📌 Manual Software Testing — Detailed Notes (Part-1)

✅ 1. Introduction to Software Testing

Definition of Software Testing:

- Software testing is the process of evaluating software to identify differences between existing and required conditions (i.e., bugs) and to ensure quality.
- It ensures that the software meets customer expectations and quality standards.

Importance:

- Prevents defects from reaching production.
- Ensures the product works as expected before delivery.
- Helps reduce development and maintenance costs by catching issues early.

🎯 2. What Is Software?

Software refers to sets of instructions or programs that tell a computer what to do. It's intangible and operates on hardware.

Types of Software:

1. **System Software:**
    - Operating systems (Windows, Linux).
    - Device drivers and utilities.
2. **Programming Software:**
    - Tools that help developers write other software (compilers, interpreters).
3. **Application Software:**
    - End-user applications (web apps, mobile apps).

🧠 3. What Is Software Testing?

Testing is the **verification and validation** of software to make sure it:

- Works correctly.
- Meets specified requirements.
- Is free from defects/bugs.

Example given in the video:
Real-life scenarios where software behaves unexpectedly, showing the need for checks and validation.

📊 4. What Is Software Quality?

Software quality is defined by:

- **Bug-free** operation.
- Delivered **on time**.
- Completed **within budget**.
- Meets customer **requirements** and expectations.
- **Maintainable** for future updates.

📌 5. Project vs Product

- **Project:**
    - Temporary endeavor to create software.
    - Has a start and end date.
- **Product:**
    - Ongoing software offered to users.
    - Updated and maintained over time.

🧪 6. Why Do We Need Software Testing?

Testing is necessary to:

- Find defects and fix them before they impact users.
- Improve reliability and performance.
- Ensure software satisfies requirements.
  Without testing, users may discover bugs that could harm reputation, cost money, or even cause failures in critical systems.

🐞 7. Error, Bug & Failure

Understanding key terms:

- **Error:** Human mistake in coding or logic.
- **Bug:** A defect or fault in software.
- **Failure:** When the software doesn't behave as expected during use.

🔍 8. Why Software Has Bugs

Common reasons include:

- **Miscommunication** between stakeholders.
- **No communication** about requirements.
- **Software complexity** growing beyond expectations.
- **Programming/logic errors.**
- **Changing requirements** during development.
- **Lack of skilled testers** or inadequate testing.

📑 9. Practical Orientation (Time Stamps)

The video covers detailed explanations with time markers:

- 00:41 – beginning of content overview.
- 08:29 – What is software?

- 10:24 – Types of software.
- 14:32 – Application software explained.
- 17:54 – Beginning of software testing topic.
- ~26:59 – Software quality discussion.
- 31:00 – Definition of project.
- 34:45 – Definition of product.
- 38:32 – Why we need testing.
- 39:49 – Error/Bug/Failure explained.
- 45:21 – Why software has bugs.

📌 10. Summary: What You Learn

By the end of the video, you should understand:

✓ What software is and its classifications
✓ Why testing is crucial in development
✓ What quality means in software context
✓ Differences between project and product
✓ What bugs are and where they come from
✓ The role testing plays in delivering reliable software

✅ Who Is This For?

This session is ideal for:

- Beginners in QA & software testing.
- Students learning foundational software concepts.
- Anyone preparing for entry-level testing roles.

Detailed Notes — Manual Software Testing (Part-2)

☐Recap of Part-1 Topics (Quick Review)

Before diving into new content, this session briefly recaps concepts from Part-1:

- **What software is and its types**

- **What software testing is and why it's needed**
- **Software quality attributes**
- **Difference between project and product**
- **Error, bugs & failures**
- **Reasons why software contains bugs**

2️⃣ SDLC — Software Development Life Cycle

Definition:
The SDLC is a structured process used by the software industry to design, develop, test, and deliver software that meets customer expectations.

📋 Purpose of SDLC

- Provides a **step-by-step framework** to produce quality software.
- Helps ensure all stakeholders (developers, testers, managers) follow organized phases.

3️⃣ The 3 "P"s in Software Projects

In any software company, three important aspects are emphasized:

1. **People** – Developers, testers, managers.
2. **Process** – Steps and workflows (SDLC).
3. **Product** – The final software delivered to the customer.

4️⃣ SDLC Phases

The SDLC includes key sequential phases:

1. **Requirement Analysis** – Understanding what the customer wants.
2. **Design** – Planning the architecture/blueprint of the software.
3. **Development (Coding)** – Actual implementation of the design.
4. **Testing** – Ensuring the software is defect-free.

5. **Deployment & Maintenance** – Launching and maintaining the software.

5️⃣ SDLC Models Covered

The video explains different SDLC models used in software projects:

⭐ 1. Waterfall Model

- **Sequential flow:** Each phase must complete before the next starts.
- **Useful for:** Projects with well-defined and unchanging requirements.
- **Advantages:** Easy to understand & manage; structured.
- **Disadvantages:** Inflexible to changes; bugs found late may be costly.

⭐ 2. Spiral Model

- Combines **iterative development** with elements of **risk analysis**.
- Each cycle (spiral) involves planning, risk analysis, engineering, and evaluation.
- Good for **large, complex projects** where risk management is critical.

