SYSTEMS QUALITY AND ASSURANCE

SOFTWARE TESTING

- a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.
- It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.
- The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.
- Some prefer saying Software testing definition as a White Box and Black Box Testing.
- In simple terms, Software Testing means the Verification of Application Under Test (AUT).

Benefits of Software Testing

- **Cost-Effective:** It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.
- **Security:** It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.
- **Product quality:** It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
- **Customer Satisfaction:** The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

Type of Software Testing

Typically Testing is classified into three categories.

1. Functional Testing
2. Non-Functional Testing or Performance Testing
3. Maintenance (Regression and Maintenance)

TESTING CATEGORY	TYPE OF TESTING
Functional Testing	Unit Testing Integration Testing SmokeUAT (User Acceptance Testing) Localization Globalization Interoperability So on
Non-Functional Testing	Performance Endurance Load Volume Scalability Usability So on
Maintenance	Regression Maintenance

Program Testing:

- A method of executing an actual software program with the aim of testing program behavior and finding errors.
- The software program is executed with test case data to analyse the program behavior or response to the test

Importance of Software Testing

- Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product.
- Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

SOFTWARE TESTER SKILLS (NON-TECHNICAL AND TECHNICAL)

Non-Technical Skills	
Analytical Skills	Analytical skills will help break up a complex software system into smaller units to gain a better understanding and create test cases.
Communication Skill	Testing artifacts (like test cases/plans, test strategies, bug reports, etc.) created by the software tester should be easy to read and comprehend.
Time Management and Organization Skills	Testing at times could be a demanding job especially during the release of code. A software tester must efficiently manage workload, have high productivity, exhibit optimal time management, and organization skills.
GREAT Attitude	To be a good software tester you must have a GREAT attitude. An attitude to 'test to break', detail orientation, willingness to learn and suggest process improvements. In the software industry, technologies evolve with an overwhelming speed, and a good software tester should upgrade his/her technical Software testing skills with the changing technologies.
Passion	A software tester must have a passion for his/her field. BUT how do you determine whether you have a passion for software testing if you have never tested before? Simple TRY it out and if software testing does not excite you switch to something else that holds your interest.

Technical Skills	
Basic knowledge of Database/SQL	Software Systems have a large amount of data in the background. This data is stored in different types of databases like Oracle, MySQL, etc. in the backend.
Basic knowledge of Linux commands	Most of the software applications like Web-Services, Databases, Application Servers are deployed on Linux machines. So it is crucial for testers to have knowledge about Linux commands.
Knowledge and hands-on experience of a Test Management Tool	Test Management is an important aspect of Software testing. Without proper test management techniques, software testing process will fail.
Knowledge and hands-on experience of any Defect Tracking Tool	Defect Tracking and Defect life cycle are key aspects of software testing. It is extremely critical to managing defects properly and track them in a systematic manner.
Knowledge and hands-on experience of Automation tool	If you see yourself as an "Automation tester" after a couple of years working on manual testing, then you must master a tool and get in-depth, hands-on knowledge of automation tools.

7 PRINCIPLES IN SOFTWARE TESTING

1. Exhaustive Testing is Not Possible

- Exhaustive testing is not possible. Instead, we must determine the optimal amount of testing based on a risk assessment of the application.
- The key question is: How do you determine this risk?
- Consider this exercise: Which operation is most likely to cause your Operating System to fail?
- Most people would guess: Opening 10 different applications all at the same time.
- Therefore, if you were testing this Operating System, you'd realize that defects are likely to be found in multi-tasking activity and this area needs to be tested thoroughly.

