Test Execution, Defect Management, and Test Closure – Practical QA Guide

A comprehensive training programme designed for QA professionals to master real-world testing execution, defect handling, reporting, and closure activities as practised in industry projects.

SECTION 1

What Is Test Execution?

Test execution is the systematic process of running test cases against the application under test to validate whether actual results match expected outcomes. It represents the core testing phase where testers interact with the software, document findings, and identify defects.

During test execution, testers follow predefined test cases, record actual behaviour, compare results against expected outcomes, and update test case statuses accordingly. This phase bridges the gap between test preparation and defect reporting.

Key Activities

- Running test cases systematically
- Comparing actual vs expected results
- Logging defects when deviations occur
- Updating test case status
- Documenting test evidence

Entry Criteria for Test Execution

Before commencing test execution, specific prerequisites must be satisfied to ensure the testing environment is stable and ready. Entry criteria serve as quality gates that prevent premature testing and wasted effort.

Test Environment Ready

All required hardware, software, and network configurations are installed, configured, and accessible to the testing team.

Test Data Prepared

Valid, invalid, and boundary test data sets are created and available in the test environment databases.

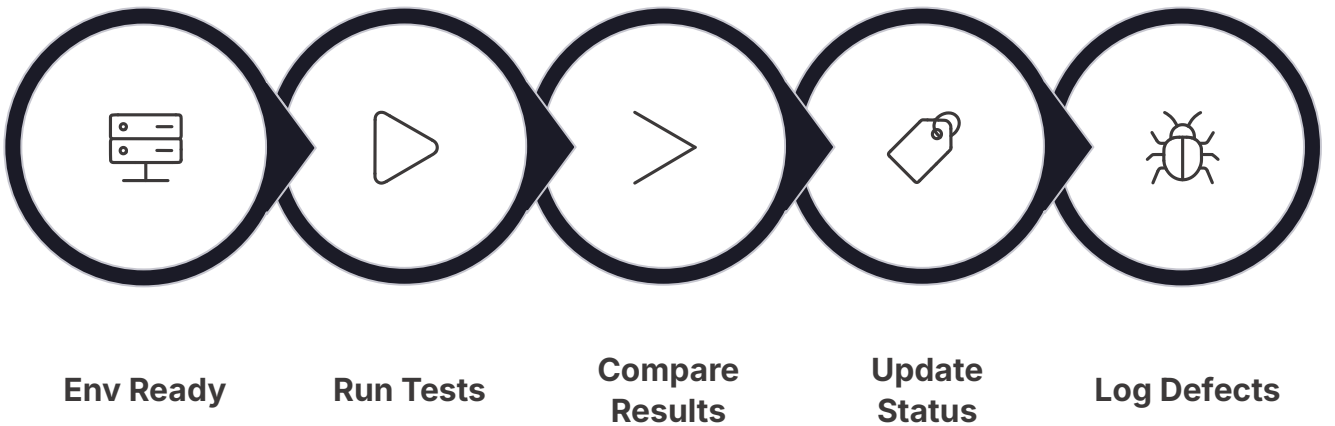
Test Cases Reviewed

All test cases have been reviewed, approved, and baselined by test leads or senior testers.

Build Deployed

A stable application build has been deployed to the test environment and smoke testing has passed.

Test Execution Process Flow



The test execution process follows a structured workflow that ensures comprehensive coverage and accurate reporting. Each step builds upon the previous one, creating a systematic approach to quality validation.

Testers must verify environment readiness before executing any test cases. During execution, they meticulously compare actual outcomes against expected results, updating statuses and logging defects when discrepancies arise. The cycle continues with retesting fixed defects and performing regression testing.

Test Execution Steps in Detail

01

Test Environment Readiness

Verify servers are running, databases are accessible, test data is loaded, and all integrations are functional. Conduct smoke tests to confirm basic functionality.

02

Test Case Execution

Execute test cases following the documented steps precisely. Enter inputs, perform actions, and navigate through the application as specified in test case documents.

03

Actual vs Expected Comparison

Compare the application's actual behaviour against expected results defined in test cases. Note any deviations, unexpected errors, or missing functionality.

04

Test Case Status Update

Update each test case with appropriate status: Pass (actual matches expected), Fail (deviation observed), or Blocked (cannot proceed due to environment or dependency issues).

05

Defect Logging

For failed test cases, log detailed defects in the defect tracking system, including steps to reproduce, actual results, expected results, severity, and priority.

06

Retesting & Regression

After developers fix defects, perform retesting to verify the fix. Execute regression test suites to ensure new changes haven't broken existing functionality.

