## **G-1 Group – Recap**

**Date:- 23rd July 2025**

|  |  |
| --- | --- |
| **Group Participants** | Points |
| **Jyotirmay Chowdhury(coordinator) - 30639** | **1-2** |
| **Nandhu -30400** | **3-4** |
| **Abhigyan Maji- 30382** | **5-6** |
| **Shikha - 31136** | **7-8** |
| **Gourab Das - 30392** | **9-10** |

**Topic:- Introduction to Software Testing**

**Introduction to Software Testing**

**What is Software Testing?**

Software testing is the process of evaluating a software application to find out if it meets the specified requirements and to identify any defects or bugs. It's about verifying that the software does what it's supposed to do, and that it doesn't do anything it's not supposed to do.

**Essentials Understanding:**

**· Quality Assurance:** Testing is a crucial part of quality assurance (QA), ensuring the product's reliability and user satisfaction.

**· Defect Detection:** The primary goal is to identify and report defects, allowing them to be fixed before the software reaches users.

**· Risk Mitigation:** Testing helps in mitigating risks associated with software failures, which can lead to financial losses, reputational damage, or even safety issues.

**Importance of Software Testing:**

**· Delivers Quality Product:** Ensures the software is high-quality, reliable, and performs as expected.

**· Customer Satisfaction:** A well-tested product leads to satisfied customers and a good user experience.

**· Cost-Effective:** Finding and fixing bugs early in the development cycle is much cheaper than fixing them after release.

**· Security:** Identifies vulnerabilities that could be exploited, protecting user data and system integrity.

**· Reputation:** A bug-free and efficient product enhances the company's reputation.

**Fundamentals of Software Testing**

The core principles that guide effective software testing:

**1. Testing shows presence of defects, not absence:** Testing can only prove that defects exist, not that the software is entirely bug-free.

**2. Exhaustive testing is impossible:** It's practically impossible to test all possible inputs, conditions, and paths in a complex system.

**3. Early testing:** Testing should start as early as possible in the software development life cycle.

**4. Defect clustering:** A small number of modules usually contain most of the defects.

**5. Pesticide paradox:** If the same tests are repeated over and over again, eventually these test cases will no longer find new bugs. Tests need to be regularly reviewed and updated.

**6. Testing is context-dependent:** The approach to testing depends on the type of software, its purpose, and the risks involved.

**7. Absence-of-errors fallacy:** Even if no defects are found, the software might still not be useful if it doesn't meet the user's needs or expectations.

**Types of Software Testing**

As seen in the provided image, software testing can be broadly categorized:

**· Manual Testing:**

o Performed by human testers without automation tools.

o Relies on human observation, judgment, and critical thinking.

o Good for exploratory testing and usability testing.

o Can be time-consuming and prone to human error for repetitive tasks.

**· Automation Testing:**

o Uses specialized software tools to execute tests and compare actual results with expected results.

o Ideal for repetitive tasks, regression testing, and large test suites.

o Faster execution and more accurate results for automated tasks.

o Requires initial investment in tools and scripting knowledge.

**Under Manual and Automation Testing, we have:**

**· White Box Testing:**

o Also known as "Clear Box" or "Glass Box" testing.

o Tests the internal structure, design, and implementation of the software.

o Requires programming knowledge to understand the code logic.

o Focuses on ensuring internal operations are performed according to specifications.

**o Example:** Unit Testing.

**· Black Box Testing:**

o Tests the functionality of the software without knowing its internal code structure.

o Focuses on inputs and outputs, as well as the user's perspective.

o Based on requirements and specifications.

**o Examples:** Functional Testing, Non-Functional Testing (Usability, Performance, Compatibility).

**· Gray Box Testing:**

o A combination of white box and black box testing.

o Testers have partial knowledge of the internal structure, often limited to high-level design or database schemas.

o Helps in designing more effective test cases by understanding both the user's perspective and some internal workings.

**Functional Testing**

This type of black box testing verifies that each function of the software application operates in conformance with its specific requirements. It ensures that the software performs its intended functions correctly.

**· Unit Testing:**

o Tests individual components or modules of the software in isolation.

o Usually performed by developers during the coding phase.

o Aims to ensure that each unit of code works correctly.

**· Integration Testing:**

o Tests the interactions between different integrated modules or components of the software.

o Aims to expose defects in the interfaces and interactions between modules.

o Can be performed using various approaches (e.g., top-down, bottom-up, big bang).

