SQA

(PS: As Usual use at your own risk)

Terms and Scenario from Q		
Term	**Definition**	**Scenario**
Testing Shows Presence of Defects	Testing can reveal defects, but it can't guarantee their absence	During testing, a small feature that was updated caused multiple other components to fail. The testers realized that the interdependencies between components were not considered during their tests.
Absence of Errors Fallacy	Just because software has no defects doesn't mean it meets user needs or business requirements.	A development team focused heavily on making sure the software had no visible defects. However, upon delivery, the client was dissatisfied because certain business requirements weren't implemented.
Pesticide Paradox	Reusing the same test cases repeatedly becomes less effective over time, as they may not catch new defects.	A tester reuses the same set of test cases for every new version of the software. Over time, the number of defects detected decreases, even though the software still contains many issues.
Defect Clustering	A small number of modules often contain a high percentage of defects.	While testing, a critical error was found in a small module. Upon further investigation, testers found that most of the serious bugs were concentrated in this particular module, while other areas had very few issues.
Exhaustive Testing is Impossible	It's practically impossible to test every single input combination, so prioritization is key.	A tester attempted to test every possible combination of inputs for a login system but realized it would take too much time and was impractical. Instead, they decided to focus on high-priority test cases.
Early Testing	Starting testing early in the development cycle to identify and fix defects sooner.	A tester begins testing as soon as the first set of requirements is documented. This allows defects to be caught early, minimizing the cost of fixing them later in development.
Sanity Testing	Quick check to verify a specific bug fix without retesting everything.	A tester manually verifies a small change in code to ensure that a particular bug is fixed, without retesting the entire application. The focus is on confirming that the specific issue is resolved.

Smoke Testing	A quick set of tests to verify basic functionality after a build.	After a build is deployed, the tester runs a predefined set of tests to ensure the basic functionalities of the application are working. This ensures that the build is stable enough for further testing.
		A tester runs an initial check of the newly deployed application to verify if core functionality, such as logging in or accessing key features, works properly before proceeding with more detailed tests.
Unit Testing	Focuses on testing individual functions or modules in isolation to ensure they work correctly.	A developer creates a set of tests for individual functions within a module to ensure that each function works as expected, without considering the interactions between different components.
		Developers want to make sure each piece of their code works as expected. They create small, isolated test cases to verify that each function and method produces the correct output.
Regression Testing	Verifies that code changes haven't introduced new defects in existing functionality.	A tester runs automated scripts on a daily basis to verify that new changes to the code do not break existing functionality. These tests cover a broad range of the application's features to ensure consistency.
		Testers execute pre-written scripts to evaluate whether newly implemented changes introduced any new defects in existing functionalities. The testing is done to ensure stability after code changes.
Integration Testing	Tests how different modules or components of a system work together.	The testing team checks how the different modules of an application interact with each other. They focus on identifying errors that occur due to the communication between different parts of the system.
		After completing individual module testing, the testing team checks whether these modules work together, such as testing how a shopping cart integrates with the payment gateway.

Non-Functional Testing	Focuses on aspects like performance, stability, security, and usability.	A tester simulates thousands of users accessing an online shopping site simultaneously to check if the system can handle high traffic without crashing or slowing down. The focus is on performance and stability.
		A tester verifies that a password field masks the input, ensuring it meets the system's security requirements and that other related security measures are working.
System Testing	Evaluates the complete, integrated system to verify it meets requirements.	Before releasing a product to the customer, testers evaluate the entire application to ensure it meets the specified requirements. This involves checking both functional and non-functional aspects of the software.
		After implementing a significant code update, the testing team manually reviews the system and runs a thorough check to ensure that the entire application works according to requirements, both in terms of functionality and performance
Test Policy	High-level document outlining the organization's overall testing goals and principles.	A document outlines the company's overall goals for testing, including how testing aligns with business objectives and the key principles that guide testing activities across projects.
Test Strategy	Outlines the overall testing approach, including levels, types, tools, and risk management.	This document specifies the testing approach for a project, defining testing levels, types of tests to be conducted, tools to be used, and risk management for the testing process.
		A formal document is created to communicate the overall approach and testing methodology that will be applied to achieve the testing goals. It aligns with the test policy and provides direction on how to execute tests.
Requirements Traceability Matrix	Maps test cases to individual requirements to ensure complete test coverage.	A tester wants to make sure that every requirement has been tested. They use a document that maps test cases to individual functional requirements to ensure full coverage.

Test Plan	Detailed document outlining the scope, objectives, resources, and schedule for testing.	The testing team creates a detailed plan for a project that includes the scope, objectives, resources, schedule, and test deliverables. It serves as a guide for the overall testing process.
Test Case	Detailed step-by-step guide for verifying specific functionality, including preconditions, steps, and expected results.	A tester prepares a step-by-step guide to verify a specific functionality in the application. It includes preconditions, the specific steps to execute, and the expected result.
Test Data	Input values used during testing to ensure the software behaves as expected.	Before executing tests, the tester generates or collects input values that will be used to perform the test cases. These inputs ensure the software behaves as expected during testing.
Test Scenario	High-level description of what to test, outlining functionalities or modules needing verification	A tester prepares a high-level description of what needs to be tested without going into detailed steps. It describes specific functionalities or modules that should be verified in testing.
Test Summary Report	Summarizes testing activities and results, including passed/failed tests, defect density, and major issues.	After testing is completed, the testing team documents the overall results, including the number of test cases passed, failed, defect density, and any major issues. The document serves as a final summary of testing activities.