Test Execution vs Test Design

Test Design Phase

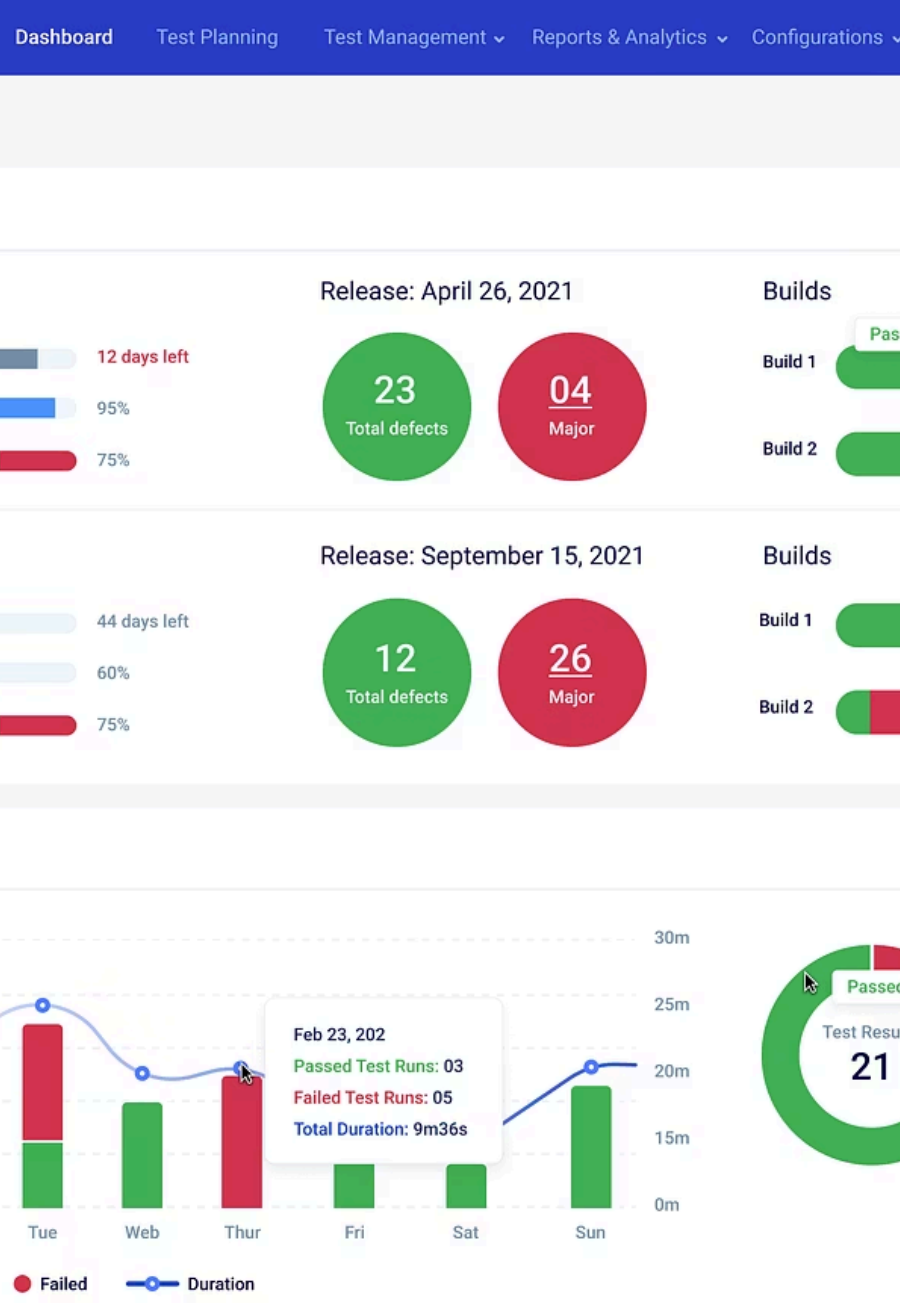
Test design occurs before execution and focuses on creating test artefacts. Activities include analysing requirements, identifying test scenarios, writing detailed test cases, preparing test data specifications, and designing test approaches.

The design phase answers "what to test" and "how to test" without actually running tests. It produces reusable test assets that guide execution activities.

Test Execution Phase

Test execution implements the design by actually running test cases against the application. Activities include setting up environments, executing test steps, comparing results, logging defects, and updating test statuses.

The execution phase answers "did it work as expected?" through hands-on testing. It produces evidence, defects, and execution reports based on designed test cases.



Test Execution Outputs

Updated Test Cases

Test cases with execution status, actual results, timestamps, and tester names documented for traceability.

Defect Reports

Detailed defect logs with reproducible steps, severity, priority, screenshots, and logs for development team.