**· System Testing:**

o Tests the complete and integrated software system to evaluate the system's compliance with its specified requirements.

o Tests the entire system as a whole, from end-to-end.

o Often performed in an environment that closely resembles the production environment.

**· User Acceptance Testing (UAT):**

o The final stage of testing, performed by the end-users or clients.

o Verifies that the software meets the business requirements and is fit for purpose in the real-world scenario.

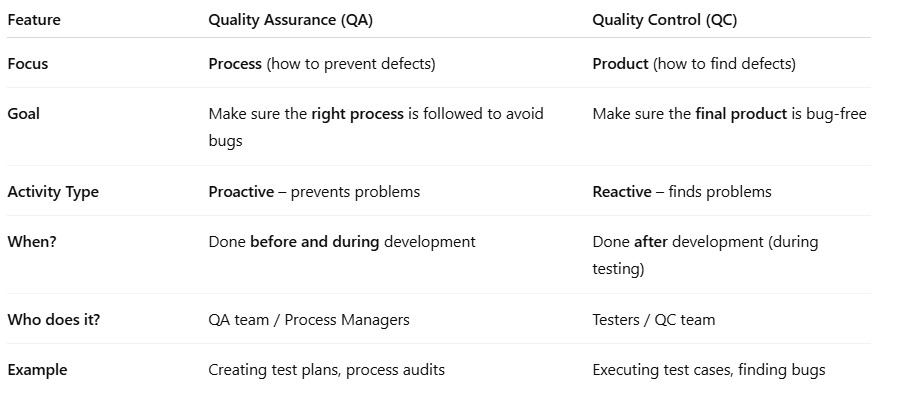
o Ensures that the software is acceptable to the users before release.

**Quality Assurance (QA) and Quality Control (QC):**

**· Quality Assurance (QA):** Focuses on the *process* of preventing defects. It's proactive and aims to build quality into the product from the start (e.g., defining standards, reviews, training).

**· Quality Control (QC):** Focuses on the *product* to identify defects. It's reactive and involves activities like testing, inspection, and defect detection after the product is built.

**Difference between Quality Assurance(QA) and Quality Control (QC):**



**Non-Functional Testing**

This type of black box testing checks non-functional aspects of the software, such as performance, reliability, usability, and security. It focuses on how well the system performs, rather than just what it does.

**· Compatibility Testing:**

o Evaluates how well the software performs in different environments (e.g., operating systems, browsers, devices, network configurations).

o Ensures the software works consistently across various platforms.

**· Performance Testing:**

o Evaluates the speed, responsiveness, and stability of the software under various workloads.

o Includes:

**§ Load Testing:** Checks system behaviour under expected load.

**§ Stress Testing:** Checks system behaviour under extreme or peak load.

**§ Scalability Testing:** Checks the system's ability to handle increasing user load or data volume.

**· Usability Testing:**

o Evaluates how easy and user-friendly the software is to use.

o Focuses on the user experience, intuitiveness, and efficiency of the interface.

o Often involves real users interacting with the system.