Pointers on Application Parts

7 Principles:

- Early Testing
- Absences of Fallacy Error
- Pesticide Paradox
- Exhaustive Testing is not possible
- Defect Clustering
- Testing is Context Dependent
- Testing Shows presence of defects

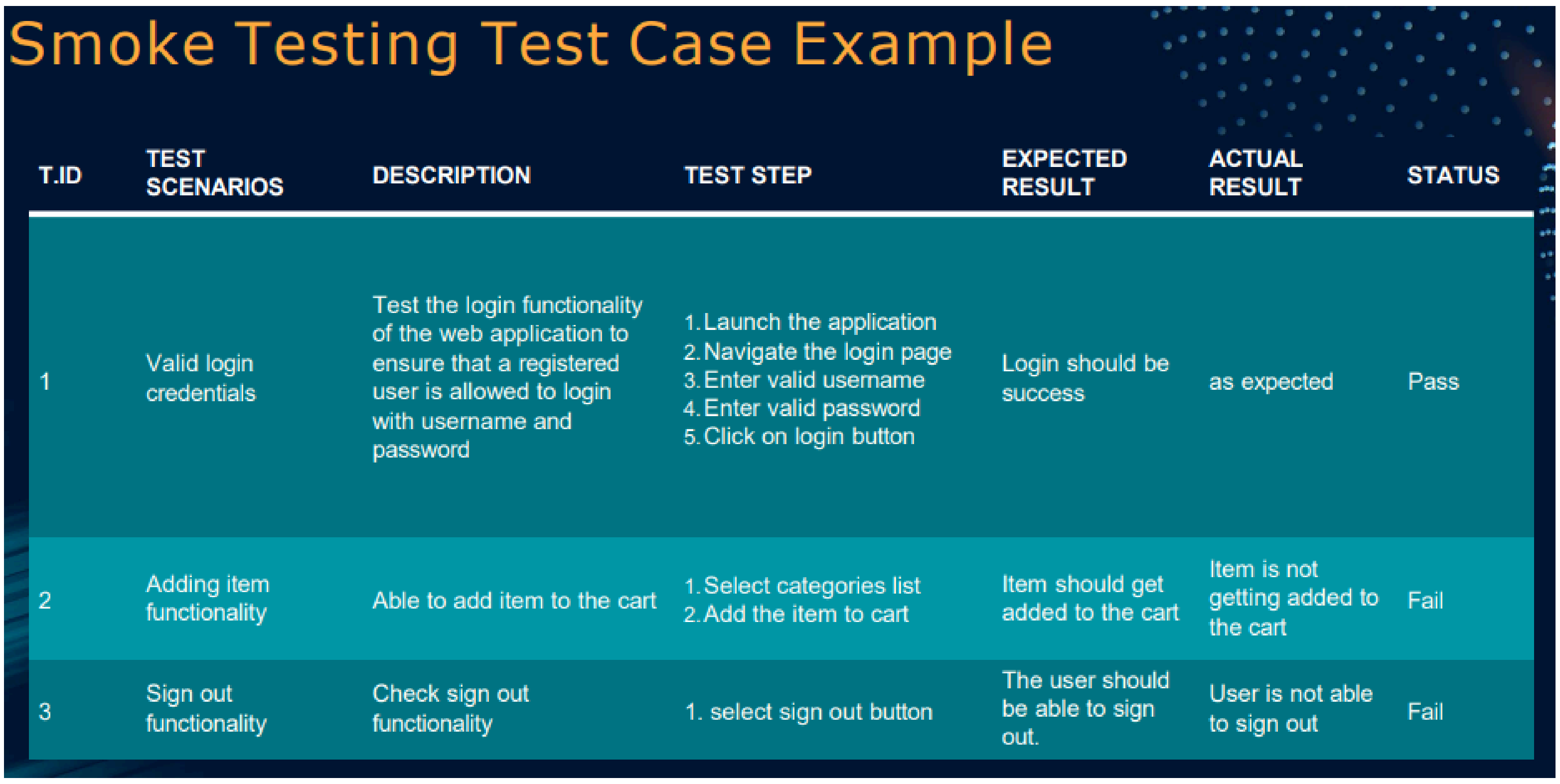
Definition and Scenario from Internet due to indirect definitions from the module			
Principle	Definition	Sample Scenario	Keywords
Early Testing	Testing should start as early as possible in the software development lifecycle to find defects sooner, when they are cheaper to fix.	A tester starts writing test cases based on the initial requirements document, even before any code is written.	• Early detection, • requirements analysis, • shift-left, • cost reduction
Absence of Errors Fallacy	Finding and fixing defects doesn't guarantee a successful product. The software must also meet user needs and business requirements.	A team delivers a bug-free application, but users find it difficult to use and it doesn't solve their problems effectively.	• User needs, • business value, • usability, • customer satisfaction
Pesticide Paradox	If the same tests are repeated over and over again, eventually they will no longer find new defects. Test cases need to be regularly reviewed and updated.	A team has been using the same test suite for years. They rarely find new bugs, but critical issues are discovered by users after release.	• Test case effectiveness, • outdated tests, • diminishing returns, • new defect discovery
Exhaustive Testing is Impossible	It is not feasible to test every possible input, combination, and scenario. Testing effort should be focused based on risk analysis and priorities.	A tester realizes they can't test every possible date input for a calendar application. They focus on boundary values, invalid dates, and common user scenarios.	• Test coverage, • prioritization, • risk analysis, • feasibility, • resource allocation
Defect Clustering	A small number of modules or areas within the software are likely to contain most of the defects. Focusing testing effort on these areas can be more efficient.	80% of the reported bugs in an application are traced back to a single complex module responsible for data synchronization.	• Defect concentration, • high-risk modules, • code quality variation, • Pareto principle
Testing is Context Dependent	Testing approaches and techniques should be tailored to the specific context of the software being developed, such as its type, intended use, and risks.	The testing strategy for a safety-critical medical device will be much more rigorous than the testing strategy for a simple mobile game.	• Project context, • risk assessment, • industry standards, • target audience

