QA RESOURCE GUIDE

SK

Manual Testing Complete Guide

Everything you need to know — from principles to practice

A comprehensive reference covering all major manual testing concepts, life cycles, types, techniques, and metrics. Curated for QA engineers and SDETs.

40+	14	QA
TOPICS	PAGES	LEVEL

Created by QA Pulse by SK · skakarh.com

01 · FUNDAMENTALS

What is Software Testing?

A method to check whether a software application matches its requirements and to ensure it is bug/defect free.

02 · SEVEN TESTING PRINCIPLES

Core Principles

1 **Testing Shows Presence of Defects**

Testing can identify errors but cannot guarantee the software is 100% defect free.

2 **Exhaustive Testing is Impossible**

Testing every possible combination of scenarios and inputs is not feasible.

3 **Early Testing**

Testing activities should start early in the SDLC — fixing defects early is far cheaper.

4 **Defect Clustering**

80% of bugs are typically found in 20% of modules (Pareto's Principle).

5 **Pesticide Paradox**

Repeating the same test cases will eventually stop finding new bugs. Tests must evolve.

6 **Testing is Context-Dependent**

Different applications require different testing types, techniques, and approaches.

7 **Absence of Errors Fallacy**

A 99% bug-free software that doesn't meet user requirements is still unusable. Quality = meeting requirements.

03 · SDLC

Software Development Life Cycle

01 Requirement › 02 Design › 03 Build / Dev › 04 Testing › 05 Deployment › 06 Maintenance

PHASE 01

Requirement

Business Analyst / PM gathers all necessary information from the client.

PHASE 02

Design

Designing of software, programming language, database, hardware and software is decided.

PHASE 03

Build / Dev

Developers start writing the code for the application.

PHASE 04

Testing

Application is tested against requirements. Bugs found are assigned to developers.

PHASE 05

Deployment

After proper testing the application is handed over to the client.

PHASE 06

Maintenance

Ongoing maintenance is provided for the application over time.

04 · STLC

Software Testing Life Cycle

01 Req. Analysis › 02 Test Plan › 03 Test Case Dev › 04 Env Setup › 05 Execution › 06 Closure

Environments: Dev (development team) → QA/Testing (test execution) → Staging/Lab (UAT before release) → Production (real users)

05 · SDLC MODELS

Development Models

Sequential

Waterfall Model

First and basic model. Execution happens in sequence — outcome of one stage is the input of the next. No backtracking allowed. Also known as the Linear Sequential Life Cycle Model.

Modified

Iterative Model

Modified version of Waterfall. Feedback can be sent to previous phases (backtracking is possible). No phase overlapping allowed.

High Risk

Spiral Model

Combination of Waterfall and Iterative. Best for high-risk projects — reduces risks and allows changes at later stages. Quadrants: Evaluation, Risk Analysis, Coding & Testing, Requirement Analysis.

V-Model

V-Shaped Model

Also called Verification and Validation Model. A testing phase is associated with each corresponding development phase.

> Verification = static testing (are we building it right?) · Validation = dynamic testing (are we building the right thing?)

Modern

Agile Model

Combination of iterative and incremental model. Focuses on adaptability and customer satisfaction through rapid delivery. Products are divided into small incremental builds provided in iterations of 1–3 weeks. Best for changing requirements and fast delivery.

06 · TESTING TYPES

Types of Software Testing

Human Executed

Manual Testing

Test cases are executed manually by a tester without using any automated tools. Purpose: identify bugs, issues, and defects in the software.

Tool Executed

Automation Testing

Uses specific tools to execute test scripts without human interference. Enhances efficiency, productivity, and test coverage. Can fetch test data, handle implementation, and compare actual vs expected output.

07 · MANUAL TESTING TYPES

White / Black / Grey Box

Developer

White Box

Done by developers. Every line of code is checked before deploying to the testing environment.

Tester

Black Box

Done by test engineers. Checks the functionality of an application. Source code is NOT visible.

Both

Grey Box

Combination of White and Black. Tester has access to internal code AND tests functionality. Reports both failures and their code-level reason.

08 · FUNCTIONAL TESTING

Types of Functional Testing

Functional Testing verifies the system against its functional requirements. It ensures the application does what it is supposed to do.