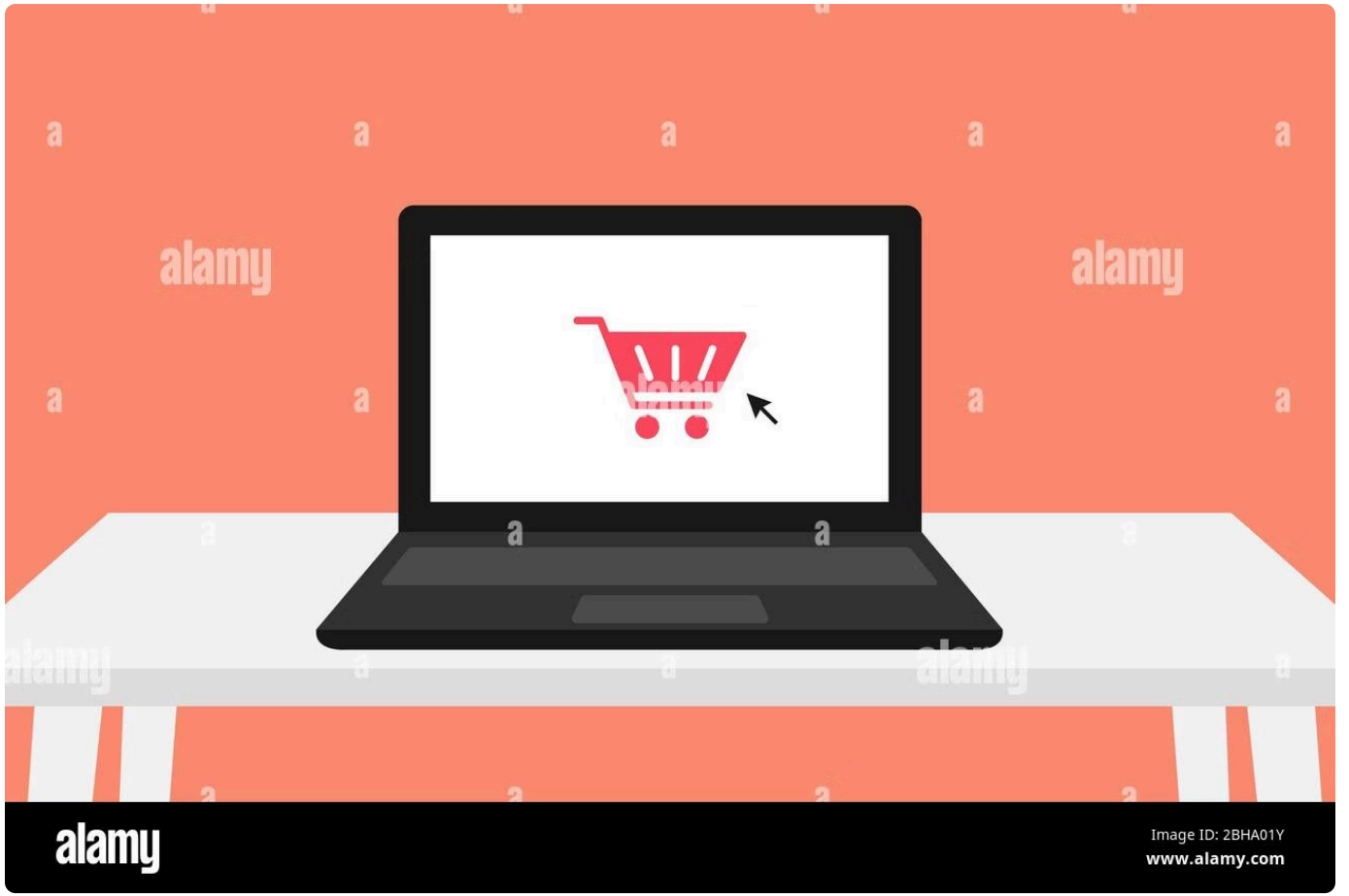
Test Execution Reports

Daily or periodic reports showing test progress, pass/fail statistics, blocked cases, and defect summaries.

Test Evidence

Screenshots, log files, database snapshots, and video recordings that support test results and defects.

Real-World Example 1: E-Commerce Checkout



Scenario

Testing the checkout process for an e-commerce platform. Test case: "Verify user can complete purchase with valid credit card."

Execution Steps

1. Add items to cart
2. Proceed to checkout
3. Enter shipping details
4. Enter payment information
5. Click "Place Order"

Outcome

Expected: Order confirmation displayed, email sent, payment processed.
Actual: Payment processed but confirmation email not sent.
Status: FAIL – Defect logged with severity High, priority High.

Real-World Example 2: Mobile App Login

Scenario

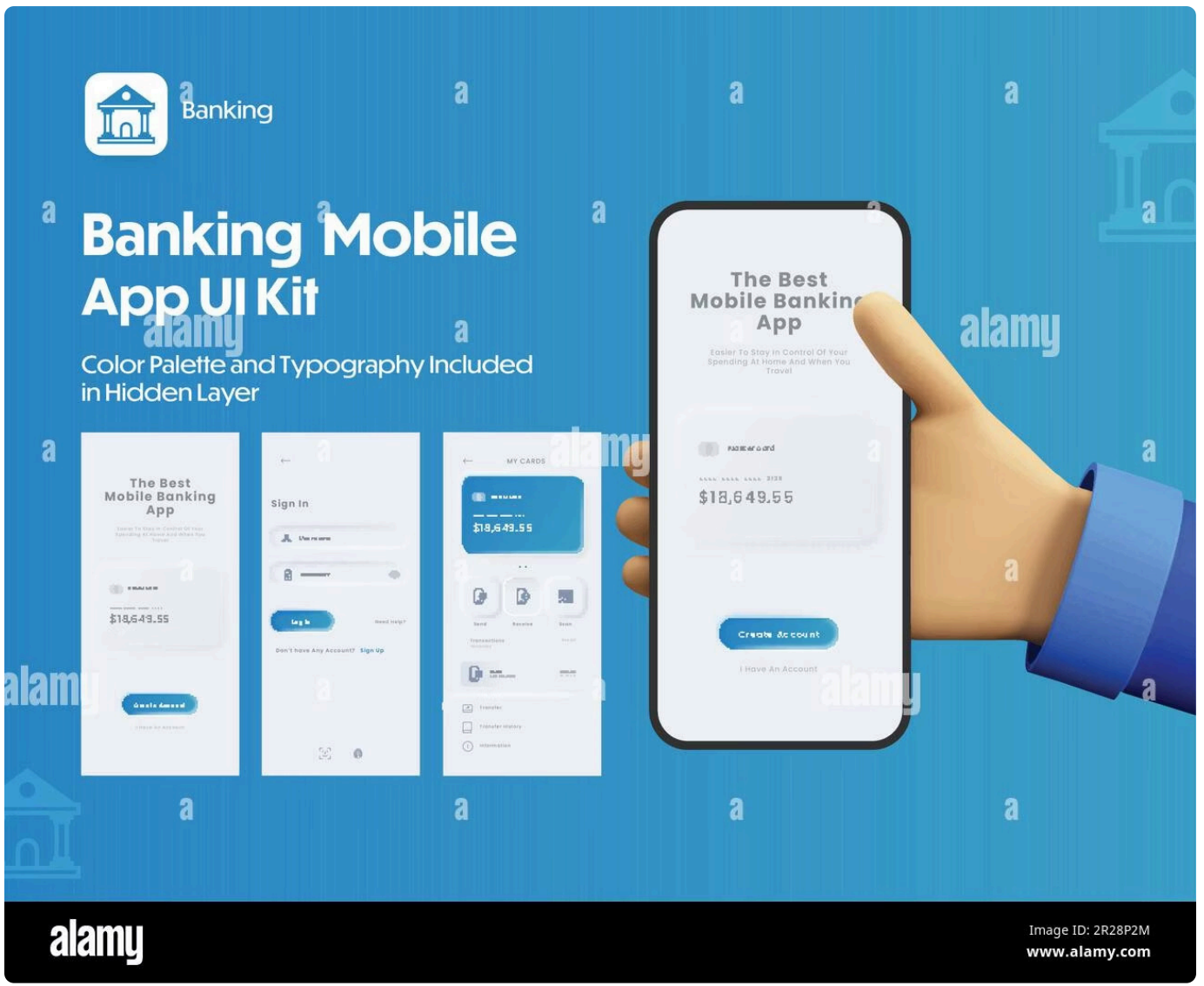
Testing login functionality for a mobile banking application. Test case: "Verify account locks after five incorrect password attempts."