2. Defect Clustering

- Defect Clustering states that a small number of modules contain most of the defects detected.
- This is the application of the Pareto Principle (the 80/20 rule) to software testing: approximately 80% of the problems are found in 20% of the modules.
- Experienced testers can identify such risky modules.
- However, if the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

3. Pesticide Paradox

- The Pesticide Paradox is named after the concept that the repetitive use of the same pesticide mix will cause insects to develop resistance over time, making the pesticide ineffective.
- The same principle applies to software testing: If the same set of repetitive tests are conducted, the method will eventually become useless for discovering new defects.
- To overcome this, test cases need to be regularly reviewed and revised, adding new and different test cases to help find more defects.
- Testers cannot simply depend on existing test techniques; they must continually look to improve existing methods to make testing more effective.
- Even after extensive testing, a product can never be definitively claimed as bug-free.

4. Testing Shows the Presence of Defects

- The core principle here is: Testing talks about the presence of defects and doesn't talk about the absence of defects.
- Software Testing reduces the probability of undiscovered defects remaining in the software.
- However, even if no defects are found, this is not a proof of correctness or a guarantee that the software is bug-free.
- The bigger challenge is: What if you make your software product 99% bug-free, but it does not meet the needs and requirements of the clients?

5. Absence of Error is a Fallacy

- The Absence of Error is a Fallacy (a false belief) means that software that is 99% bug-free can still be unusable.
- This can happen if the system is tested thoroughly for the wrong requirements.
- Software testing is not merely about finding defects; it is also about checking that the software addresses the business needs.
- Finding and fixing defects does not help if the system built is unusable and does not fulfill the user's needs and requirements.

6. Early Testing

- Early Testing means that testing should start as early as possible in the Software Development Life Cycle (SDLC).
- Starting early ensures that any defects in the requirements or design phase are captured in the initial stages.
- It is much cheaper to fix a defect in the early stages of testing than later on.
- It is recommended that you start finding bugs the moment the requirements are defined.

7. Testing is Context Dependent

- Testing is context dependent means that the way you test an application is specific to its type and purpose.
- All developed software applications are not identical.
- You must use different approaches, methodologies, techniques, and types of testing depending upon the application type.
- For instance, testing a POS (Point of Sale) system at a retail store requires a different approach than testing an ATM machine.

SDLC (Software Development Life Cycle): It is the sequence of activities carried out by Developers to design and develop high-quality software.

STLC (Software Testing Life Cycle): It consists of a series of activities carried out by Testers methodologically to test your software product.

V-Model

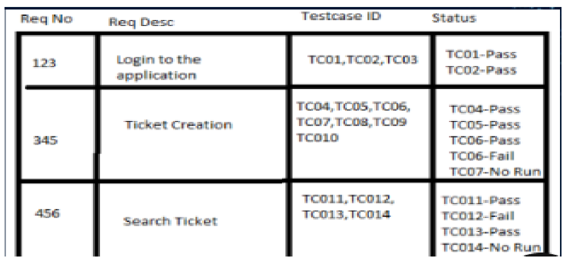
- V Model is a highly disciplined SDLC model in which there is a testing phase parallel to each development phase.
- The V model is an extension of the waterfall model in which testing is done on each stage parallel with development in a sequential way.
- It is known as the Validation or Verification Model.

Waterfall Model: A sequential model divided into different phases of software development activity. Each stage is designed for performing the specific activity. Testing phase in waterfall model starts only after implementation of the system is done.

Entry Criteria: gives the prerequisite items that must be completed before testing can begin.

Exit Criteria: defines the items that must be completed before testing can be concluded

Requirements Traceability Matrix (RTM)



Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

A **Requirements Traceability Matrix (RTM)** is a document used in software development and testing to link and track requirements through every phase of the project lifecycle, ensuring that all requirements are covered by corresponding test cases.