Testing Shows Presence of Defects	Testing can reveal that defects are present, but it cannot prove that there are no defects.	A testing team executes a comprehensive test plan and finds 20 defects. This demonstrates the presence of defects, but it doesn't guarantee that there are no more defects to be found.	• Defect detection, • verification, • validation, • software quality

---

SMOKE TESTING AND SANITY TESTING

SMOKE TESTING	
WHAT	A software testing technique performed post software build to verify that the critical functionalities of software are working fine.
PURPOSE	To reject a software application with defects so that QA team does not waste time testing broken software application
WHEN	• Performed immediately after a new build is deployed. • It is executed before any detailed functional or regression tests are executed
SCOPE	• Covers the most essential features, like login, main navigation, and core workflows. • a typical smoke test would be -Verify that the application launches successfully, Check that the GUI is responsive ...etc.



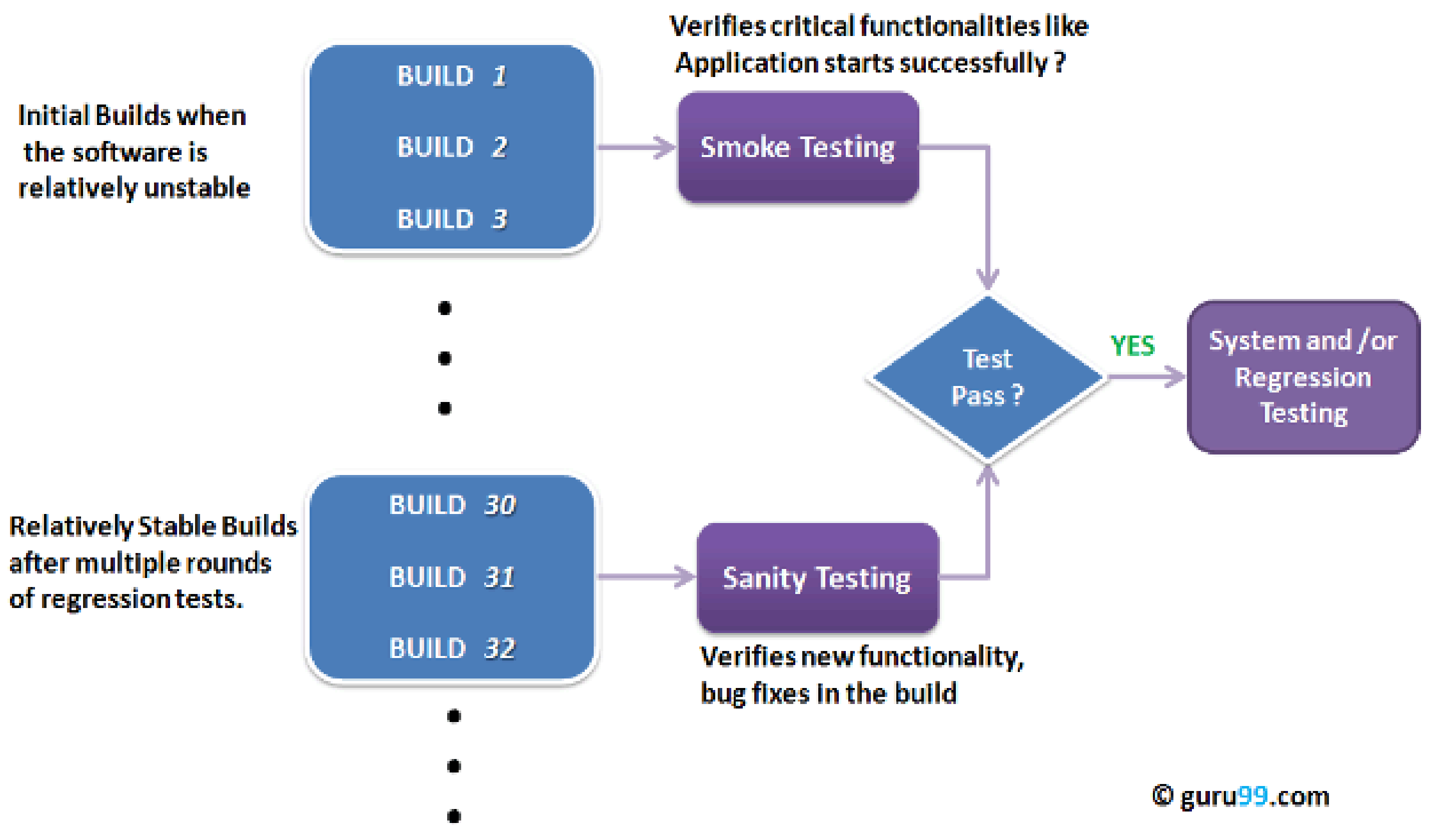
Smoke Testing Test Case Example

T.ID	TEST SCENARIOS	DESCRIPTION	TEST STEP	EXPECTED RESULT	ACTUAL RESULT	STATUS
1	Valid login credentials	Test the login functionality of the web application to ensure that a registered user is allowed to login with username and password	1. Launch the application 2. Navigate the login page 3. Enter valid username 4. Enter valid password 5. Click on login button	Login should be success	as expected	Pass