⭐ 3. V-Model (Verification & Validation Model)

- A variation of Waterfall where testing processes are **planned concurrently** with development phases.
- Testing activities are mapped to corresponding development stages (e.g., Unit Testing matches Development).
- Helps catch defects earlier and improves product quality.
- **Advantages:** Testing planned early, better resource utilization.
- **Disadvantages:** Rigid like Waterfall; not ideal for evolving requirements.

6️⃣ Static vs Dynamic Testing

Testing is broadly grouped into two categories:

✓ Static Testing

- Does not involve executing the code.
- Examples: **Reviews**, **Walkthroughs**, **Inspections**.
- Helps identify defects early before software runs.

✓ Dynamic Testing

- Requires running the software to observe its behavior.
- Actual inputs are given and outputs are checked.

7️⃣ Verification vs Validation

These are fundamental quality processes in testing:

- **Verification:**
  - Checks if the product is being built *correctly*.
  - Ensures compliance with specifications.
- **Validation:**
  - Checks if the product built is *what the customer wants*.
  - Involves customer acceptance & usability.

8️⃣ Testing Method Types

The video also covers core testing method distinctions:

◯ White Box Testing

- Test design uses internal code/algorithm knowledge.
- Often performed by developers (e.g., Unit Testing).

⚫ Black Box Testing

- Test design based on system functionality without internal code knowledge.
- Typically performed by testers & customers (e.g., System Testing, UAT).

1️⃣ Advantages and Disadvantages of SDLC Models

The video discusses strengths and weaknesses of key models:

✓ Waterfall Model

- Simple and easy to follow.
- But **rigid** and difficult to adapt to change.

✓ Spiral Model

- Good for risk-based projects.
- However, **complex and cost-heavy**.

✓ V-Model

- Testing starts earlier, improving quality.
- But still **less flexible** if requirements change drastically.

✅ Key Takeaways

By the end of Part-2, you should understand:

📌 What SDLC is and why it's crucial.
📌 How different models (Waterfall, Spiral, V-Model) guide project execution.
📌 Difference between static and dynamic testing.
📌 How verification and validation help ensure software quality.
📌 The difference between white box and black box testing.

Manual Software Testing Training — Part 3 (Detailed Notes)

Source: Manual Software Testing Training Part-3 video by SDET-QA on YouTube

---

🧩 1. Static Testing vs Dynamic Testing

Static Testing

- Testing performed *without executing code*.
- Examines documents, requirement files, or code structure.
- Helps catch errors early (before execution).

- Techniques include:
  - **Review**
  - **Walkthrough**
  - **Inspection**

Dynamic Testing

- Involves *running the actual software*.
- Observes actual behavior & outputs.
- Detects defects that only appear during execution.

🧠 2. Review, Walkthrough & Inspection

These are **static testing techniques** that help improve quality early:

📝 Review

- A general evaluation of documentation, test cases, or code.
- Conducted by testers or peers.

👥 Walkthrough

- Informal review where author explains work to others.
- Helps identify issues through discussion.

🔍 Inspection

- Formal and structured examination.
- Defined roles (moderator, reader, recorder, etc.).
- Best for catching requirement/design issues before coding.

✓ 3. QA, QC & QE

These are related but distinct quality concepts in software testing:

🔷 Quality Assurance (QA)

- Process-oriented.
- Focuses on *preventing* defects.
- Deals with quality *before* execution via standards and processes.

🔷 **Quality Control (QC)**

- Product-oriented.
- Involves *detecting* defects after they occur.
- Usually done via testing practices.

🔷 **Quality Engineering (QE)**

- Broader discipline including QA + QC.
- Focuses on building quality *into* the system through engineering practices.

🧪 4. Levels of Software Testing

Testing is performed in multiple tiers to ensure software reliability:

🔷 **Unit Testing**

- Performed on individual components or functions.
- Usually done by developers.

🔷 **Integration Testing**

- Tests interaction between modules.
- Detects interface defects between integrated parts.

🔷 **System Testing**

- End-to-end testing of the *whole system*.
- Checks complete integrated software.

🔷 **User Acceptance Testing (UAT)**

- Final level of testing.
- Real users test to validate business requirements.

- Determines if software is ready for release.

📌 5. Key Concepts Covered

Here's a summary of the core ideas from this video section:

📍 Static vs Dynamic

- Static tests *don't execute* code.
- Dynamic tests *run the software*.

📍 Review Types

- Review (informal), Walkthrough (semi-formal), Inspection (formal).

📍 Quality Disciplines

- QA = *prevention*
- QC = *detection*
- QE = *engineering complete quality*

📍 Levels of Testing

- Unit → Integration → System → UAT (ensures completeness).

📌 Why These Concepts Matter

Understanding these foundational parts of manual testing helps you:

✓ Identify defects *early* using static techniques.
✓ Structure testing from small components up to full system validation.
✓ Align testing efforts with organizational quality goals (QA/QC/QE).

📌 Manual Software Testing Training — Part 4 (Comprehensive Notes)

🕐 Topics & Timeline

The video covers a wide range of **system**, **functional**, and **non-functional testing types**, plus testing categories for real-world applications:

1️⃣ System Testing

- Performed after integration testing, on the fully integrated product.
- Ensures the complete system works according to requirements.