**· Security Testing:**

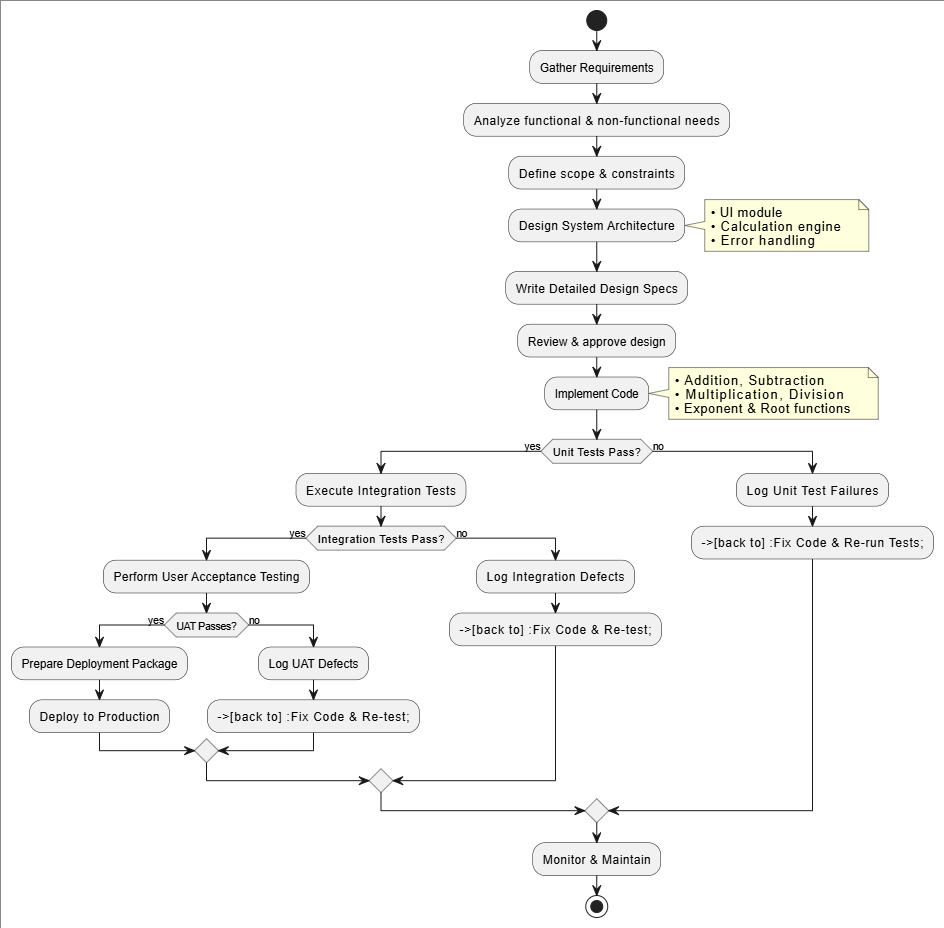
o Identifies vulnerabilities and weaknesses in the software that could be exploited by malicious attacks.

o Aims to ensure data confidentiality, integrity, and availability.

o Includes penetration testing, vulnerability scanning, and security audits.

**Software Testing Life Cycle (STLC)**

The **Software Testing Life Cycle (STLC)** is a process that verifies whether the Software Quality meets the expectations or not. STLC is an important process that provides a simple approach to testing through the step-by-step process, which we are discussing here. Software Testing Life Cycle (STLC) is a fundamental part of the Software Development Life Cycle (SDLC).



**1. Impact Assessment (Requirement Analysis):**

**o Description:** This initial phase involves understanding the requirements from stakeholders and analysing their impact on the testing effort. Information is gathered to define the scope of testing.

**o Activity:** Reviewing requirement specifications, understanding the system, identifying testable requirements.

**2. Test Planning:**

**o Description:** All parties involved sit together to prepare a detailed testing schedule, define the testing process, and determine expected outcomes. This is where the strategy for testing is laid out.

**o Activity:** Defining test objectives, scope, strategy, resources, environment, and entry/exit criteria. Estimating effort and timeline.

**3. Daily Scrums (Test Case Development):**

**o Description:** While the image mentions "Daily Scrums" as a phase, in the context of STLC, this phase often involves the actual design and development of test cases and test scripts. Agile teams attend daily stand-ups to update on testing status and set priorities.

**o Activity:** Creating detailed test cases based on requirements, preparing test data, and setting up the test environment. Daily scrums facilitate communication and progress tracking.

**4. Test Agility Review (Test Execution):**

**o Description:** Stakeholders regularly evaluate the creation progress to see if it's going according to plan. This phase is where the developed test cases are executed. Defects are identified, reported, and tracked.

**o Activity:** Running test cases, recording results, logging defects, retesting fixed defects, and conducting regression testing. Regular reviews ensure alignment with goals.

**5. Release Readiness (Test Cycle Closure):**

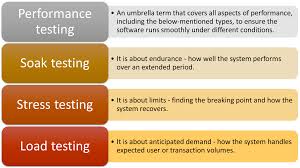
**o Description:** Final check-ups are conducted to decide whether developed features are ready for release or need to turn back to previous stages. This phase involves summarizing the testing efforts.

**o Activity:** Evaluating test results against exit criteria, preparing test summary reports, assessing release readiness, and conducting a lessons learned session for future improvements.

**Soak Testing:**

**Definition**

Soak testing is a performance testing technique that measures a system’s stability and resource usage by applying a realistic load continuously over an extended period.

****

**Objective / Description**

Performance Testing Overview

The diagram presents four primary categories of performance testing, each targeting different aspects of system behavior under load:

1. Performance Testing (Umbrella)

* Encompasses all performance-related evaluations.
* Ensures applications meet responsiveness, stability, and scalability requirements.