2	Adding item functionality	Able to add item to the cart	1. Select categories list 2. Add the item to cart	Item should get added to the cart	Item is not getting added to the cart	Fail
3	Sign out functionality	Check sign out functionality	1. select sign out button	The user should be able to sign out.	User is not able to sign out	Fail

SANITY TESTING	
WHAT	A kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes.
PURPOSE	• To determine that the proposed functionality works roughly as expected • Objective is "not" to verify thoroughly the new functionality but to determine that the developer has applied some rationality (sanity) while producing the software.
WHEN	• After receiving a software build • After bug fixes or minor code changes.
SCOPE	Limited to the specific functionality affected by the change.

Key Differences Between Smoke and Sanity Testing		
Feature	Smoke Testing	Sanity Testing
Purpose	Verify build stability	Verify specific changes and rationality
Scope	Broad, critical functionalities	Narrow, focused on changes
Timing	After every new build	After bug fixes or minor changes
Personnel in charge of Execution	both developers or testers	testers
Documentation	Documented	Not Documented

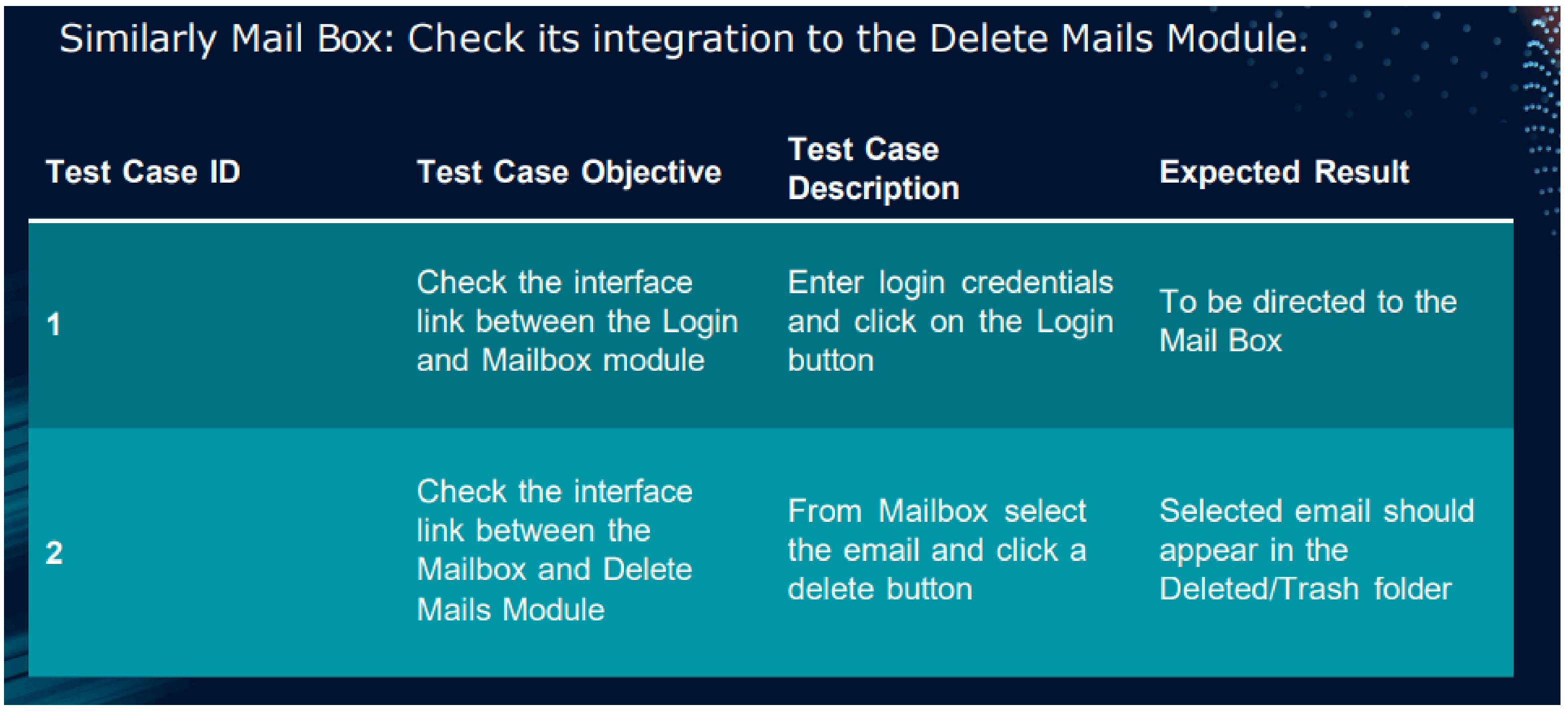
Illustration of the Difference between Smoke and Sanity Testing