Dev Phase

Unit Testing

Individual units or components of an application are tested in isolation. Done during the development phase.

> Example → Dev tests login page with correct credentials and confirms successful redirect.

After Unit Testing

Integration Testing

Individual units or components are combined and tested together to verify they function correctly as a group.

> Example → In Gmail, send a mail and verify it appears in the Sent Items folder.

After Integration

System Testing

Complete integrated software is tested end-to-end by the QA team. Both functional and non-functional testing is applied. Done before acceptance testing.

Final Phase

User Acceptance Testing (UAT)

Performed by the end user or client to verify/accept the software before moving to production. Done after functional, integration, and system testing.

09 · FUNCTIONAL VS NON-FUNCTIONAL

Functional vs Non-Functional

Functional Testing	Non-Functional Testing
Verifies operations and actions of an application	Verifies the **behaviour** of an application
Based on **business requirements**	Based on **performance requirements**
Examples: Unit, Smoke, Integration, Regression	Examples: Performance, Load, Stress Testing

10 · INTEGRATION TESTING

Types of Integration Testing

Incremental

Top-Down Approach

Higher level modules are tested with lower level modules. Stubs are used as temporary modules. Major design flaws can be detected early. Modu
the previous one.

Incremental

Bottom-Up Approach

Lower level modules are tested with higher level modules. Drivers are used as temporary modules. Top-level critical modules are tested last. Mod
parent of the previous one.

Non-Incremental

Big Bang Method

All modules are integrated and tested at once. Convenient for small systems; identification of defects is difficult for large systems. Testing team ha

11 · NON-FUNCTIONAL TESTING

Non-Functional Testing Types

Non-Functional Testing tests parameters such as reliability, load, performance, and accountability. Always performed AFTER functional testing.

Performance

Performance Testing

Eliminates reasons behind slow performance. Measures **Response Time** (server response to client), **Load** (N users simultaneously), and **Stability** (system under sustained load).

Capacity

Load Testing

Tests the system by applying load equal to or less than the desired load. Checks how many users can work simultaneously.

> Example → 1000 users (desired load) · Goal: 3 requests/sec

Stress

Stress Testing

Checks application behaviour under load GREATER than the desired load. Tests robustness and stability under extreme conditions.

> Example → 1100 users (above 1000 desired) · Goal: 4 req/sec

Scale

Scalability Testing

Upward: Increase users until crash point (find max capacity).
Downward: Decrease users until goal is met (find bottleneck).

Security

Security Testing

Detects security flaws by investigating system architecture from an attacker's mindset. Test cases focus on areas most likely to be attacked.

Portability

Portability Testing

Verifies the software runs on different operating systems without bugs. Also tests same OS with different hardware.

Reliability

Reliability Testing

Checks whether the system runs without failure under specified conditions for a specified time period.

Efficiency

Efficiency Testing

Examines the number of resources needed to develop the software and how many were actually used. Verifies customer requirements are satisfied.

12 · OTHER TESTING TYPES

More Testing Types

Regression Testing

Ensures new code changes do not break existing functionality. Confirms old code still works after the latest changes are applied.

Smoke Testing

Done when a new build is received to check if the build is stable enough for testing. Focuses on core functionality only.

Sanity Testing

Done after Smoke Testing to verify all bugs are fixed and no new bugs were introduced. A subset of regression testing.

Recovery Testing

Verifies software ability to recover from failures (hardware, software, network, power cut). Example: Chrome prompting to restore previous session after crash.

Alpha Testing

First end-to-end testing of a product. Performed by **internal employees** in a lab/staging environment before release to actual users.

Beta Testing

Acceptance testing performed by **real users in the real environment** to verify all functionality before final production release.

Ad Hoc Testing

Performed without planning or documentation, after formal testing is done. Tester randomly checks the software outside normal test case flow.

Monkey Testing

Software is tested using random inputs with the sole purpose of breaking the system. No rules — runs entirely on tester's instinct and experience.

Agile Testing

Testing practice that follows the principles of Agile software development — continuous, iterative testing throughout sprints.

Exploratory Testing

Allows testers to think outside the box. Testers explore the application and design tests on the fly without pre-written test cases.