2. Load Testing

* Applies expected user or transaction volumes.
* Verifies system behavior under normal peak loads.
* Monitors response time, throughput, and resource utilization.

3. Stress Testing

* Pushes the system beyond its anticipated capacity.
* Identifies the breaking point and examines failure modes.
* Assesses recovery and failover mechanisms.

4. Soak Testing

* Applies a realistic load over an extended period (hours to days).
* Detects memory leaks, resource depletion, and performance degradation.
* Validates long-term stability and reliability.

**Conclusion**

Soak testing reveals long-term issues like memory leaks and performance degradation, ensuring the software remains reliable during sustained operation.

**Assignment Questions:**

**Q1: Impact Assessment**

Impact assessment is a systematic process that identifies, predicts, and evaluates the potential effects of adding or changing calculator features on functionality, performance, and user experience to guide risk mitigation and decision-making.

**Q2: Purpose of Test Planning**

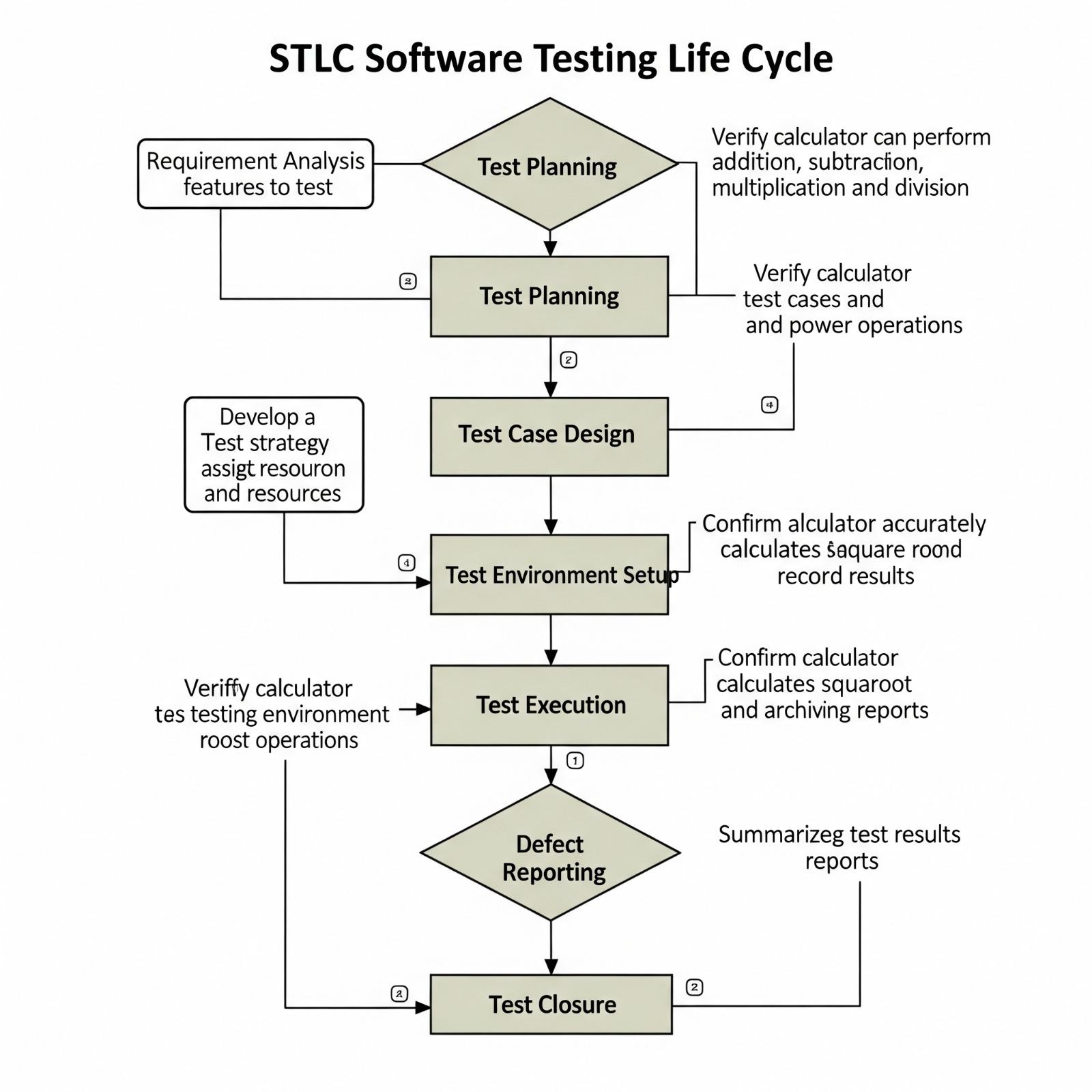
The exact purpose of test planning is to define the scope, objectives, resources, schedule, deliverables, and risk mitigation strategies for testing the calculator so that coverage is clear and execution stays organized.