Verifies critical functionalities like Application starts successfully ?
Initial Builds when the software is relatively unstable
BUILD 1
BUILD 2
BUILD 3
Smoke Testing
Test Pass ?
YES
System and /or Regression Testing
Relatively Stable Builds after multiple rounds of regression tests.
BUILD 30
BUILD 31
BUILD 32
Sanity Testing
Verifies new functionality, bug fixes in the build
© guru99.com

INTEGRATION TESTING

INTEGRATION TESTING	
WHAT	• A type of testing where software modules are integrated logically and tested as a group. • focuses on checking data communication amongst these modules
PURPOSE	To expose defects in the interaction between these software modules when they are integrated
WHEN	After unit testing and before system testing
SCOPE	• Focuses mainly on the interfaces & flow of data/information between the modules • Priority is to be given for the integrating links rather than the unit functions which are already tested.

SAMPLE INTEGRATION TESTING FROM MODULE



Similarly Mail Box: Check its integration to the Delete Mails Module.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear in the Deleted/Trash folder

Sample Test Cases:

Scenario: Imagine an e-commerce website with the following modules:

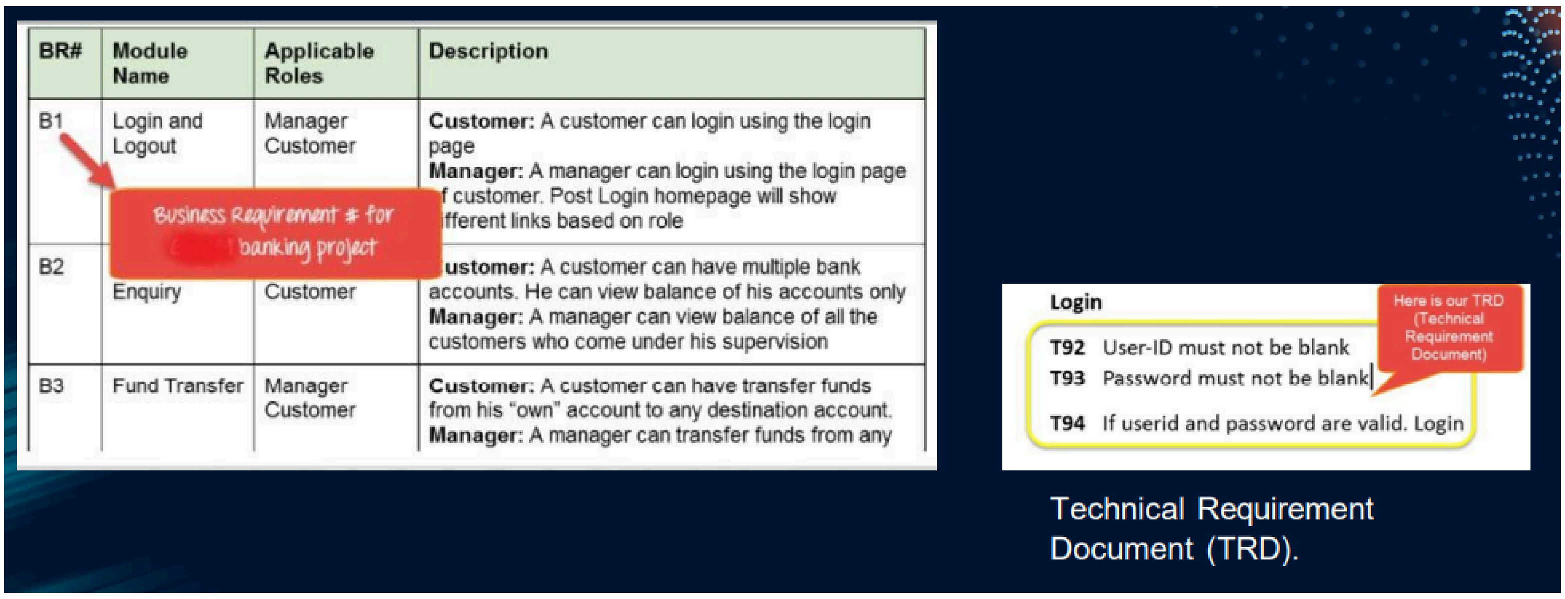
- **User Authentication:** Handles user registration, login, and logout.
- **Product Catalog:** Displays product information, search functionality, and filtering.
- **Shopping Cart:** Allows users to add/remove items, view cart contents, and proceed to checkout.
- **Payment Gateway:** Processes payments through various methods (credit card, PayPal, etc.).
- **Order Management:** Handles order placement, confirmation, and tracking.