Execution Steps

1. Open mobile app
2. Enter valid username
3. Enter incorrect password five times
4. Observe application behaviour

Outcome

Expected: Account locked, message displayed: "Account locked due to multiple failed attempts."
Actual: Account locked but generic error message displayed: "Login failed."
Status: FAIL – Defect logged with severity Minor, priority Medium (usability issue).



Case Study: Missed Execution Step Causes Production Defect

The Incident

A healthcare application released to production experienced critical failures during patient appointment booking. Investigation revealed testers skipped testing with concurrent users during test execution, focusing only on single-user scenarios.

The application worked perfectly in single-user tests but failed when multiple patients attempted to book the same appointment slot simultaneously. The root cause was a missing concurrency test execution step.

Lesson Learned

Always execute test cases covering realistic usage patterns, including concurrent access, peak load conditions, and multi-user scenarios.

Prevention

Include non-functional test execution steps in test plans, not just functional scenarios. Verify entry criteria include performance test environment readiness.

Quick Knowledge Check: Test Execution

1

Question 1

What is the correct order of test execution activities?
A) Log defects → Execute tests → Update status → Compare results
B) Execute tests → Compare results → Update status → Log defects
C) Update status → Execute tests → Log defects → Compare results
D) Compare results → Execute tests → Log defects → Update status

Answer: B – Execute tests first, then compare actual vs expected, update status, and log defects if needed.

2

Question 2

When should retesting be performed?
A) Before test execution begins
B) After defects are fixed by developers
C) During test design phase
D) Only after all test cases pass

Answer: B – Retesting verifies that fixed defects no longer occur and the fix works as intended.

3

Question 3

Which is NOT an entry criterion for test execution?
A) Test environment is ready
B) Test cases are reviewed and approved
C) All defects are fixed
D) Test data is prepared

Answer: C – Defect fixes are outcomes of execution, not entry criteria. Entry criteria focus on readiness to begin testing.

Interview Keywords: Test Execution

Entry Criteria

Prerequisites that must be met before test execution can begin, ensuring the environment and artefacts are ready for testing.

Exit Criteria

Conditions that must be satisfied before test execution can be considered complete, such as pass rates and defect closure.

Retesting

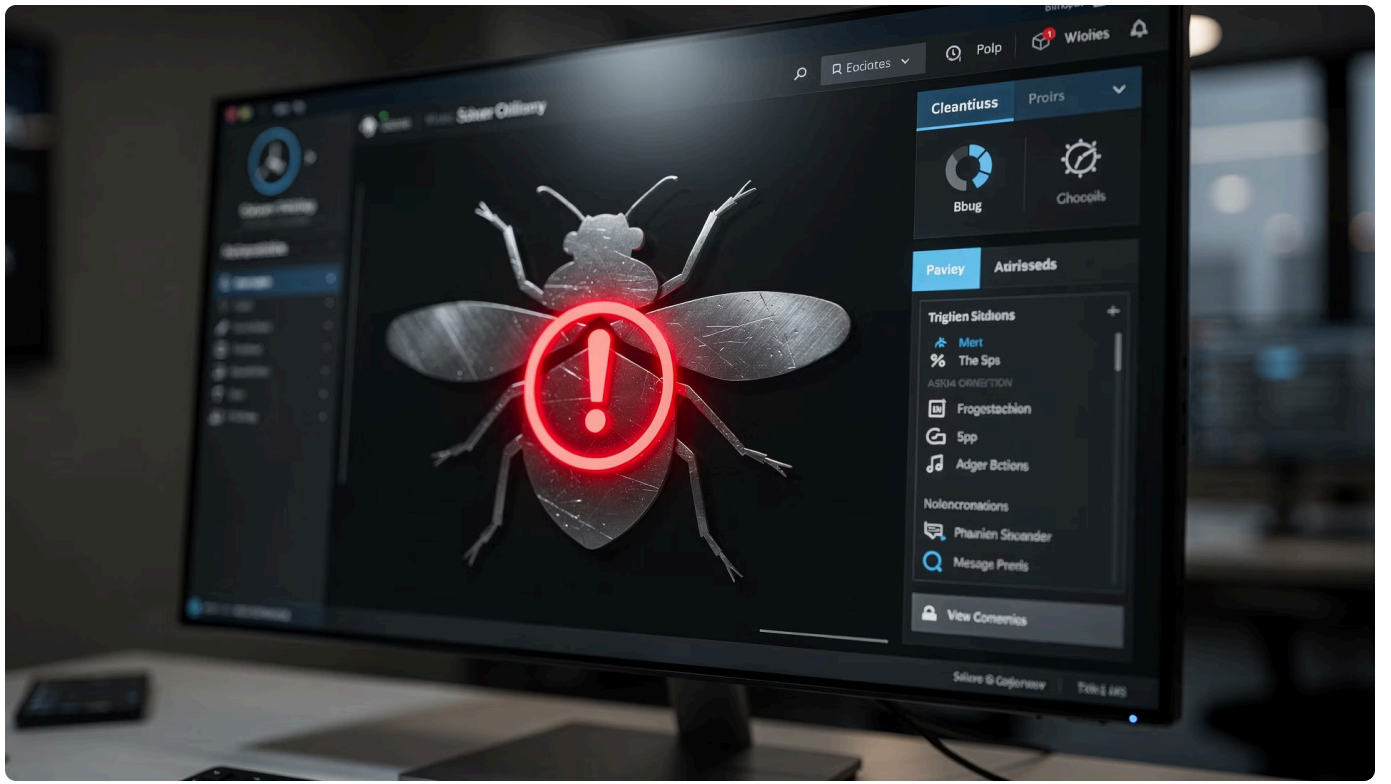
Re-executing specific test cases that previously failed, after developers fix the defects, to confirm the fixes work correctly.