**Q3: Work of Daily Scrums**

Daily scrums are brief, time-boxed meetings where the team synchronizes on what was done yesterday, plans what will be done today, and raises any impediments blocking calculator development.

**STLC Pattern for Product Calculator**

Step-by-Step STLC Procedure for a Calculator Application



**1. Requirement Analysis**

* List all functional requirements:
  + Basic arithmetic operations (+, –, ×, ÷)
  + Decimal input/output, negative numbers
  + Memory functions (M+, M–, MR, MC)
* Identify non-functional requirements:
  + Performance (response <100 ms per operation)
  + Accuracy to 8+ decimal places
  + UI responsiveness and accessibility
  + Error handling (divide by zero, invalid input)
* Clarify ambiguities with stakeholders and finalize the Software Requirements Specification (SRS).
* Create a Requirement Traceability Matrix (RTM) mapping each requirement to tests.

**2. Test Planning**

* Define testing scope and objectives: unit tests, integration tests, UI tests, performance checks.
* Select tools and frameworks:
  + Pytest for unit and integration testing
  + Selenium WebDriver for UI automation
  + JMeter or Locust for performance/load testing
* Estimate timelines, resources, and responsibilities.
* Establish entry and exit criteria:
* Entry: SRS approved, environment ready
* Exit: All critical tests passed, no P1/P2 defects open

**3. Test Case Design**

* Identify test scenarios and categories:
  + Functional: each arithmetic operation, memory workflows
  + Edge cases: large numbers, floating-point precision, overflow
  + Negative cases: non-numeric input, empty input, chain of operations
* Write detailed test cases with:
  + ID, preconditions, test steps, expected results
* Review and baseline test cases with peers and stakeholders.

**4. Test Environment Setup**

* Provision test machines matching target platforms (Windows/macOS/Linux).
* Install calculator build, dependencies, drivers, and test frameworks.
* Configure test data (input scripts, data files).
* Verify environmental health by executing a smoke-test suite of core operations.

**5. Test Execution**

* Run unit tests via Pytest and capture results.
* Execute automated UI tests in headless and headed browsers.
* Perform performance tests under varying loads.
* Log defects in a tracking tool (e.g., JIRA) with detailed repro steps and screenshots.
* Update the RTM and track test coverage and pass rates in real time.

**6. Test Closure**

* Verify all exit criteria are met:
  + No open critical defects
  + Required coverage metrics achieved
* Collect and analyze test metrics: total tests, pass/fail, defect density.
* Prepare Test Closure Report and archive test artifacts (plans, cases, logs).
* Conduct a lessons-learned session to improve future cycles.

**Design test planning and execution of a software product that checks the validity of any other product**

**Validity check to be done on the parameter - Expiry Date**

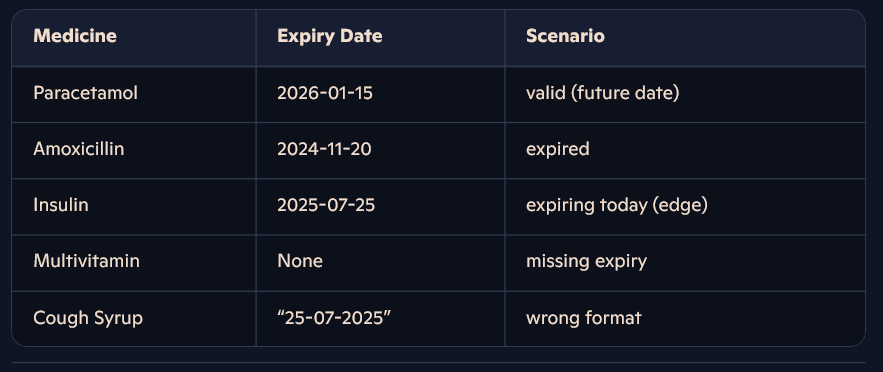
**1. Define Objective**

- Identify medicines whose expiry date is before today

- Fail products if expiry\_date < current\_date

- Pass products if expiry\_date >= current\_date

**2. Collect Sample Test Data**

****

- Paracetamol – Expiry: 2026-01-15 (valid future date)