Purpose

✓ Evaluate overall functionality
✓ Validate interactions across components
✓ Verify customer requirements are met

System Testing *includes multiple sub-types of tests* below.

2️⃣ Functional Testing

Functional testing checks *features and functionality* of software per requirements.

📌 GUI Testing (Graphical User Interface)

- Tests the **visual elements** of the software.

What GUI Testing includes

✓ Screen layout, elements placement

✓ Fonts, colors, icons

✓ Buttons, links, and menus

✓ Alignment and consistency

✓ Mandatory field indicators

✓ Tab order and flow

(*GUI checklist helps ensure everything is validated systematically.*)

📌 Other Functional Areas

Functional testing also checks:

✓ **Object properties testing** – Validate UI control properties behave as expected

✓ **Database testing (back-end testing)** – Ensure database records are correct & maintained

✓ **Error handling testing** – App correctly shows messages for invalid actions

✓ **Calculation & data manipulation testing** – Computes values correctly

✓ **Link existence & execution** – All links open and go to correct pages

✓ **Cookies & session handling** – Web sessions behave properly and data persists

---

5️⃣ Non-Functional Testing

These are tests that check *how well* the system works — beyond just features:

📌 Performance Testing

- Measures responsiveness, speed, stability under load.

📌 Security Testing

- Checks vulnerabilities, data protection, access control.

📌 Recovery Testing

- Evaluates how well the system recovers after failures.

📌 Compatibility Testing

- Ensures app works across different browsers, OS, devices.

📌 Installation Testing

- Validates installation, setup, uninstall procedures.

📌 Sanitation / Garbage Testing

- Supplies *invalid / random inputs* to observe app stability.

4️⃣ Differences Between Functional & Non-Functional Testing

- **Functional** tests *what* the system does.
- **Non-functional** tests *how* the system performs those things.

🔍 Key Takeaways for Testers

💠 System Testing is *broad and end-to-end* — ensures overall software acceptance.
💠 Functional Testing focuses on *exact requirements and user workflows*.
💠 Non-Functional Testing ensures *software performance, reliability, and user trust*.

🧠 Context Summary

In the sequence of manual testing topics this course goes:

- Theory → SDLC → Techniques → Levels → Functional/Non-Functional breakdown → deeper real-world test focus.

📌 Manual Software Testing Training — Part 5 | Detailed Notes

🧠 1. Regression Testing

Definition:
Regression testing is performed to ensure that changes (code fixes, enhancements, or updates) do **not negatively affect existing features** of the software.
Essentially, after a change is made, regression testing verifies that:

✓ Old functionality still works
✓ New bugs have not been introduced
✓ Stability of system remains intact

When it's done:

- After bug fixes
- After feature changes
- After requirement changes

Types of Regression Testing:

1. **Unit Regression Testing** – retesting individual units/modules.
2. **Partial Regression Testing** – retesting related modules after a change.
3. **Full Regression Testing** – retesting the entire application when major changes occur.

🔁 2. Re-Testing

Definition:
Re-testing is the testing activity carried out *only to verify that a specific defect has been fixed.* It focuses just on that failed test case(s).

- It uses the **same test cases** that identified the defect earlier.
- It does **not involve testing other parts** of the application.

🔍 3. Difference Between Re-Testing & Regression

Aspect	Re-Testing	Regression Testing
Purpose	Verify a specific defect fix	Ensure changes haven't broken existing functionality
Scope	Limited to that defect	Can be wide, affecting related modules
Test Cases	Only failed ones	Many test cases from test suite

Aspect	Re-Testing	Regression Testing
Frequency	After bug fix	After every change in build

🚧 4. Smoke Testing

Definition:
Smoke testing is a **shallow and broad initial test** conducted on a new build to ensure the **critical functionalities** are working.

- Also called **Build Verification Testing (BVT)**
- If smoke tests fail, the build is rejected for further testing.

Key Point:
It checks core features *quickly* without going into deep details.

✅ 5. Sanity Testing

Definition:
Sanity testing verifies whether a **small section/functionality** of the application works after minor changes or bug fixes. It is **narrower and deeper** than smoke testing.

Difference from Smoke Testing:

- Smoke = wide, shallow (basic checks)
- Sanity = narrow, deep (focused checks)

🧠 6. Exploratory Testing

Definition:
A testing approach where learning, test design, and test execution occur *simultaneously*. The tester explores the application **without predefined test cases**.

- Useful when documentation is incomplete or time is limited.
- Heavily relies on tester skill/experience.

🧪 7. Ad-Hoc Testing

Definition:
Testing done **without any formal test plans, test cases, or process**, aiming to find defects through random usage of the application.

- Often done by experienced testers
- No documentation required
- Hard to reproduce defects sometimes because of randomness

🐵 8. Monkey Testing

Definition:
A type of random testing where testers (or even automated scripts) provide **random inputs/actions** to the application to see if it breaks.

- No logic or pattern followed
- Seeks unintended faults/crashes due to chaos

📜 9. Differences: Exploratory vs Ad-Hoc vs Monkey

Testing Type	Structured or Not	Focus	Approach
Exploratory	Semi-structured	Discover new defects through insight	Adapts tests as application is explored
Ad-Hoc	Unstructured	Break system through experience	Random/manual actions
Monkey	Purely Random	Crash system through chaos	Random automated or manual input

---

🅿️ 10. Positive Testing

Definition:
Testing done with **valid inputs and valid conditions** to ensure the system *behaves as expected*.