Regression Testing

Re-executing a suite of test cases to ensure new changes or fixes haven't introduced defects in previously working functionality.

SECTION 2

What Is a Defect?



A defect, also called a bug or fault, is a deviation of the software from its expected behaviour or specified requirements. Defects occur when the actual result does not match the expected result during testing or production use.

Defects can range from functional errors (features not working), to performance issues (slow response times), to usability problems (confusing interfaces). They represent gaps between what was specified and what was delivered.

Not all unexpected behaviours are defects. Testers must analyse whether the issue stems from incorrect implementation, misunderstood requirements, or environmental factors before logging a defect.

Why Defect Lifecycle Management Matters

Effective defect lifecycle management ensures defects are tracked, prioritised, and resolved systematically. Without structured defect management, critical issues can be overlooked, communication breaks down, and release quality suffers.

Clear Communication

The lifecycle provides a common language between testers, developers, and managers, ensuring everyone understands defect status and ownership at any time.

Accountability

Each defect state has a designated owner responsible for the next action, preventing defects from languishing unresolved or falling through cracks.

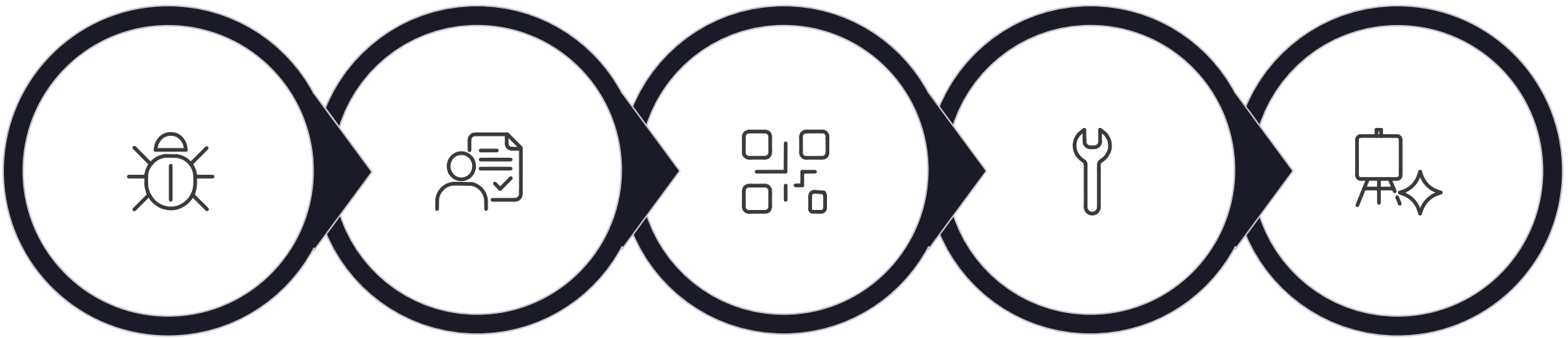
Metrics and Reporting

Defect lifecycle data enables meaningful metrics such as defect density, fix rates, and reopen rates that inform quality decisions and process improvements.

Timely Resolution

Tracking defect progression through lifecycle states ensures critical defects are addressed promptly, preventing delays in release schedules and customer impact.

Defect Lifecycle Flow



New Assigned Open Fixed Retest

Defect Lifecycle States Explained

1

New

Tester discovers and logs the defect in the tracking system with all required details: steps to reproduce, expected vs actual results, severity, and evidence.

2

Assigned

QA lead or test manager reviews the defect, validates it, and assigns it to a developer or development team for investigation and fixing.

3

Open

Developer acknowledges the defect, confirms it's valid, and begins working on identifying the root cause and implementing a fix.

4

Fixed

Developer completes the code fix, commits changes to version control, and updates the defect status indicating the fix is ready for retesting.

5

Retest

Tester re-executes the failed test case and verifies whether the fix resolves the defect without introducing new issues.

6

Verified

Tester confirms the fix works correctly and the defect is resolved. The defect moves towards closure pending final approval.

7

Closed

QA lead or test manager performs final review and closes the defect, confirming resolution is satisfactory and complete.

Alternative Defect States

Reopen

Defect still exists after developer marked it fixed. Tester reopens it with details explaining why the fix is insufficient or incomplete.

Rejected

Developer or QA lead determines the reported issue is not a defect. Reasons include working as designed, environment issue, or misunderstood requirement.

Deferred

Valid defect but not critical for current release. Team decides to postpone fixing to a future release due to time, resource, or priority constraints.

Duplicate

Defect has already been reported by another tester. The duplicate defect is linked to the original and closed to avoid redundant work.

Alternative states represent deviations from the standard lifecycle flow. Understanding when and why to use these states prevents confusion and ensures accurate defect tracking.

Teams must establish clear criteria for applying alternative states. For example, rejection should include documented reasoning, and deferred defects require documented justification and target release information.

Roles in Defect Lifecycle

Tester

Discovers, logs, and retests defects. Provides detailed reproduction steps, evidence, and verification results. Reopens defects if fixes are inadequate.

Developer

Analyses defect root causes, implements fixes, and updates defect status. Rejects invalid defects with justification and provides fix estimates.

QA Lead

Reviews and assigns defects, validates severity and priority, makes deferral decisions, and performs final closure approval. Mediates disputes between testers and developers.

Example 1: Valid Defect Flow

Scenario

Banking application allows withdrawal of more money than account balance, causing negative balance.

Lifecycle Journey

1. **New:** Tester logs defect with steps to reproduce, screenshots showing negative balance
2. **Assigned:** QA lead assigns to backend developer
3. **Open:** Developer confirms insufficient balance validation in withdrawal API
4. **Fixed:** Developer adds balance check before withdrawal
5. **Retest:** Tester re-executes test case
6. **Verified:** Withdrawal now prevented when balance insufficient
7. **Closed:** QA lead closes defect

Key Observations

This defect followed the ideal lifecycle path with no complications. Clear communication, prompt action, and thorough verification ensured quick resolution.