STLC Phases	
Requirement Phase Testing/ Requirement Analysis	Test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail. • Requirements could be either functional or non-functional. • Automation feasibility for the testing project is also done in this stage. **Activities in Requirement Phase Testing** • Identify types of tests to be performed. • Gather details about testing priorities and focus. • Prepare the Requirement Traceability Matrix (RTM). • Identify test environment details where testing is supposed to be carried out. • Automation feasibility analysis (if required). The **Requirement Traceability Matrix (RTM)** is a key deliverable created during the requirement phase to link and track requirements through the testing process.
Test Planning	This is where a Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations, and the testing schedule are also determined. The Test Plan is prepared and finalized in this phase. **Test Planning Activities** • Preparation of test plan/strategy document for various types of testing • Test tool selection • Test effort estimation • Resource planning and determining roles and responsibilities. • Training requirement **Deliverables of Test Planning** • Test plan /strategy document. • Effort estimation document.
Test Case Development Phase	Involves the creation, verification, and rework of test cases & test scripts after the test plan is ready. Initially, the test data is identified, then created and reviewed, and then reworked based on the preconditions. The QA team then starts the development process of test cases for individual units. **Test Case Development Activities** • Create test cases, automation scripts (if applicable) • Review and baseline test cases and scripts • Create test data (If Test Environment is available) **Deliverables of Test Case Development** • Test cases/scripts • Test data
Test Environment Setup	Decides the software and hardware conditions under which a work product is tested. It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase. The test team may not be involved in this activity if the development team provides the test environment; however, the test team is required to do a readiness check (smoke testing) of the given environment. **Test Environment Setup Activities** • Understand the required architecture, environment set-up, and prepare hardware and software requirement list for the Test Environment. • Setup test Environment and test data • Perform smoke test on the build **Deliverables of Test Environment Setup** • Environment ready with test data set up • Smoke Test Results.
Test Execution Phase	This is carried out by the testers, where the testing of the software build is done based on the test plans and test cases prepared. The process consists of test script execution, test script maintenance, and bug reporting. If bugs are reported, they are reverted back to the development team for correction, and retesting will be performed. **Test Execution Activities** • Execute tests as per plan • Document test results, and log defects for failed cases • Map defects to test cases in RTM • Retest the Defect fixes • Track the defects to closure **Deliverables of Test Execution** • Completed RTM with the execution status • Test cases updated with results • Defect reports
Test Cycle Closure	The completion of test execution which involves several activities like test completion reporting, collection of test completion metrics, and test results. Testing team members meet, discuss, and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test

	cycle. The idea is to remove process bottlenecks for future test cycles. **Test Cycle Closure Activities** • Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality. • Prepare test metrics based on the above parameters. • Document the learning out of the project **Prepare Test closure report** • Qualitative and quantitative reporting of quality of the work product to the customer. • Test result analysis to find out the defect distribution by type and severity.

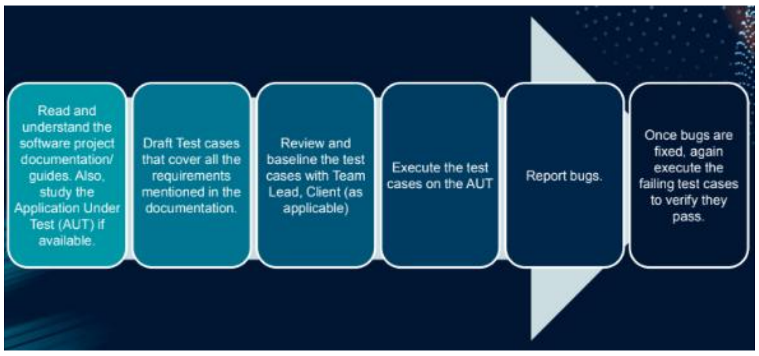
Principles of STLC

- **Validation**: Am I building the product right?
- **Verficaition:** Am I building the right product?

MANUAL TESTING

- a type of software testing in which test cases are executed manually by a tester without using any automated tools.
- The purpose of Manual Testing is to identify the bugs, issues, and defects in the software application.
- Manual software testing is the most primitive technique of all testing types and it helps to find critical bugs in the software application.
- Any new application must be manually tested before its testing can be automated
- Manual Software Testing requires more effort but is necessary to check automation feasibility.
- Manual Testing concepts does not require knowledge of any testing tool.
- One of the Software Testing Fundamental is “100% Automation is not possible“. This makes Manual Testing imperative.

