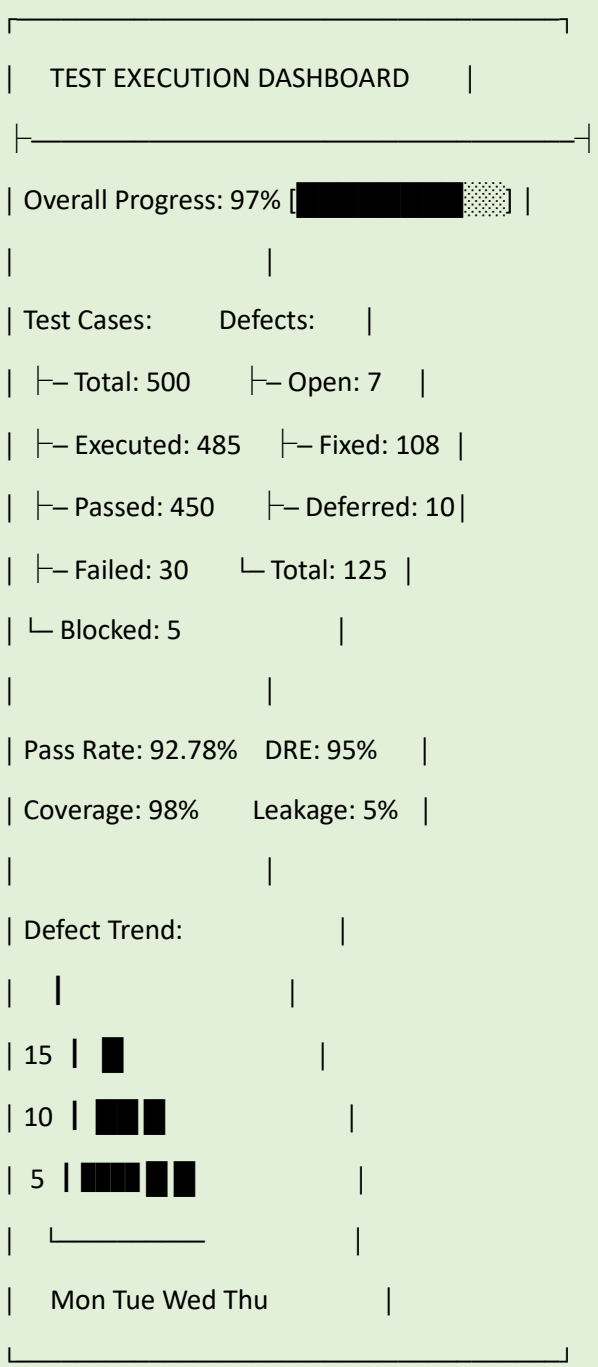
Time: 50 hours

Productivity: 2 test cases/hour

79. What is Test Metrics Dashboard?

Definition: Visual representation of key testing metrics for quick insights and decision-making.

Components:



SECTION 9: TESTING TOOLS & AUTOMATION

80. What is Test Automation?

Definition: Using automation tools to execute test cases, compare actual and expected results, and report test results automatically.

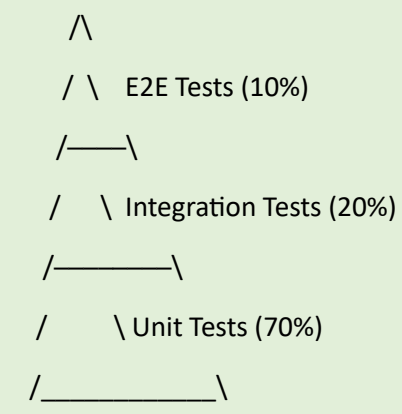
When to Automate:

- Regression testing
- Repetitive tests
- Data-driven tests
- Smoke tests
- Performance tests
- Tests across multiple environments

When NOT to Automate:

- One-time tests
- Exploratory testing
- Usability testing
- Tests still evolving
- Ad-hoc testing
- Tests with frequent UI changes

Automation Pyramid:



Benefits:

- Faster execution
- Reusable tests
- Reliable results

- Better coverage
- Cost-effective long-term
- Parallel execution
- 24/7 execution

Challenges:

- Initial investment
- Maintenance overhead
- Tool selection
- Skill requirement
- Not all tests can be automated
- False positives/negatives

81. Popular Testing Tools

Test Management:

- JIRA
- TestRail
- Zephyr
- qTest
- PractiTest

Defect Tracking:

- JIRA
- Bugzilla
- Mantis
- Redmine

Functional Automation:

- Selenium (Web)
- Cypress (Web)
- Playwright (Web)
- Appium (Mobile)
- TestComplete
- UFT/QTP

API Testing:

- Postman
- REST Assured
- SoapUI
- JMeter

Performance Testing:

- JMeter
- LoadRunner
- Gatling
- K6

Security Testing:

- Burp Suite
- OWASP ZAP
- Nessus

SECTION 10: REAL-WORLD SCENARIOS

82. What is Smoke Test Suite?

Example for E-commerce:

1. Application launches
2. Homepage loads
3. User registration works
4. User login works
5. Search functionality works
6. Add to cart works
7. Checkout process initiates
8. Payment gateway accessible
9. Database connectivity
10. API responses received

Duration: 15-20 minutes

If any fails: Build rejected

83. What is Regression Test Suite?

Example Selection Criteria:

Priority 1 (Always Execute):

- Critical business flows

- Recently changed features

- Previously failed tests

Priority 2 (Execute if Time):

- Important features

- Integration points

- Common user paths

Priority 3 (Execute Periodically):

- Minor features

- Edge cases

- Less frequently used features

84. Test Scenario Example - E-commerce Cart

TEST SCENARIO: Shopping Cart Functionality

TS_CART_001: Add Product to Cart

TS_CART_002: Update Product Quantity

TS_CART_003: Remove Product from Cart

TS_CART_004: Apply Coupon Code

TS_CART_005: Calculate Total Price

TS_CART_006: Cart Persistence

TS_CART_007: Maximum Quantity Limit

TS_CART_008: Out of Stock Handling

TS_CART_009: Guest User Cart

TS_CART_010: Registered User Cart

TS_CART_011: Cart Merge After Login

TS_CART_012: Empty Cart

TS_CART_013: Cart on Multiple Devices

TS_CART_014: Cart Timeout

TS_CART_015: Price Changes in Cart

85. Test Case Example - Detailed

TEST CASE ID: TC_CART_001_01

Scenario ID: TS_CART_001

Title: Add single product to empty cart

Module: Shopping Cart

Priority: High

Severity: Critical

PRECONDITIONS:

1. User is logged in

2. Cart is empty

3. Product in stock

4. Network connectivity stable

TEST DATA:

Product: Laptop ABC123

Price: $999.99

Quantity: 1

STEPS:

Step 1:

Action: Navigate to product detail page

Expected: Product details displayed correctly

Step 2:

Action: Verify product price and stock

Expected: Price shows $999.99, In Stock status visible

Step 3:

Action: Click "Add to Cart" button

Expected: Loading indicator appears

Step 4:

Action: Observe cart icon

Expected: Cart count updates to 1

Step 5:

Action: Verify success message

Expected: "Product added to cart" message displays

Step 6:

Action: Click on cart icon

Expected: Cart overlay/page opens

Step 7:

Action: Verify product in cart

Expected:

- Product name: Laptop ABC123
- Quantity: 1
- Price: $999.99
- Subtotal: $999.99

POSTCONDITIONS:

- Cart contains 1 item
- Database updated
- Cart count shows 1

ACTUAL RESULT: [To be filled]

STATUS: [Pass/Fail/Blocked]

EXECUTED BY: [Tester name]

EXECUTION DATE: [Date]

COMMENTS: [Any observations]

ATTACHMENTS: [Screenshots if failed]

This completes the comprehensive 100+ Functional Testing concepts guide. Each concept has been explained with clear definitions, examples, and practical applications relevant to real-world testing scenarios and interview preparation.