13 · BUG LIFE CYCLE

Bug Life Cycle

Bug Life Cycle is the set of states a defect goes through from discovery to closure.

01 **New** NEW
Bug is found during testing and assigned status New.

02 **Assigned** ASSIGNED
Test lead approves the bug and assigns it to the developer team.

03 **Open** OPEN
Developer starts analyzing and working on the defect fix. May move to Deferred, Rejected, Duplicate, or Not a Bug.

04 **Fixed** FIXED
Developer fixes the bug and assigns it back to the tester for re-testing.

05 **Re-Testing** TESTING
Tester performs re-testing to verify the fix is correct.

06 **Re-Open** RE-OPEN
If the issue reappears after the fix, the bug is reassigned to the developer.

07 **Verified** VERIFIED
Tester confirms the defect has been fixed — status marked Verified.

08 **Closed** CLOSED
Defect no longer exists — tester changes status to Closed.

BRANCH / SPECIAL STATES

Duplicate
Bug is repeated twice or corresponds to the same existing bug concept.

Rejected
Developer feels the defect is not a genuine defect.

Deferred
Not high priority — expected to be fixed in the next release.

Not a Bug
Does not affect the functionality of the application.

14 · SEVERITY & PRIORITY

Severity vs Priority

Severity

Impact of the defect on the application. Set by the **QA Tester**.

- Blocker — cannot proceed to next module
- Critical — major feature broken
- Major — significant impact
- Minor — low impact

Priority

How soon the bug should be fixed. Set by **Test Lead / Project Manager**.

- Urgent — fix immediately
- High — fix in current sprint
- Medium — fix soon
- Low — fix when possible

15 · TEST SCENARIO VS TEST CASE

Scenario vs Test Case

Test Scenario

Gives us the idea of **what** we have to test.

> e.g. → Verify the login functionality of a Gmail account.

Test Case

Gives us the idea of **how** to test it — specific steps and data.

> e.g. → Enter valid username + valid password → verify redirect.

Test Case Parameters: Test Case ID · Test Scenario · Test Case Description · Test Steps · Prerequisite · Test Data · Expected Result · Actual Result · Status · Bug ID · Bug Status

16 · BOUNDARY VALUE ANALYSIS

Boundary Value Analysis (BVA)

BVA is a Black-Box technique that checks errors at the boundaries of an input domain.

Invalid

Below Minimum

Values less than the minimum boundary. e.g. for range 18–30: values 12, 14, 15, 16, 17.

Valid

Boundary & Range

Min boundary (18), max boundary (30), and all values within: 19, 20 ... 29.

Invalid

Above Maximum

Values greater than the maximum boundary. e.g. for range 18–30: values 31, 32, 34, 36, 40.

17 · EQUIVALENCE PARTITIONING

Equivalence Partitioning

Input data is divided into partitions of valid and invalid values. All values in a partition must exhibit the same behaviour. Test cases should cover each partition at least once.

Invalid

Invalid Partition 1

DIGITS >= 7 (e.g. 93847262) — too many digits

Invalid

Invalid Partition 2

DIGITS <= 5 (e.g. 9845) — too few digits

Valid

Valid Partition

DIGITS = 6 (e.g. 456234, 451483) — exactly 6 digits

18 · RTM

Requirement Traceability Matrix

RTM maps and traces user requirements with test cases. Ensures all requirements are tested and no functionality is unchecked. Parameters: Requirement ID · Requirement Type & Description · Test Cases with Status.

19 · DECISION TABLE TESTING

Decision Table Testing

Decision Table Testing tests system behaviour for different input combinations. Also called a Cause-Effect table.

CONDITIONS	RULE 1	RULE 2	RULE 3	RULE 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E — Error	E — Error	E — Error	H — Home

T = Correct · F = Wrong · E = Error message shown · H = Home screen shown

20 · STATE TRANSITION TESTING

State Transition Testing

State Transition Testing is a black-box technique where changes in input conditions cause state changes or output changes in the AUT.

States

The different states the software can be in. e.g. 1st Try, 2nd Try, 3rd Try, Access Granted, Account Blocked.

Transitions

Moving from one state to another — represented by arrows in the state diagram.