How manual testing is performed


Read and understand the software project documentation/ guides. Also, study the Application Under Test (AUT) if available.
Draft Test cases that cover all the requirements mentioned in the documentation.
Review and baseline the test cases with Team Lead, Client (as applicable)
Execute the test cases on the AUT
Report bugs.
Once bugs are fixed, again execute the failing test cases to verify they pass.

Types of Manual testing

Diagram depicts Manual Testing Types. In fact, any type of software testing type can be executed both manually as well using an automation tool.

- Black Box Testing
- White Box Testing
- Unit Testing
- System Testing
- Integration Testing
- Acceptance Testing

Black Box Testing: System Testing is the opposite. System test involves the external workings of the software from the user’s perspective.

White Box Testing: the testing of the internal workings or code of a software application.

Unit Testing

- a type of software testing where individual units or components of a software are tested.
- The purpose is to validate that each unit of the software code performs as expected.
- Unit Testing is done during the development (coding phase) of an application by the developers.
- Unit Tests isolate a section of code and verify its correctness.
- A unit may be an individual function, method, procedure, module, or object.
- In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing.
- Unit testing is a WhiteBox testing technique that is usually performed by the developer.
- Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

System Testing

- a level of testing that validates the complete and fully integrated software product.
- The purpose of a system test is to evaluate the end-to-end system specifications.
- Usually, the software is only one element of a larger computer-based system.
- Ultimately, the software is interfaced with other software/hardware systems.
- System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system. Integration Testing
- System test falls under the **black box** testing category of software testing.

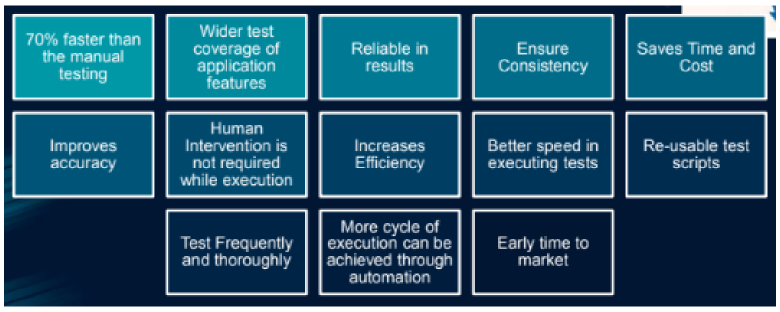
Acceptance Testing: beta testing of the product done by the actual end users.

AUTOMATION TESTING

- a software testing technique that performs using special automated testing software tools to execute a test case suite.
- On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.
- The automation testing software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports. Software Test Automation demands considerable investments of money and resources.
- Successive development cycles will require execution of same test suite repeatedly.
- Using a test automation tool, it’s possible to record this test suite and re-play it as required.
- Once the test suite is automated, no human intervention is required. This improved ROI of Test Automation.
- The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual Testing altogether.

Automation Testing advantages / disadvantages

Benefits:


70% faster than the manual testing
Wider test coverage of application features
Reliable in results
Ensure Consistency
Saves Time and Cost
Improves accuracy
Human Intervention is not required while execution
Increases Efficiency
Better speed in executing tests
Re-usable test scripts
Test Frequently and thoroughly
More cycle of execution can be achieved through automation
Early time to market

Disadvantages

- ***Automation Proces Cost and Resource Investment:*** Software Test Automation demands considerable investments of money and resources.
- ***Scope Limitations:***
  - Automated testing is recommended only for stable systems.
  - It is not possible to achieve 100% automation, making manual testing an essential, imperative component.
  - Certain types of testing, like ad-hoc and monkey testing, are more suited for manual execution.
- ***Unsuitable Test Cases:*** The following categories of test cases are not suitable for automation:
  - Test Cases that are newly designed and have not been executed manually at least once.
  - Test Cases for which the requirements are frequently changing.
  - Test cases that are executed on an ad-hoc basis.