✓ Expected path

✓ Correct data

✓ Normal workflow

❌ 11. Negative Testing

Definition:
Testing with **invalid or unexpected inputs** to ensure the application handles errors gracefully.

✓ Check validations

✓ Error messages

✓ No crashes on invalid data

📄 12. Positive vs Negative Test Cases

Positive Test Cases:

- Valid inputs
- Expected behavior
- Checks success paths

Negative Test Cases:

- Invalid or boundary inputs
- Check error handling and defensive coding

🔄 13. End-to-End (E2E) Testing

Definition:
Validates the entire application flow **from start to finish** — including UI, database, backend, integrations, and external systems — to simulate real user scenarios.

- Ensures complete system works as a whole.

🌐 14. Localization and Globalization (Internationalization) Testing

Localization Testing:

- Testing application behavior in a *specific locale*
- Checks language, currency, date formats, UI direction, etc.

Globalization / Internationalization Testing:

- Ensures software supports *multiple regions/cultures* with adaptability.
- Focuses on supporting different languages and cultural formats

---

📌 Summary of Part-5 Key Concepts

✓ Regression vs Re-testing
✓ Smoke vs Sanity
✓ Exploratory, Ad-Hoc & Monkey
✓ Positive vs Negative Testing
✓ End-to-End Testing
✓ Globalization & Localization Testing

📌 Manual Software Testing Training — Part 6 | Full Notes

🧠 1. Test Case Design Techniques (Introduction)

Test Case Design Techniques are systematic strategies used to create test data and test scenarios that maximize test coverage while minimizing the number of test cases.
They help testers efficiently explore the input domain of software and detect defects without testing every possible value.

The video covers the following key techniques:

1. **Equivalence Class Partitioning**
2. **Boundary Value Analysis**
3. **Decision Table Testing**
4. **State Transition Testing**
5. **Error Guessing**

🔷 2. Equivalence Class Partitioning (ECP)

Purpose:
To divide input data into groups (equivalence classes) where test cases from each group are expected to behave similarly. This reduces redundant test cases and improves efficiency.

Concept:

- If one value in a group shows correct behavior, then other values in the same group are assumed to behave similarly.
- Input ranges are identified as **valid** and **invalid** partitions.

Example:
If an age field accepts values 18–60:

- Valid partition ➤ 18–60
- Invalid partition ➤ <18 and >60

Then only one representative value is tested from each partition.

🔷 3. Boundary Value Analysis (BVA)

Purpose:
To focus testing on the *edge values* of input ranges where defects commonly occur.

Technique:
Test cases are written for:

- Minimum boundary
- Minimum + 1
- Maximum boundary
- Maximum – 1
- Also values just outside the boundaries (invalid)

Why:
Errors often occur right at the boundaries rather than in the center of input ranges.

🔷 4. Decision Table Testing

Purpose:
Used for testing *business rules or complex logic* where multiple conditions influence outcomes.

Concept:
A table is created listing:

- **All possible input conditions**
- **Corresponding actions**

Each unique combination becomes a test scenario, ensuring that all logical paths are tested.
Useful when many input combinations exist.

🔷 5. State Transition Testing

Purpose:
Best for applications where software moves between *different states* depending on inputs or user actions.

Concept:

- The application is modeled as a set of states and transitions between them.
- Testers verify that a state changes correctly given specific inputs/events.

Example Use:
Login attempts, shopping cart workflows, session timeouts.

🔷 6. Error Guessing

Purpose:
An *experience-based* test design technique where experienced testers use intuition to guess where defects are likely to occur.

Approach:

- No formal rules or tables.
- Testers use past knowledge and judgment to select inputs/actions.
- Common error areas are targeted (e.g., empty fields, invalid formats, unexpected actions).

Examples of Error Guessing:

- Entering text in numeric fields
- Skipping required fields
- Inputting special characters
- Random clicks or workflows that may break the app

🧠 7. Why These Techniques Matter

✓ They **reduce the number of test cases** without reducing coverage.
✓ They ensure **better detection of defects** at common fault points.
✓ They help structure exploratory and systematic testing.

These techniques are widely used in real QA teams to create **robust, efficient test suites** for functional and GUI testing scenarios.

🧠 Quick Summary of Techniques

Technique	Focus Area	Use Case
Equivalence Partitioning	Reducing redundant tests	Range-based inputs
Boundary Value Analysis	Edge testing	Min/Max limits
Decision Table	Complex logic	Many condition combos
State Transition	State change behavior	Workflow flows
Error Guessing	Intuition-based cases	Edge/error conditions

📌 Manual Software Testing Training — Part 7 | Detailed Notes

Source: *Manual Software Testing Training Part-7 – SDET-QA* YouTube video.

🧠 Overview — What This Video Covers

In this part of the manual testing series, the focus is on the **Software Testing Life Cycle (STLC)** — the step-by-step flow testers follow to plan, design, execute, and close testing activities. The main topics are:

1. **Test Planning**
2. **Test Design / Development**
3. **Test Execution**
4. **Defect Reporting & Tracking**
5. **Test Closure**

✅ 1. STLC — Software Testing Life Cycle (Definition)

The **Software Testing Life Cycle** (STLC) is a structured process used to perform software testing effectively. Unlike the SDLC (which covers the entire software development), STLC focuses **only on the testing activities** and ensures high quality delivery.