Severity: Critical (financial data integrity)
Priority: High (must fix before release)
Resolution Time: 2 days

Example 2: Rejected Defect Flow

Key Observations

This defect was rejected because it represented a misunderstanding of requirements, not a software fault. The tester learned to verify requirements before logging defects.

Initial Severity: Major
Outcome: Rejected
Reason: Working as designed per business rules
Time Spent: 4 hours (investigation)

Scenario

E-commerce application does not allow order cancellation after 30 minutes. Tester logs defect: "Users cannot cancel orders after placing them."

Lifecycle Journey

1. **New:** Tester logs defect claiming missing functionality
2. **Assigned:** QA lead assigns to developer for investigation
3. **Open:** Developer reviews requirement specification
4. **Rejected:** Developer marks as "Working as Designed" with requirement reference: "Business rule BR-45: Order cancellation allowed only within 30 minutes of placement"

Case Study: Defect Reopened Due to Poor Verification

The Incident

A travel booking platform released a fix for a defect where hotel search results showed incorrect pricing. The tester marked it verified after checking only one hotel. In production, users reported the same pricing issue for international hotels.

Investigation revealed the tester verified the fix only for domestic hotels. The developer's fix addressed domestic hotel pricing but missed international hotel price calculations. Incomplete verification led to a production defect and customer complaints.

Root Cause

Insufficient test coverage during retesting. Tester did not verify the fix across all relevant scenarios and data sets.

Prevention

Create a retest checklist covering all scenarios related to the defect. Test with diverse data sets including edge cases before marking verified.

Process Improvement

Implement peer review of verification results before closing critical defects. Document verification scenarios in defect comments.

Quick Knowledge Check: Defect Lifecycle

1

Question 1

What is the difference between Reopen and Retest states?
A) Reopen means defect still exists; Retest means ready to verify fix
B) Reopen means assigned to developer; Retest means assigned to tester
C) Reopen means duplicate defect; Retest means new defect
D) No difference, they are synonyms

Answer: A – Retest is when tester verifies a fix. Reopen occurs when the fix didn't work and the defect persists.

2

Question 2

Who is typically responsible for assigning defects to developers?
A) Project manager
B) Business analyst
C) QA lead or test manager
D) The tester who found it

Answer: C – QA leads review new defects, validate them, and assign them to appropriate developers based on module ownership.

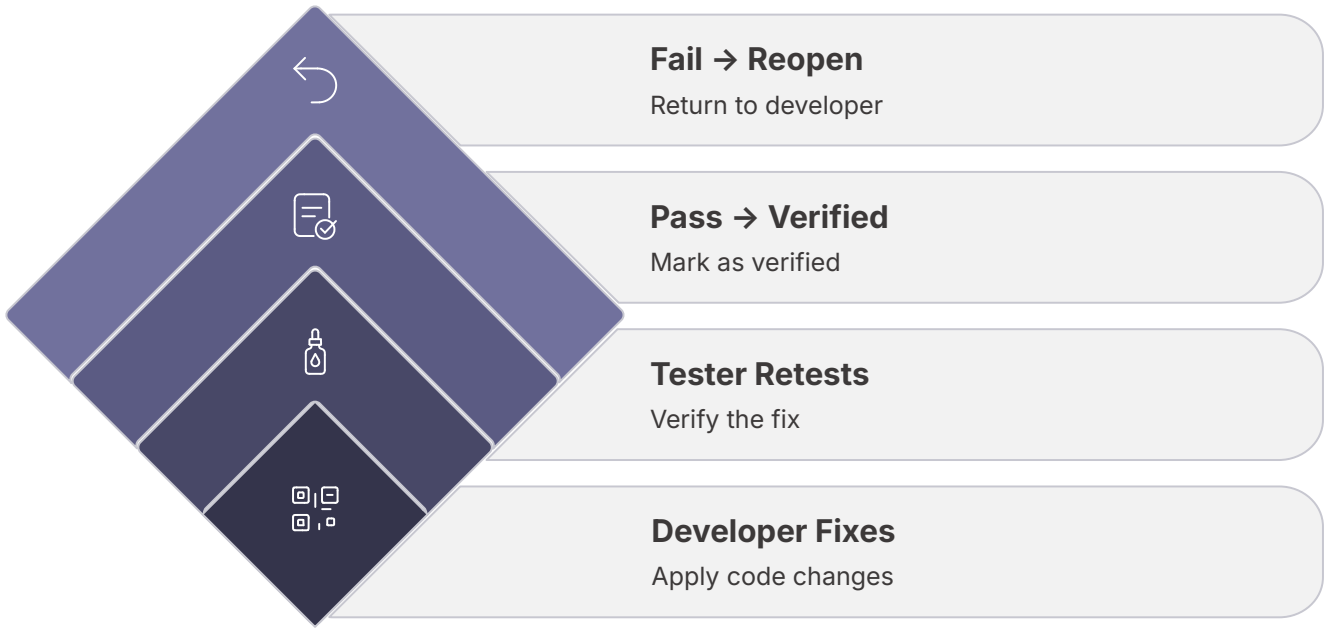
3

Question 3

When should a defect be marked as Deferred?
A) When it's a duplicate of another defect
B) When it's not valid according to requirements
C) When it's valid but not critical for current release
D) When the tester cannot reproduce it

Answer: C – Deferred defects are valid issues postponed to future releases due to time, priority, or resource constraints.

Interview Tips: Reopen vs Retest



Common Interview Question

"Explain the difference between Reopen and Retest states in defect lifecycle."

Professional Answer

Retest is a defect state indicating the developer has marked the defect as fixed, and it's now the tester's responsibility to verify the fix by re-executing the failed test case.

Reopen is a state indicating the tester verified the fix but found the defect still exists. The defect is sent back to the developer with evidence showing the fix was inadequate or incomplete.