Types of Framework Under Automation Testing

- ***Data Driven Automation Framework*** - input values are read from data files and stored into variables in test scripts.
- ***Keyword Driven Automation Framework*** - uses data files to contain the keywords related to the application being tested.
- ***Modular Automation Framework*** - the approach where all the test cases are first analyzed to find out the reusable flows. Then while scripting, all these reusable flows are created as functions and stored in external files and called in the test scripts wherever required.
- ***Hybrid Automation Framework*** – combination of data driven and keyword driven.

UNIT TESTING

- a type of software testing where individual units or components of a software are tested.
- The purpose is to validate that each unit of the software code performs as expected.
- Unit Testing is done during the development (coding phase) of an application by the developers.
- Unit Tests isolate a section of code and verify its correctness.

- A unit may be an individual function, method, procedure, module, or object.
- In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing.
- Unit testing is a WhiteBox testing technique that is usually performed by the developer.
- Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

White Box Testing: involves testing the functional behaviour of the software application

Black Box Testing: involves testing of user interface along with input and output,

Gray Box Testing: testing that is used to execute test suites, test methods, test cases and performing risk analysis.

Unit testing Automation Approach

(1)

- A developer writes a section of code in the application just to test the function.
- They would later comment out and finally remove the test code when the application is deployed.

(2)

- A developer could also isolate the function to test it more rigorously.
- This is a more thorough unit testing practice that involves copy and paste of code to its own testing environment than its natural environment.
- Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces in the product.
- These dependencies can then be eliminated.

(3)

- A coder generally uses a UnitTest Framework to develop automated test cases.
- Using an automation framework, the developer codes criteria into the test to verify the correctness of the code. During execution of the test cases, the framework logs failing test cases.
- Many frameworks will also automatically flag and report, in summary, these failed test cases.
- Depending on the severity of a failure, the framework may halt subsequent testing.

(4)

- The workflow of Unit Testing is
    1) Create Test Cases
    2) Review/Rework
    3) Baseline
    4) Execute Test Cases.

Code Coverage

- Code coverage is a measure which describes the degree of which the source code of the program has been tested.
- It is one form of white box testing which finds the areas of the program not exercised by a set of test cases.
- It also creates some test cases to increase coverage and determining a quantitative measure of code coverage.

Code coverage techniques

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

INTEGRATION TESTING

- a type of testing where software modules are integrated logically and tested as a group.
- A typical software project consists of multiple software modules, coded by different programmers.
- The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated
- Integration Testing focuses on checking data communication amongst these modules.
- Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing'.

Integration Testing approach

- ***Big Bang Approach:***
    - an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit.
    - This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.
- ***Incremental Approach:***
    - Is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully.
    - Incremental Approach, in turn, is carried out by two different Methods:
    - Incremental Approach Types
        - Top Down Approach
        - Bottom Up Approach
        - Sandwich Approach - Combination of Top Down and Bottom Up

Bottom-Up Integration Testing

- a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.
- ***Advantages:***
    - Fault localization is easier.
    - No time is wasted waiting for all modules to be developed unlike Big-bang approach
- ***Disadvantages:***
    - Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
    - An early prototype is not possible

Top-Down Integration Testing

- A method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.
- ***Advantages:***
    - Fault Localization is easier.
    - Possibility to obtain an early prototype.
    - Critical Modules are tested on priority; major design flaws could be found and fixed first.
- ***Disadvantages:***
    - Needs many Stubs.
    - Modules at a lower level are tested inadequately.

Sandwich Testing

- a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called Hybrid Integration Testing. It makes use of both stubs as well as drivers.