📌 2. Phases of STLC

Each phase of STLC builds on the previous one, and testers typically follow these steps in order:

📜 1. Test Planning

- **Purpose:** Define what will be tested, how testing will be done, and who will do it.
- **Key Components of a Test Plan:**
  - **Scope of testing:** What features/modules will be tested.
  - **Testing Approach:** Choice of functional vs non-functional, manual vs automation, etc.
  - **Test Strategy:** High-level approach and techniques.
  - **Resources & Roles:** Who is responsible for each task.
  - **Schedule/Timeline:** Start and end dates for testing activities.
  - **Test Environment Needs:** Environments, data requirements, tools needed.

⚙️ 2. Test Design / Development

- **Purpose:** Create **test conditions, cases, and data** needed for testing.
- **Activities:**
  - Identify test conditions from requirements.
  - Prepare detailed test cases (inputs, expected results).
  - Define test data sets for execution.
  - Set up scripts or procedures testers will follow.

▶ 3. Test Execution

- **Purpose:** Run the **actual test cases** against the application build.
- **Activities:**

- Execute test cases manually or with tools.
- Log actual results and compare with expected results.
- Record discrepancies as defects.
- Re-execute failed cases after fixes.

🐞 4. Defect Reporting & Tracking

- **Purpose:** Log issues found during execution and ensure they get fixed properly.
- **Activities:**
  - Raise defects in a bug tracking tool (e.g., Jira, Bugzilla).
  - Add clear info: steps to reproduce, severity, screenshot, environment.
  - Assign to developers and track status (Open → Fixed → Verified → Closed).
  - Re-open if issues reappear after a fix.

🛑 5. Test Closure

- **Purpose:** Review all testing activities and formally close the testing cycle.
- **Activities:**
  - Ensure all test cases executed.
  - Prepare **Test Summary Report** with results, coverage, defects found, and quality insights.
  - Share reports with stakeholders.
  - Archive test artifacts (test plan, cases, closure report).

🧩 STLC vs SDLC

- **SDLC** (Software Development Life Cycle) includes development from requirement gathering to deployment.
- **STLC** focuses *only on testing activities* within that cycle.
  Together they ensure product quality and completeness from both development and testing perspectives.

📌 Why STLC Is Important

- Ensures systematic testing — not ad-hoc or chaotic.
- Helps define responsibilities and expectations clearly.
- Improves quality by planning ahead rather than reacting to defects.
- Provides documentation that can be reviewed and audited.

📍 Summary: STLC Phases at a Glance

Phase	Activity	Output
Test Planning	Define scope, strategy & resources	Test Plan Document
Test Design	Create test cases & data	Test Cases, Test Data
Test Execution	Run and log results	Test Execution Results
Defect Reporting & Tracking	Log and monitor bugs	Defect Reports
Test Closure	Wrap up and report	Test Summary Report

🧠 Best Practices in STLC

✓ Early involvement in requirement analysis improves quality.
✓ Keep test cases clear, reusable & traceable to requirements.
✓ Use a standard bug report format to reduce ambiguity.
✓ Maintain detailed test documentation for audits and team collaboration.

📌 Manual Software Testing Training — Part 8 | Detailed Notes

Source: Manual Software Testing Training Part-8 by *SDET-QA* on YouTube 📺 — Topics listed in the video description include various test documentation concepts and defect handling fundamentals.

🧠 1. Test Plan

Definition:
A **Test Plan** is a comprehensive document that outlines the overall strategy, scope, objectives, and approach for testing a software product before execution begins. It serves as the *blueprint* for testing activities.

Key Components of a Test Plan:

- Scope of testing
- Test strategy and approach
- Testing types (functional, regression, etc.)
- Resources and roles
- Test environment requirements
- Entry and exit criteria
- Schedule and milestones
- Risk identification and mitigation

📌 2. Use Case vs Test Scenario vs Test Case

Understanding these terms is crucial in planning and documenting test activities:

🔷 Use Case

- Describes *how a user interacts* with a system to achieve a task.
- Focuses on *user goals* and steps involved.

🔷 Test Scenario

- A *high-level testable idea* derived from requirements.
- Represents a possible way the software can be tested (e.g., “Login with valid credentials”).

🔷 Test Case

- A *detailed document* describing precise test steps, input data, preconditions, and expected results to validate a specific behavior.

📋 3. Differences: Use Case, Test Scenario & Test Case

Concept	Purpose	Level
Use Case	User behavior flow	Highest
Test Scenario	What to test	Medium
Test Case	How to test and results	Detailed

🗂️ 4. Test Suite

A **Test Suite** is a collection of test cases that are grouped together because they validate related features or functionality. It helps organize test cases logically and run them in batches during execution.

📄 5. Test Case Template

A **standard template** ensures consistency in test documentation. Typical fields include:

- Test Case ID
- Description
- Preconditions
- Test Steps
- Test Data
- Expected Results
- Actual Results
- Status (Pass/Fail)
- Notes/Comments

🔗 6. Requirement Traceability Matrix (RTM)

RTM Definition:
A *traceability matrix* maps test cases back to requirements to ensure full coverage. It confirms that *every requirement* has corresponding test cases.