Key distinction: Retest is the verification activity; Reopen is the outcome when verification fails. Not all defects in Retest state get reopened – only those where the fix doesn't work.

SECTION 3

What Is Severity?

Severity defines the degree of impact a defect has on the application's functionality, data integrity, or system stability. It represents the technical magnitude of the problem from a quality perspective.

Severity is determined by testers or QA leads based on how severely the defect affects the software's ability to function correctly. It answers: "How bad is this defect from a technical standpoint?"

Severity focuses on the defect's impact on the system itself, independent of business considerations or user needs. A severe defect fundamentally breaks critical functionality or corrupts data.

What Is Priority?

Priority defines the urgency with which a defect should be fixed, based on business impact, customer needs, and release schedules. It represents the order in which defects should be addressed.

Priority is determined by business analysts, product owners, or project managers based on business considerations. It answers: "How soon must we fix this defect?"

Priority considers factors beyond technical impact, including customer visibility, contractual obligations, market timing, and competitive advantage. A high-priority defect requires immediate attention regardless of technical severity.

Why Both Severity and Priority Are Needed

Severity and priority serve different purposes in defect management. Using both ensures defects receive appropriate technical and business attention, preventing misallocation of development resources.

Complete Picture

Severity shows technical impact; priority shows business urgency. Together, they provide a complete view of each defect's significance and necessary response timeframe.

Resource Allocation

Development teams use both to make informed decisions about which defects to fix first, balancing technical risks with business needs and deadlines.

Stakeholder Communication

Testers communicate technical impact through severity; business stakeholders communicate urgency through priority, ensuring all perspectives inform decisions.

Release Decisions

Release managers evaluate both severity and priority to determine if defects are release blockers, must be fixed immediately, or can be deferred to future releases.

Severity vs Priority: Comparison

Aspect	Severity	Priority
Definition	Technical impact of defect on system functionality and stability	Business urgency determining when defect must be fixed
Determined By	Tester, QA lead, or test manager	Product owner, business analyst, or project manager
Perspective	Technical quality and system impact	Business value, customer impact, and timing
Question	"How bad is this defect technically?"	"How soon must we fix this defect?"
Can Change?	Rarely changes once set correctly	Can change based on business priorities and timelines
Example	Application crashes (Critical severity)	Fix before customer demo tomorrow (High priority)

Severity Levels Explained

Critical

Application crashes, data corruption, security breaches, or complete feature failure. System is unusable or poses significant risk. Testing cannot proceed.

Major

Key functionality broken but application remains usable. Significant features fail, but workarounds exist. Major impact on user experience or business processes.

Minor

Limited functionality affected with minimal user impact. Features work but with unexpected behavior. Users can accomplish tasks with slight inconvenience.

Cosmetic

UI/UX issues with no functional impact. Spelling mistakes, alignment issues, color inconsistencies. Does not affect functionality or prevent task completion.

Priority Levels Explained

High Priority

Must be fixed immediately or before next release. Blocks critical business processes, affects key customers, or violates regulatory requirements. Development team should address before other work.

Medium Priority

Should be fixed in current release cycle but not immediately. Important but not blocking release. Can be scheduled alongside other development work with reasonable timeframes.

Low Priority

Can be fixed in future release or when resources available. Minimal business impact, affects few users, or has acceptable workarounds. Fix when convenient without disrupting other priorities.

Example 1: High Severity, Low Priority

Scenario

Banking application's "Export transactions to PDF" feature crashes the application when user selects "All transactions" option for accounts with more than 10,000 transactions.

Why High Severity?

Application crash represents a critical technical failure. Complete loss of functionality and potential data corruption or loss qualifies as Critical or Major severity.

Why Low Priority?

- Very few customers have 10,000+ transactions
- Workaround exists: export by date range
- Feature is rarely used (analytics show 2% usage)
- No contractual or regulatory requirement

Decision

Team defers fix to next release. Business accepts the risk given low user impact and available workaround.

Key Lesson

Severe technical issues don't always require immediate fixes when business impact is minimal and alternatives exist. Priority reflects business reality, not just technical severity.

Example 2: Low Severity, High Priority

Decision

Team fixes the spelling error immediately, prioritizing it over more severe technical defects. Release postponed until fix is deployed.

Key Lesson

Minor technical issues can become critical business priorities due to external factors. Priority considers context beyond code quality, including reputation, contracts, and market positioning.

Scenario

E-commerce application's homepage banner misspells the company name: "Amazom" instead of "Amazon". No functional impact – the site works perfectly.

Why Low Severity?

Simple spelling mistake with zero functional impact. Application works correctly, all features function, no data corruption, no system instability. Purely cosmetic issue.

Why High Priority?

- Homepage visible to millions of customers
- Brand reputation at stake
- Media attention and social media ridicule
- Marketing campaign launching tomorrow
- Executive visibility and embarrassment

Case Study: Wrong Priority Causes Business Impact

The Incident

A retail company's inventory management system had a defect where the "Generate weekly report" button showed a minor formatting error in the header. The tester marked it Low Severity, Low Priority. The QA lead agreed and deferred the fix.

What wasn't understood: the CEO presented these reports weekly to the board of directors. The formatting error made reports look unprofessional during a critical board meeting. The CEO escalated, demanding immediate fix and investigation into why it wasn't caught.

Root Cause

Testing team lacked visibility into business context and stakeholder usage patterns. Nobody asked who used the reports and for what purpose.

Lesson Learned

Always consult business analysts or product owners when setting priority. Technical teams cannot always assess business criticality without stakeholder input.

Process Change

New requirement: defects affecting reports, dashboards, or executive tools must be reviewed with business stakeholders before setting priority, regardless of technical severity.

Quick Knowledge Check: Severity vs Priority

1

Question 1

Which combination is most common in real projects?
A) High Severity, High Priority
B) High Severity, Low Priority
C) Low Severity, High Priority
D) Low Severity, Low Priority

Answer: A – Most critical technical defects also have high business urgency. However, exceptions exist (as shown in previous examples).