Test Id	Test Objective	Modules Tested	Test Steps	Expected Result
IT-001	Verify successful user registration and login.	• User Authentication • Product Catalog	1. Access the website. 2. Click on "Register". 3. Fill in the registration form with valid data. 4. Submit the form. 5. Verify a confirmation email is sent. 6. Click the confirmation link in the email. 7. Log in using the newly created credentials. 8. Verify successful login and redirection to the product catalog.	Users should be successfully registered, confirmed, and logged in, landing on the product catalog page.

IT-002	Verify adding items to the shopping cart.	• Product Catalog, • Shopping Cart	1. Log in to the website. 2. Browse the product catalog. 3. Select a product and click "Add to Cart". 4. Repeat steps 2-3 for additional products.	The selected products should be added to the shopping cart, and the cart icon should reflect the correct item count.
IT-003	Verify proceeding to checkout from the shopping cart.	• Shopping Cart, • Payment Gateway	1. Log in to the website. 2. Add items to the shopping cart. 3. Click "Proceed to Checkout".	The user should be redirected to the checkout page, where they can enter payment and shipping information.
IT-004	Verify successful payment processing.	• Payment Gateway, • Order Management	1. Proceed to checkout with items in the cart. 2. Enter valid payment information. 3. Click "Place Order".	The payment should be processed successfully, and an order confirmation should be generated.
IT-005	Verify order confirmation and email notification.	• Order Management	1. Successfully place an order.	An order confirmation page should be displayed with order details, and a confirmation email should be sent to the user.

Requirement Traceability Matrix Step by Step (From Module)

Refer to the Technical Requirement Document(TRD) and Business Requirements



BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page f customer. Post Login homepage will show fferent links based on role
B2	Enquiry	Customer	ustomer: A customer can have multiple bank accounts. He can view balance of his accounts only **Manager:** A manager can view balance of all the customers who come under his supervision
B3	Fund Transfer	Manager Customer	**Customer:** A customer can have transfer funds from his "own" account to any destination account. **Manager:** A manager can transfer funds from any

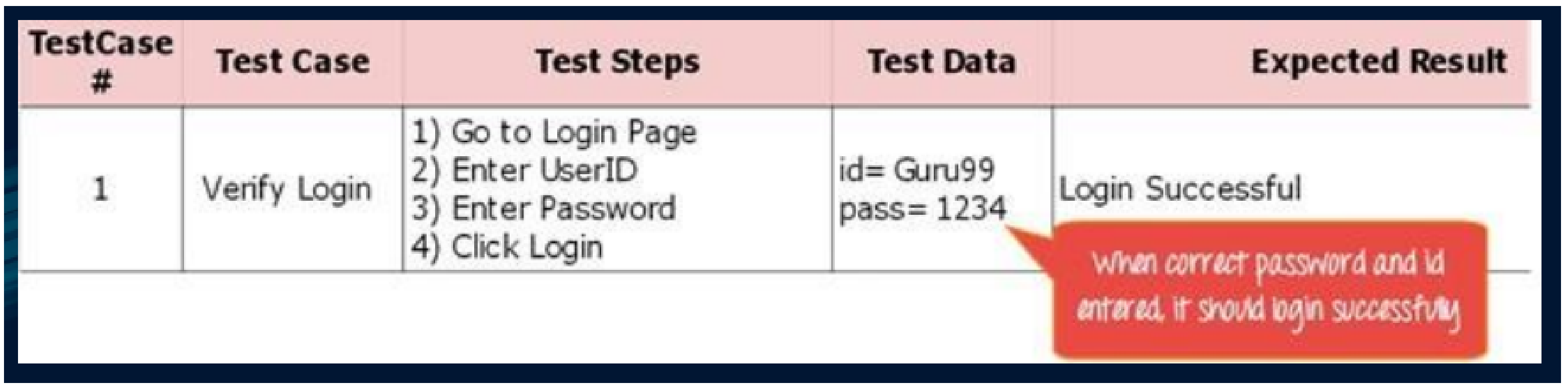
Login

T92	User-ID must not be blank
T93	Password must not be blank
T94	If userid and password are valid. Login

Technical Requirement Document (TRD).

Business Requirement # for [redacted] banking project
Here is our TRD (Technical Requirement Document)

Step 1: sample Test Case is "Verify Login, when correct ID and Password is entered, it should log in successfully"