Purpose:
✓ Ensures all requirements are tested
✓ Helps track test coverage
✓ Aids in impact analysis when requirements change

🧪 7. Test Environment

Definition:
A **Test Environment** is the setup where testing will be executed. It includes hardware, software, networks, databases, and tools that replicate the real production environment.

Components:

- Operating systems
- Browsers & versions
- Devices
- Network configurations
- Database versions
- Test data sets

▶ 8. Test Execution

Definition:
Test execution is the phase where test cases are *actually run* against the software build. Testers record whether the software behaves as expected and log any discrepancies as defects.

Activities include:

- Running test cases
- Logging actual vs expected results

- Reporting defects
- Re-testing after fixes

🐞 9. Defects/Bugs

Definition:
A **defect (bug)** is an issue in the software that causes it to behave differently than expected based on requirements.

Contents of a Defect Report

A good bug report usually contains:

- Defect ID
- Summary/Title
- Description
- Steps to Reproduce
- Severity & Priority
- Environment
- Expected vs Actual Results
- Screenshots/Attachments

🧠 10. Defect Classification (Severity & Priority)

Understanding how defects are categorized helps development teams fix them efficiently:

🔷 Severity

Describes the **impact of the defect on the system**:

- **Critical/Blocker:** System unusable
- **Major:** Major functionality broken
- **Minor:** Small impact but incorrect behavior
- **Trivial:** Cosmetic issues

🔷 Priority

Indicates how **urgently the defect should be fixed**:

- **High:** Must be fixed immediately
- **Medium:** Fix soon but not blocker
- **Low:** Fix later, if time permits

📌 Summary of Part-8 Topics

This video covers essential aspects of **test documentation and execution fundamentals** that every manual tester must know:

✓ What a Test Plan is and how it's structured

✓ Differences between use cases, scenarios, and cases

✓ Test suite and standardized templates

✓ Requirement Traceability Matrix (RTM)

✓ Setting up a test environment

✓ Executing tests and logging outcomes

✓ Defect reporting and classification by severity & priority

📌 Manual Software Testing Training — Part 9 | Detailed Notes

Source: Manual Software Testing Training Part-9 (SDET-QA) — topics from the video description and course syllabus.

☐Defect / Bug Life Cycle

The **Defect Life Cycle** (also called *Bug Life Cycle*) tracks the stages a defect goes through from detection to closure. It helps teams manage and monitor issues effectively.

🔁 Typical Steps in Defect Life Cycle

1. **New** – Tester logs a defect when an issue is first found.
2. **Assigned** – The lead assigns it to a developer for a fix.
3. **Open** – Developer starts working on the defect.
4. **Fixed** – Developer resolves the issue and marks it as fixed.
5. **Retest** – Tester re-executes test cases to verify the fix.
6. **Closed** – If the issue is resolved successfully, the defect is closed.
7. **Reopened** – If the issue still exists around the same steps, the tester reopens it for further work.

💡 Other possible statuses:

- **Rejected/As Designed** – If it's not a bug but intended behavior.
- **Duplicate** – If the same bug has already been logged.
- **Not Reproducible** – Developer can't reproduce the issue.
- **Deferred** – Bug valid but postponed to a future release.

Understanding this lifecycle is crucial for clear communication between testers and developers.

2️⃣ Test Closure / When to Stop Testing

Since testing cannot exhaustively cover *every possible scenario*, clear criteria are defined to conclude testing.

🛑 Common Exit Criteria for Test Closure

Testing is generally stopped when:

- All planned test cases have been executed.
- The required *test pass rate* threshold (e.g., ≥95%) is met.
- Critical and high-severity defects have been fixed and retested.
- No open show-stoppers or blockages remain.
- Quality stakeholders agree and sign-off.

Test Cycle Closure Activities

- Analyze test results and metrics.
- Prepare a **Test Summary/Closure Report**.
- Document lessons learned and retest coverage.
- Archive test artifacts for future reference.

3️⃣ Software Testing Metrics

Metrics help evaluate the *quality of testing efforts* quantitatively.

📊 Common Testing Metrics

Examples include:

- **Test Execution Percentage:** % of test cases executed vs planned.
- **Test Pass Percentage:** % of executed cases that passed.
- **Defect Density:** Number of defects per module or per thousand lines of code.

- **Defect Detection Rate:** How many defects are found over a time period.
- **Defect Leakage:** Bugs found after release (in production).

Metrics help teams measure progress and quality throughout the test cycle.

4️⃣ QA / Testing Activities

QA/testing is more than just executing test cases — it includes **activities throughout the project**.

🔹 Typical Testing Activities

✓ Requirement analysis and review
✓ Creating test plans and strategy
✓ Writing test cases, test scenarios, and preconditions
✓ Test environment setup & test data preparation
✓ Test execution and defect reporting
✓ Regression testing after changes
✓ Test closure and reporting
✓ Tracking quality improvements and process enhancements

These activities ensure comprehensive quality coverage, from planning to closure.

5️⃣ Principles of Software Testing

The video also emphasizes *core principles* that guide effective testing (these are commonly referenced in manual testing education).