2

Question 2

Who primarily determines defect priority?
A) Tester
B) Developer
C) Business analyst or product owner
D) QA lead

Answer: C – Priority reflects business urgency, so business stakeholders determine it based on customer impact, contracts, and release timing.

3

Question 3

A defect causes complete application failure but affects only one deprecated feature scheduled for removal next month. What's the likely classification?
A) High Severity, High Priority
B) High Severity, Low Priority
C) Low Severity, High Priority
D) Low Severity, Low Priority

Answer: B – Technical severity is high (complete failure) but priority is low (affects deprecated feature being removed soon).

Common Interview Questions and Answers

1

Question: Give an example of High Severity, Low Priority

Answer: "In a mobile banking app, the 'View transaction history for the past 5 years' feature crashes for users with exactly 60+ months of history. This is high severity because it's a crash, but low priority because very few users access 5-year-old data, and recent transaction viewing works fine. Business decided to defer the fix as the workaround (viewing by year) is acceptable."

2

Question: Give an example of Low Severity, High Priority

Answer: "On an e-commerce site launching a major holiday sale, the promotional banner displays 'Save 50%' instead of 'Save 60%' due to a content management error. This is low severity – it's just text – but high priority because it misleads customers during a critical sales event and could lead to legal and brand issues. Must fix immediately before campaign launches."

3

Question: Can severity or priority change during a defect's lifecycle?

Answer: "Severity rarely changes once set correctly – the technical impact is fixed. However, priority can change based on business factors. For example, a low-priority cosmetic defect might become high priority if a major client specifically complains about it, or if it appears in media coverage. Priority reflects current business context, which can evolve."

SECTION 4

What Is a Test Report?

A test report is a formal document summarising testing activities, outcomes, and quality status during or after test execution. It communicates test progress, defect status, risks, and recommendations to stakeholders.

Test reports serve as the primary communication mechanism between testing teams and management, providing visibility into quality status and enabling informed release decisions. They transform raw testing data into actionable insights.

Effective test reports balance detail with clarity, presenting metrics, trends, and recommendations in formats stakeholders can quickly understand and act upon.

Why Test Reports Matter for Stakeholders

Test reports enable data-driven decision-making by providing objective quality insights. Different stakeholders use test reports for different purposes, making clear and comprehensive reporting essential.

Project Managers

Track testing progress against schedules, identify bottlenecks, assess resource needs, and determine if testing is on track for release deadlines.

Business Stakeholders

Understand quality levels, assess business risks, make go/no-go release decisions, and determine if product meets quality expectations and customer needs.

Development Teams

Identify defect patterns, understand quality hotspots requiring attention, prioritise fixes, and assess effectiveness of their bug fixes through retest results.

QA Management

Evaluate test effectiveness, identify process improvements, allocate testing resources, and report quality status to senior management and clients.

Types of Test Reports

Daily Status Report

Brief update on testing progress from the previous day, including test cases executed, pass/fail count, new defects logged, and blockers or impediments requiring attention.

Test Execution Report

Comprehensive report covering a test cycle or sprint, detailing test coverage, execution status, defect analysis, and quality trends over the testing period.

Defect Summary Report

Focused report analysing defect data including counts by severity/priority, defect aging, reopened defects, fix rates, and defects by module or feature area.

Final Test Summary Report

Formal document marking test completion, providing comprehensive quality assessment, exit criteria status, recommendations, and go/no-go release decision support.

Test Report Contents

Test Execution Summary

Total test cases planned, executed, passed, failed, and blocked. Include pass percentage and execution completion rate.

Defect Analysis

Defect counts by severity, priority, and status. Show open vs closed defects, defect density, and age of open defects.

Test Coverage

Requirements covered, modules tested, test types completed (functional, integration, regression, performance).

Quality Observations

Quality trends, recurring defect patterns, high-risk areas, and notable quality improvements or concerns.

Risks and Issues

Blockers, environmental issues, resource constraints, scope changes, and any factors impacting testing or quality.

Recommendations

Go/No-Go recommendation with justification, suggested actions, areas requiring additional testing, and release readiness assessment.

Comprehensive test reports include both quantitative metrics and qualitative analysis. Numbers provide objective data whilst observations and recommendations add context and guidance.

The recommendation section is crucial – stakeholders rely on the testing team's expert assessment to inform release decisions. Clear, justified recommendations demonstrate professionalism and build trust.

Sample Test Summary Report Structure

> ### Test Summary Report: Online Banking Portal – Release 3.5
>
> **Test Period:** 15th March 2024 – 28th March 2024
> **Prepared By:** QA Team
> **Date:** 29th March 2024

Metric	Value
Total Test Cases	450
Executed	445 (98.9%)
Passed	398 (89.4%)
Failed	42 (9.4%)
Blocked	5 (1.1%)
Total Defects Logged	67
Critical/Major Open	3
Defects Fixed & Verified	58

Recommendation: No-Go for production release. Three critical defects remain open affecting payment processing. Recommend additional week for fixes and regression testing.

Example: Daily Report vs Final Report

Daily Status Report

Date: 20th March 2024

- **Test cases executed:** 35
- **Passed:** 28
- **Failed:** 7
- **New defects:** 7 (2 major, 5 minor)
- **Blockers:** Environment downtime from 2 PM – 4 PM
- **Plan for tomorrow:** Complete payment module testing, retest 4 fixed defects

Brief, focused on immediate progress and next steps. Sent daily to immediate team and project manager.

Final Test Summary Report

Test Period: 15th – 28th March 2024

- **Total execution:** 445/450 test cases (98.9%)
- **Pass rate:** 89.4%
- **Total defects:** 67 (58 fixed, 6 deferred, 3 open critical)
- **Coverage:** All requirements tested, regression complete
- **Risks:** Payment processing stability concern
- **Recommendation:** No-Go until critical defects resolved

Comprehensive, formal, includes analysis and recommendations. Sent to all stakeholders and archived.