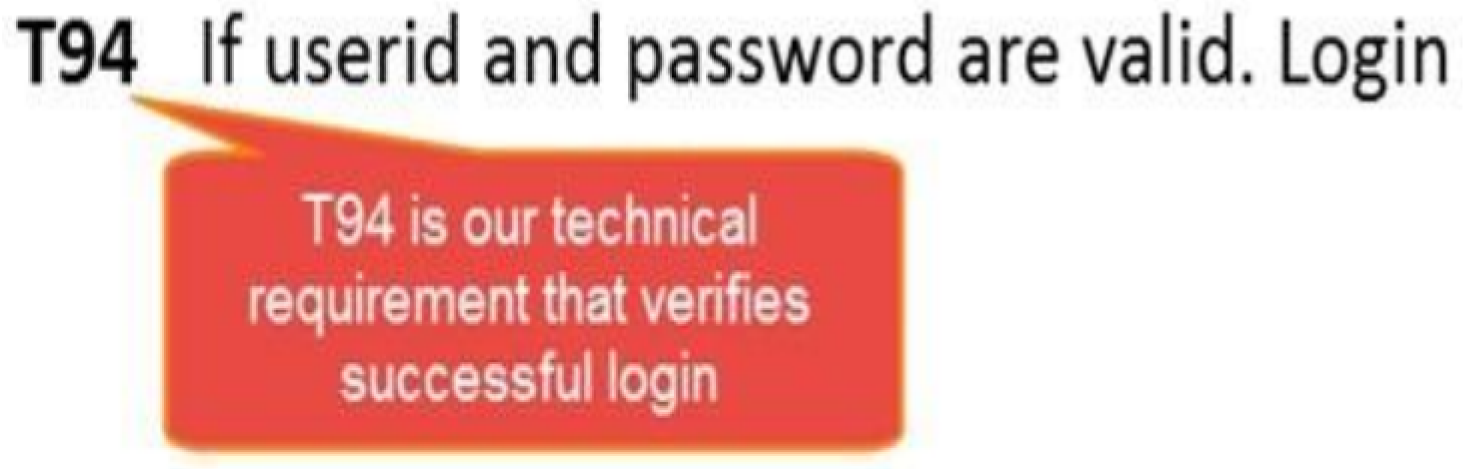


TestCase #	Test Case	Test Steps	Test Data	Expected Result

1	Verify Login	1) Go to Login Page
		2) Enter UserID
		3) Enter Password
		4) Click Login
			id= Guru99
			pass= 1234
				Login Successful

When correct password and id entered, it should login successfully

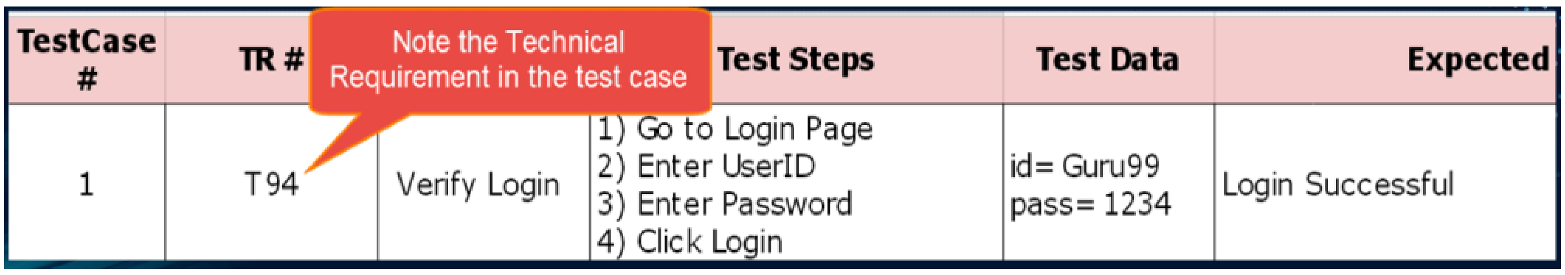
Step 2: Identify the Technical Requirement that this test case is verifying. For our test case, the technical requirement is T94 is being verified.



T94 If userid and password are valid. Login

T94 is our technical requirement that verifies successful login

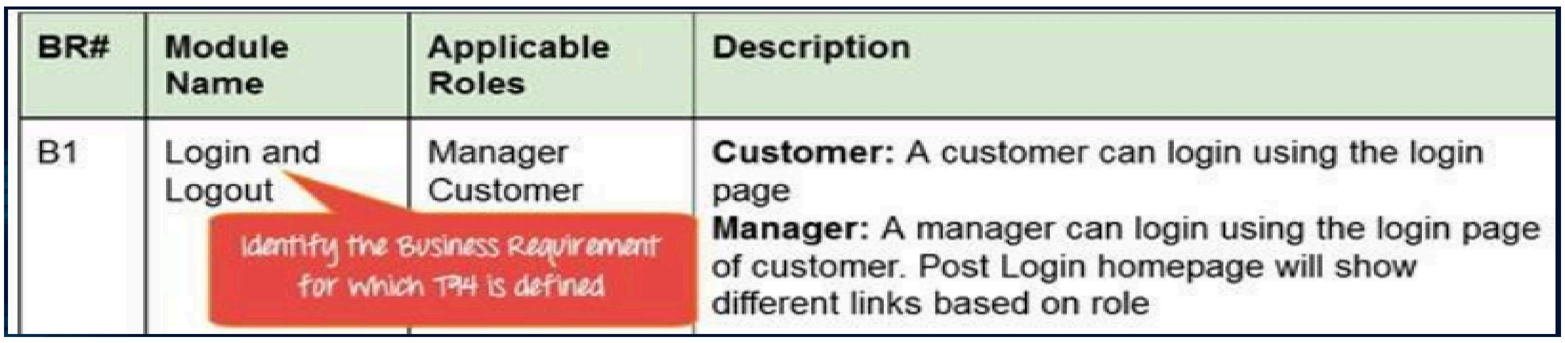
Step 3: Note this Technical Requirement (T94) in the Test Case