📜 Seven Fundamental Principles

1. **Testing shows presence of defects** – It can confirm bugs, not prove absence of them.
2. **Exhaustive testing is impossible** – You cannot test all input combinations.
3. **Early testing saves time & cost** – Find defects early in SDLC.

4. **Defect clustering** – Most defects are often found in a few modules.
5. **Pesticide Paradox** – Re-running same tests eventually misses new bugs.
6. **Testing is context dependent** – Different systems require different techniques.
7. **Absence of errors fallacy** – A bug-free product by code metrics does not mean usable or fit for purpose.

These principles help testers prioritize effort and improve testing effectiveness.

🌟 Summary — Part-9 Key Concepts

Topic	What You Learn
Defect / Bug Life Cycle	How bugs flow from detection to closure
Test Closure	When and how to stop testing
Test Metrics	Quantitative measures of testing progress
QA Activities	All tasks involved from planning to closure
Testing Principles	Guiding rules for effective testing

📌 Manual Software Testing LIVE Project — Part 1 | Detailed Notes

Source: Manual Software Testing LIVE Project Part-1 by SDET-QA on YouTube.

🔎 1. Project Introduction

The video starts with a *live project overview*, where the instructor explains that this section focuses on applying manual testing techniques in a *practical scenario*.

Goal:
To demonstrate real testing activities such as exploring the application, preparing project artifacts, creating test scenarios, and planning testing for a real system.

🧭 2. Explore AUT (Application Under Test)

Before writing test artifacts, testers must **understand the AUT** — its purpose, features, functionality, flow, and expected behavior.

Key Activities:

- Navigate through the application.
- Identify main modules and workflows.
- Understand user actions and responses.
- Note areas that require validation later.

This initial exploration helps testers build context before planning testing work.

📄 3. FRS Document (Functional Requirement Specification)

The **FRS (Functional Requirement Specification)** document describes *what the software should do* — the features, use cases, and rules.

Testers use the FRS to:
✓ Understand requirements clearly
✓ Identify what needs to be tested
✓ Track coverage via traceability later

The FRS becomes the primary reference for creating test scenarios and cases.

📝 4. Test Scenarios

Test scenarios are high-level ideas outlining *what must be tested* based on requirements.

Test Scenario Characteristics:

- One testable idea per scenario
- Derived directly from FRS and user workflows
- Focuses on application behavior rather than detailed steps

Examples of test scenarios (in general context):

- Validate login with valid credentials
- Check form field validations
- Ensure important buttons work correctly

Note: These examples illustrate the type of scenarios expected in a project context.

📋 5. Test Plan

The **Test Plan** is introduced as a documented strategy for testing the project. It outlines **how testing will be performed**, including scope, schedule, and resources.

Typical Test Plan includes:

- Scope of testing
- Features to be validated
- Testing types (functional, regression)
- Dependencies and risks
- Test environment details

Test plans ensure that all stakeholders know the testing approach and expectations.

📌 Summary — Live Project Part 1

This session begins the *hands-on* application of manual testing knowledge and introduces essential project activities:

✓ Project introduction and setup

✓ Exploring the Application Under Test (AUT)

✓ Reviewing requirements via the FRS document

✓ Identifying test scenarios based on the AUT and FRS

✓ Starting the creation of the test plan to organize testing work

These activities set the **foundation for subsequent live project parts**, where testers will write *test cases*, manage *test execution*, and log *defects* in practice.

📌 Manual Software Testing LIVE Project — Part 2 | Detailed Notes

Video: *Manual Software Testing LIVE Project Part-2* by SDET-QA on YouTube
Main Topics Covered:

1. **RTM (Requirement Traceability Matrix)**
2. **Test Cases**
   (These topics are listed in the official video description.)

📜 1. RTM — Requirement Traceability Matrix

What is RTM?

- RTM stands for **Requirement Traceability Matrix**.

- It's a **document** that maps every **requirement** of the project to its corresponding **test scenarios** and **test cases**.

Purpose of RTM

✓ To ensure **all requirements are covered** by testing.
✓ To track which requirements have test cases written and executed.
✓ Helps ensure **no requirement is missed**.
✓ Useful for regression and impact analysis when requirements change.

Key Components of RTM

A typical RTM includes columns such as:

- **Requirement ID / Description**
- **Test Scenario ID**
- **Test Case IDs**
- **Status of Execution (e.g., Pass/Fail)**
- **Comments/Remarks**

RTM provides *visibility* across requirements → scenarios → detailed test cases.

🧪 2. Test Cases

Definition

A **Test Case** is a documented set of conditions and steps used to verify whether a specific feature of the application behaves as expected. It is more detailed than a test scenario and includes:
✓ Preconditions
✓ Inputs
✓ Test steps
✓ Expected outcomes
✓ Status of execution

Purpose

- To **validate features** of the application in a repeatable way.

- To ensure clarity so that *any tester can execute the case and verify results*.
- Provides evidence of what was tested and the outcomes.

Typical Structure of a Test Case

A test case document commonly contains:

1. **Test Case ID**
2. **Test Scenario Reference**
3. **Description**
4. **Preconditions**
5. **Test Steps**
6. **Test Data**
7. **Expected Result**
8. **Actual Result / Status**
9. **Remarks / Notes**

Relation Between Components

- **RTM → Test Scenarios → Test Cases**
  Each requirement links to one or more scenarios, and each scenario has one or more test cases that cover different conditions.