Stubs and Drivers

- a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called Hybrid Integration Testing. It makes use of both stubs as well as drivers.
- ***Stub:*** Is called by the Module under Test.
- ***Drivers:*** Calls the Module to be tested.

SYSTEM TESTING

- a level of testing that validates the complete and fully integrated software product.
- The purpose of a system test is to evaluate the end-to-end system specifications.
- Usually, the software is only one element of a larger computer-based system.
- Ultimately, the software is interfaced with other software/hardware systems.
- System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

Types of System Testing

1. ***Usability Testing***– mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
2. ***Load Testing–*** is necessary to know that a software solution will perform under real-life loads.
3. ***Regression Testing***– involves testing done to make sure none of the changes made over the course of the development process have caused

new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.

4. ***Recovery testing*** – is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
5. ***Migration testing-*** is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. ***Functional Testing*** – Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. ***Hardware/Software Testing*** – IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

SMOKE TESTING

- a software testing technique performed post software build to verify that the critical functionalities of software are working fine. It is executed before any detailed functional or regression tests are executed. The main purpose of smoke testing is to reject a software application with defects so that QA team does not waste time testing broken software application.
- In Smoke Testing, the test cases chose to cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system are working fine.
- For Example, a typical smoke test would be – Verify that the application launches successfully, Check that the GUI is responsive ... etc.

Sanity Testing

- 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. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.
- The objective is "not" to verify thoroughly the new functionality but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50.

Regression Testing

- a type of software testing to confirm that a recent program or code change has not adversely affected existing features.
- Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Re-Testing

- means testing the functionality or bug again to ensure the code is fixed. If it is not fixed, Defect needs to be re-opened. If fixed, Defect is closed.

Non-Functional Testing

- a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application.
- It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.
- An excellent example of non-functional test would be to check how many people can simultaneously login into a software.
- Non-functional testing is equally important as functional testing and affects client satisfaction.

Please study how to create TEST CASES. Remember the format headers of the TEST CASE

(TEST CASE ID, TEST DESCRIPTION, Test Steps, Test Data, Expected Result, Actual Result)

Column Header	What It Should Contain	Rules and Guidelines
Test Scenario ID	A unique ID representing the high-level feature or module being tested.	Use a prefix for tracking (e.g., TS_NF_001 for Test Scenario, NetFlix).
Test Scenario Description	A brief summary of the feature or test concept.	Focus on the what (e.g., "Check login with valid email address and correct password").
Test Case ID	A unique ID for the specific test instance (atomic test) within the scenario.	Must be unique (e.g., TC_N_Login_001). This ID is often linked to a defect if the test fails.
Test Case Description	A concise statement detailing the specific condition being tested.	Focus on the how and the specific data variation (e.g., "Login with a valid email address and valid password").
Test Steps	The sequential, step-by-step actions required to execute the test.	Must be clear, precise, and repeatable. Use numbered lists (1., 2., 3.).
Preconditions	Any necessary setup or starting state for the test environment or data.	Example: "Valid Netflix login URL and registered user credentials."
Test Data	The specific input values required to execute the test steps.	List the exact data to be used (e.g., Email: test@mail.com, password: P@ssw0rd123).
Post Conditions	The expected state of the system or data after the test has finished.	Example: "User should be redirected to the home page."
Expected Result	The precise, verifiable outcome required for the test to Pass.	Must be unambiguous and measurable (e.g., "Successful login," "Error message should appear: 'Password required'").
Actual Result	The actual behavior observed by the tester during execution.	Filled in during testing. Must record what happened, even if it differs from the Expected Result.
Status	The final outcome of the test run.	Select one: Pass, Fail, Blocked, N/A (Not Applicable).
Executed by	The name or ID of the tester who performed the execution.	Tester_T1D00
Executed Date	The date the test was last run.	2025-10-10
Comments (if any)	Any additional notes, details on failures, or environment changes.	Example: "Confirmed defect filed under BUG-105."