TestCase #	TR #		Test Steps	Test Data	Expected
1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

Note the Technical Requirement in the test case

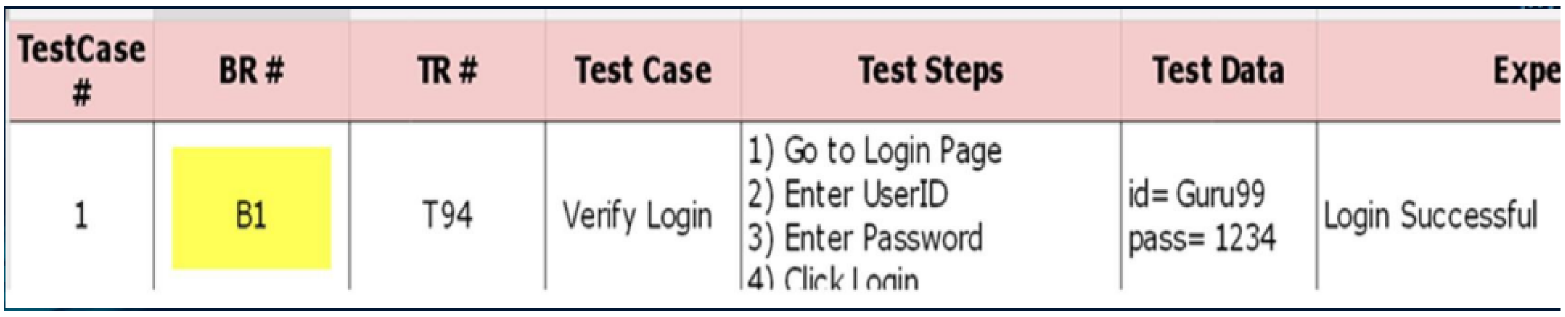
Step 4: Identify the Business Requirement for which this TR (Technical Requirement-T94) is defined



BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page of customer. Post Login homepage will show different links based on role

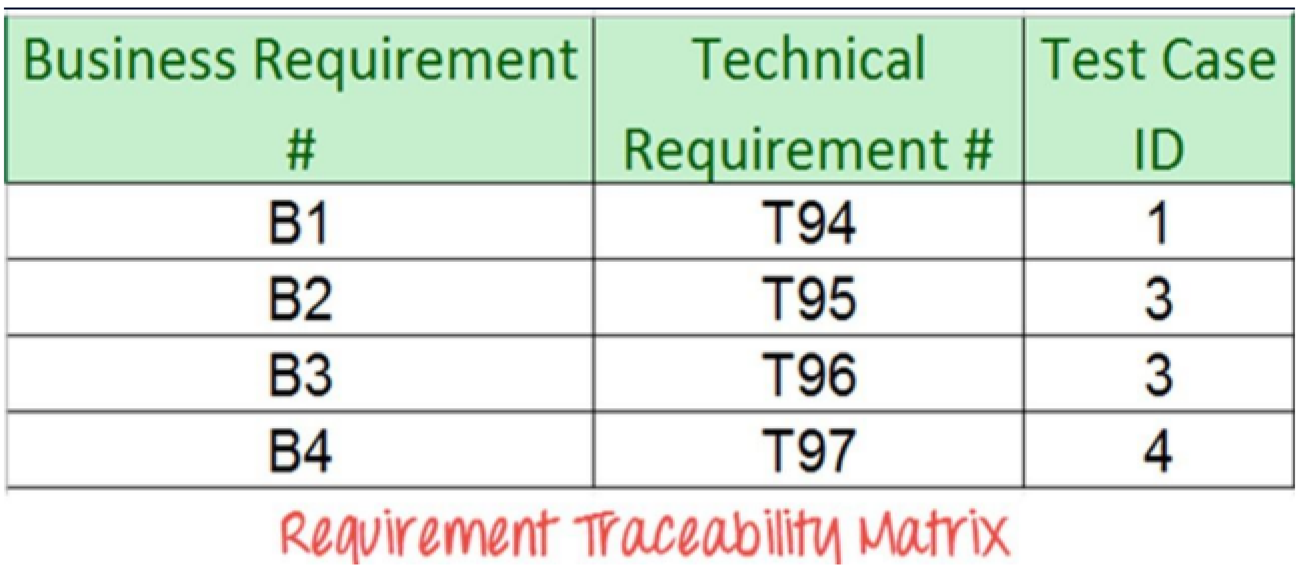
Identify the Business Requirement for which T94 is defined

Step 5: Note the BR (Business Requirement) in Test Case



TestCase #	BR #	TR #	Test Case	Test Steps	Test Data	Expe
1	B1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

Step 6: Do above for all Test Cases. Later Extract the First 3 Columns from your Test Suite. RTM in testing is Ready!



Business Requirement #	Technical Requirement #	Test Case ID
B1	T94	1
B2	T95	3
B3	T96	3
B4	T97	4

Requirement Traceability Matrix