📌 Summary of Live Project Part 2

This video focuses on two practical and essential artefacts in real testing projects:
✓ **RTM (Requirement Traceability Matrix)** — for tracking coverage of requirements by tests.
✓ **Test Cases** — detailed executed scripts that validate specific functionality based on requirements.

These are *foundation documents* that every manual tester creates when working on a real project. They ensure that testing is organized, measurable, and traceable back to business needs.

📌 Manual Software Testing LIVE Project — Part 3 | Detailed Notes

Source: *Manual Software Testing LIVE Project Part-3* by *SDET-QA* on YouTube 🎓 — this video continues the real-world testing project started in earlier parts.

🔍 1. Test Cases Review

What this means

At this stage in the live project, the instructor goes over the **test cases** that were prepared earlier in Part 2 of the project. This includes reviewing:

✓ test case structure
✓ test steps
✓ expected results
✓ coverage of scenarios

The goal is to **validate and refine test cases** so they are ready for execution.

⚙️ 2. Environment Setup / Test Bed

Why it's important

Before executing any tests, you must set up a **test environment** (sometimes called a *test bed*) where the application will run. This includes:

✓ Hardware and OS configuration
✓ Browsers and versions (if web application)
✓ Network settings
✓ Installed software prerequisites
✓ Correct test data setup

The instructor demonstrates how to prepare and confirm that the environment is ready for actual testing.

🧱 3. Build Deployment / Development & Test Build

Understanding builds

In real projects, QA receives a **build** from the development team. A *build* is a compiled version of the application that testers will execute test cases against.

The video explains:

- What a build is
- How QA gets the build
- How to confirm the build version
- *When* testing begins (usually after smoke tests pass)

🔥 4. Smoke & Sanity Testing

Smoke Testing

- A **preliminary test** performed on each new build to ensure critical functions work
- It's shallow and quick
- If smoke testing fails, the build is rejected and sent back to development

Example areas for smoke testing might include:

✓ launching the application

✓ basic navigation

✓ login/signup workflows

Sanity Testing

- A **focused testing** on specific functionality after minor changes or fixes
- Narrower in scope than smoke testing

Sanity tests ensure that specific fixes or new features work before deeper testing continues.

📌 Summary — Live Project Part 3

This video covers foundational steps that bridge *planning* and *execution* in a real manual testing project:

1. **Reviewing Test Cases** — validating that all test cases are correct and ready for execution.
2. **Setting Up the Test Environment** — preparing hardware, software, and test data.
3. **Understanding Build Delivery and Deployment** — recognizing how QA receives build versions and when execution starts.
4. **Smoke & Sanity Testing** — first checks on the build to determine readiness for further testing.

These steps are critical in a real QA workflow because they ensure that tests are executed on stable builds in correct environments. Next parts of the live project will likely move into **actual test execution**, **defect logging**, and **results reporting**.

📌 Manual Software Testing LIVE Project — Part 3 | Detailed Notes

Source: *Manual Software Testing LIVE Project Part-3* by *SDET-QA* on YouTube 🎓 — this video continues the real-world testing project started in earlier parts.

🔍 1. Test Cases Review

What this means

At this stage in the live project, the instructor goes over the **test cases** that were prepared earlier in Part 2 of the project. This includes reviewing:

✓ test case structure
✓ test steps
✓ expected results
✓ coverage of scenarios

The goal is to **validate and refine test cases** so they are ready for execution.

⚙️ 2. Environment Setup / Test Bed

Why it's important

Before executing any tests, you must set up a **test environment** (sometimes called a *test bed*) where the application will run. This includes:

✓ Hardware and OS configuration
✓ Browsers and versions (if web application)
✓ Network settings
✓ Installed software prerequisites
✓ Correct test data setup

The instructor demonstrates how to prepare and confirm that the environment is ready for actual testing.

🧱 3. Build Deployment / Development & Test Build

Understanding builds

In real projects, QA receives a **build** from the development team. A *build* is a compiled version of the application that testers will execute test cases against.

The video explains:

- What a build is
- How QA gets the build
- How to confirm the build version
- *When* testing begins (usually after smoke tests pass)

🔥 4. Smoke & Sanity Testing

Smoke Testing

- A **preliminary test** performed on each new build to ensure critical functions work
- It's shallow and quick
- If smoke testing fails, the build is rejected and sent back to development

Example areas for smoke testing might include:
✓ launching the application
✓ basic navigation
✓ login/signup workflows

Sanity Testing

- A **focused testing** on specific functionality after minor changes or fixes
- Narrower in scope than smoke testing

Sanity tests ensure that specific fixes or new features work before deeper testing continues.

📌 Summary — Live Project Part 3

This video covers foundational steps that bridge *planning* and *execution* in a real manual testing project:

1. **Reviewing Test Cases** — validating that all test cases are correct and ready for execution.
2. **Setting Up the Test Environment** — preparing hardware, software, and test data.
3. **Understanding Build Delivery and Deployment** — recognizing how QA receives build versions and when execution starts.
4. **Smoke & Sanity Testing** — first checks on the build to determine readiness for further testing.

These steps are critical in a real QA workflow because they ensure that tests are executed on stable builds in correct environments. Next parts of the live project will likely move into **actual test execution**, **defect logging**, and **results reporting**