Events

What triggers a transition. e.g. Incorrect PIN, Correct PIN, closing a file.

ATM Example: Valid PIN on any of 3 attempts → Access Granted. Invalid PIN on 1st or 2nd try → prompt re-entry. Invalid PIN on 3rd try → Account Blocked.

21 · SMOKE VS SANITY

Smoke vs Sanity Testing

Smoke Testing	Sanity Testing
Broad approach — all parts tested	Narrow approach — specific parts tested
Measures **stability** of the system	Measures **rationality** of the system
Can be manual or automated	Done without test cases or scripts
Performed by **testers and developers**	Performed by **testers only**
Subset of **acceptance testing**	Subset of **regression testing**

22 · LOAD VS STRESS

Load vs Stress Testing

Load Testing	Stress Testing
Load ≤ Desired Load	Load > Desired Load
Finds performance of app under normal load	Finds stability/response under extreme load
Tests web-based and client-server apps	Tests sudden traffic increase
Example: 100 users at 2.5/sec (desired = 100) → PASS	**Example:** 200 extra users (300 total) at 2.5/sec → PASS (stress condition met)

23 · ALPHA VS BETA

Alpha vs Beta Testing

Alpha Testing	Beta Testing
Performed by **internal employees**	Performed by **real end users**
At developer's site / lab environment	At client's location / real environment
White box + black box techniques	Black box testing only
Longer execution cycles	Only a few weeks needed
Done before product market launch	Final test before shipping to customers

24 · KEY DEFINITIONS

Essential Concepts

Error vs Defect vs Bug vs Failure

Error = mistake in coding · **Defect** = error found by tester · **Bug** = defect accepted by dev team · **Failure** = build does not meet requirements.

Entry & Exit Criteria

Entry Criteria — prerequisite items that must be completed before testing begins. **Exit Criteria** — items that must be completed before testing can be concluded.

Authentication vs Authorization

Authentication = identifying a user to provide system access (who are you?). **Authorization** = giving permission to access resources (what can you do?).

Bug Release vs Defect Cascading

Bug Release = software handed over to testing knowing a defect is present (low severity). **Defect Cascading** = one defect triggers discovery of other defects.

Positive Testing

Tests the application using **valid input data** to verify it behaves as expected. Example: entering a valid 7-digit phone number in a numeric field.

Negative Testing

Tests the application using **invalid input** to ensure it handles errors gracefully. Also called error path testing. Example: entering letters in a numeric field.

25 · METRICS

Defect Metrics

Defect Rejection Ratio

Formula

```
DRR = (Defects Rejected / Total Defects) × 100
```

Defect Leakage Ratio

Formula

```
DLL = (Defects Missed / Total Defects) × 100
```

Defect Density

Formula

```
DD = Defect Count / Lines of Code (LOC)
```

26 · TEST PLAN VS TEST STRATEGY

Test Plan vs Test Strategy

Test Plan	Test Strategy
Detailed document with test objectives, scope, schedule, resources, risks, entry/exit criteria	High-level document validating test levels, techniques, and modules to be tested
Derived from Use Cases, SRS, Product Description	Derived from BRS (Business Requirement Spec)
Dynamic — updated when requirements change	**Static** — cannot be changed or modified

27 · SOFTWARE TESTING METRICS

Testing Metrics

Testing Metrics are quantitative measures to estimate progress, quality, productivity, and health of the testing process. "We cannot improve what we cannot measure."

Metrics Lifecycle

1. Analysis — Identify & define metrics. **2. Communicate** — Educate the team on data points. **3. Evaluation** — Capture, verify & calculate. **4. Report** — Build report & distribute to stakeholders.

Example Metric

Percentage of Test Cases Executed:

```
(Cases Executed / Total Cases Written) × 100
```

Also track: passed, failed, blocked, not executed.

28 · GLOBALIZATION & LOCALIZATION

Globalization vs Localization

Globalization Testing

Tests software developed for multiple languages to ensure it supports various languages and features globally.

> Example → google.com supports many languages for users worldwide.

Localization Testing (L10N)

Tests a specific application based on country or region. The localized product supports only precise languages usable in that specific region.

> Example → QQ.com supports only Chinese — accessible by specific countries.