- Amoxicillin – Expiry: 2024-11-20 (already expired)

- Insulin – Expiry: 2025-07-25 (expiring today)

- Multivitamin – Expiry: None (missing date)

- Cough Syrup – Expiry: “25-07-2025” (invalid format)

**3. Choose Test Environment**

- Language: Python (for date parsing and comparison)

- Framework: Pytest (optional, for structured test cases)

- Tools: Date libraries, test runner, logging utilities

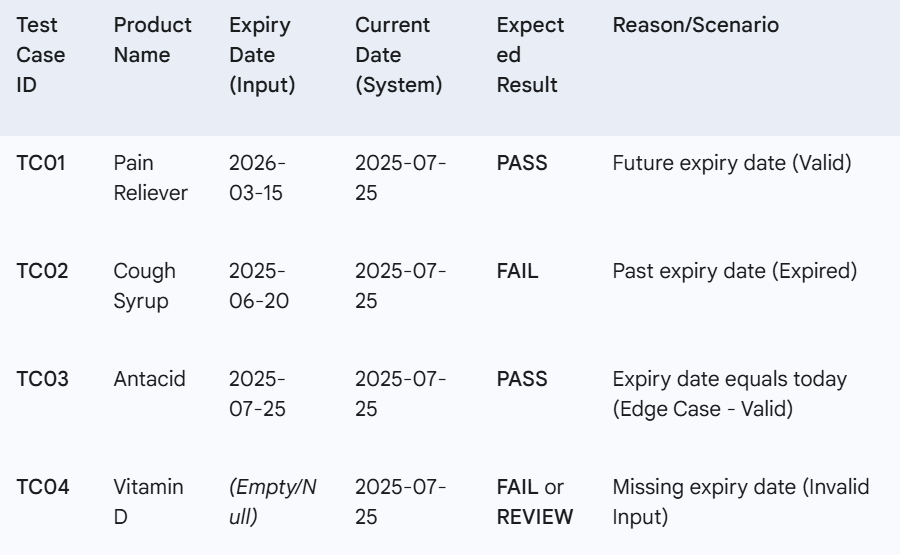
**4. Build Expiry Validation Logic**

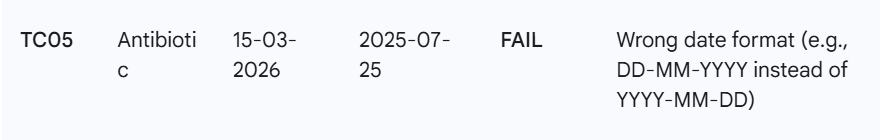
- Parse expiry date string using a consistent format (YYYY-MM-DD)

- Compare parsed date to the current system date

- Return a boolean or status flag (valid vs. invalid)

- Handle parsing errors and null values gracefully





**5. Design Test Cases**

- TC01: Future expiry date → Expected “PASS”

- TC02: Past expiry date → Expected “FAIL”

- TC03: Expiry date equals today → Expected “PASS” (edge case)

- TC04: Missing expiry date → Expected “FAIL” or “REVIEW”

- TC05: Wrong date format → Expected “FAIL” (format error)

**6. Plan Test Execution**

- Iterate through each medicine record

- Apply expiry validation logic

- Log or record the result as PASS/FAIL

- Include timestamp, product name, expiry value, and status

**7. Extend with Automation**

- Parameterize test cases in Pytest or another test framework

- Use fixtures for sample medicine lists

- Generate automated test reports after each run

**8. Reporting & Alerts**

- Summarize results in CSV, JSON, or database table

- Create Power BI or Excel dashboards to visualize pass/fail trends

- Trigger email or messaging alerts for failed/near-expiry items

**9. Edge Case Handling**

- Null expiry dates → auto-fail and flag for manual review

- Non-parseable strings → log as “format error” and fail

- Future data anomalies (e.g., year far ahead) → review data consistency

**10. Review & Optimize**

- Validate that all realistic medicine scenarios are covered

- Scale logic for bulk data ingestion (CSV/XLSX files)

- Profile performance on large catalogs and optimize date comparisons

- Periodically audit test data and update scenarios as new edge cases emerge