SOFTWARE TESTING AND QUALITY ASSURANCE

TYBSC CS

WHAT IS SOFTWARE TESTING?

- Software Testing is a method to assess the functionality of the software program.
- The process checks whether the actual software matches the expected requirements and ensures the software is bug-free.
- The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements.
- It mainly aims at measuring the specification, functionality, and performance of a software program or application.

TWO STEPS OF SOFTWARE TESTING

- Verification: It refers to the set of tasks that ensure that the software correctly implements a specific function. It means “Are we building the product right?”.
- Validation: It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. It means “Are we building the right product?”.

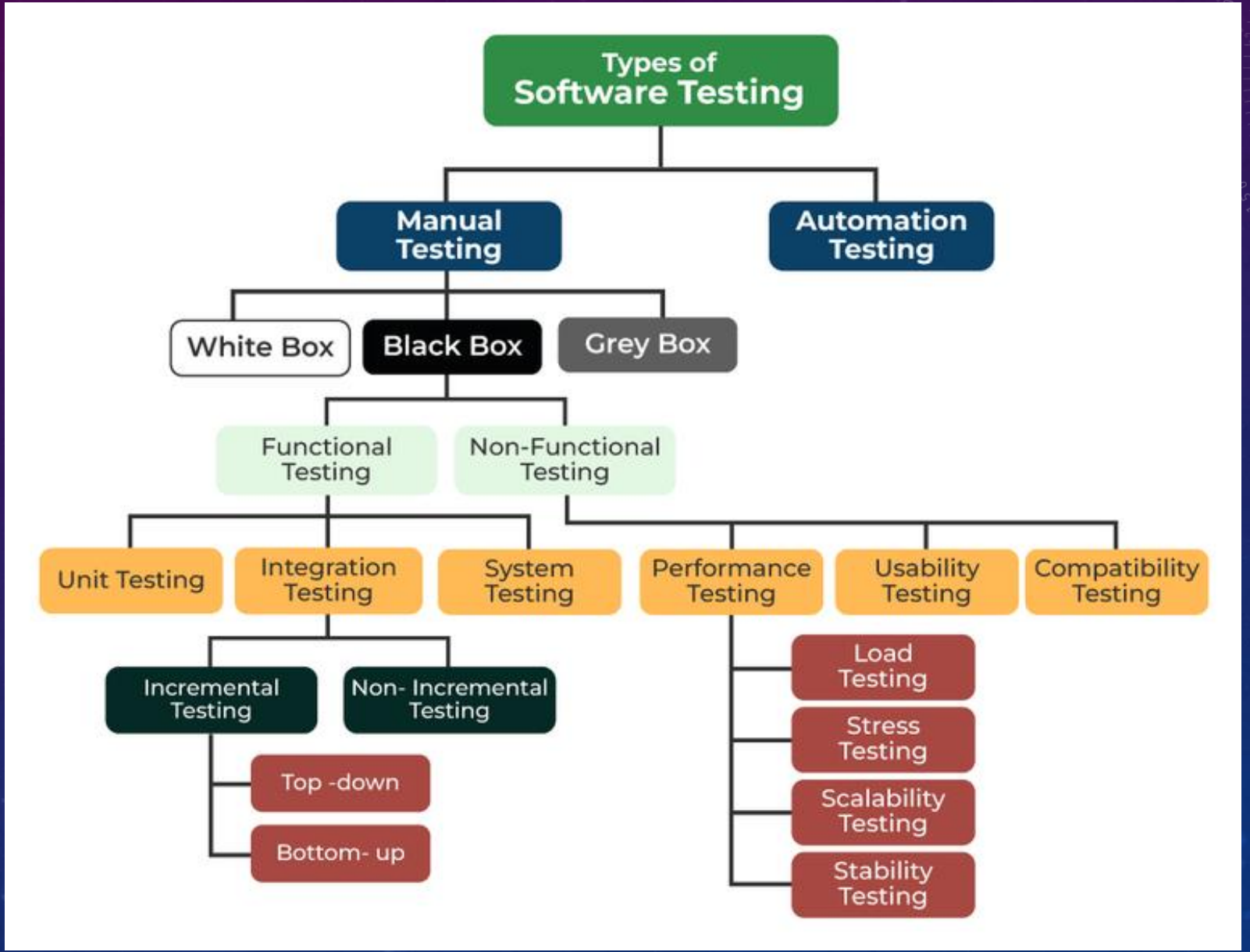
TWO STEPS OF SOFTWARE TESTING

Sr. No.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behavior.
3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.	It has dynamic activities, as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify a software.	It is a subjective process and involves subjective decisions on how well a software works.

IMPORTANCE OF SOFTWARE TESTING

- **Defects can be identified early**: Software testing is important because if there are any bugs they can be identified early and can be fixed before the delivery of the software.
- **Improves quality of software**: Software Testing uncovers the defects in the software, and fixing them improves the quality of the software.
- **Increased customer satisfaction**: Software testing ensures reliability, security, and high performance which results in saving time, costs, and customer satisfaction.
- **Helps with scalability**: Software testing type non-functional testing helps to identify the scalability issues and the point where an application might stop working.
- **Saves time and money**: After the application is launched it will be very difficult to trace and resolve the issues, as performing this activity will incur more costs and time. Thus, it is better to conduct software testing at regular intervals during software development.

DIFFERENT TYPES OF SOFTWARE TESTING



CLASSIFICATION

- **Manual testing**: It includes testing software manually, i.e., without using any automation tool or script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing. Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

- **Automation testing**: It is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.

TESTING TECHNIQUE

- **Black box Testing**: Testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without any concern with the internal logical structure of the software known as black-box testing.
- **White box Testing**: Testing in which the tester is aware of the internal workings of the product, has access to its source code, and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.
- **Grey Box Testing**: Testing in which the testers should have knowledge of implementation, however, they need not be experts.

TESTING TECHNIQUE - DIFFERENCE

S No.	Black Box Testing	White Box Testing
1	Internal workings of an application are not required.	Knowledge of the internal workings is a must.
2	Also known as closed box/data-driven testing.	Also known as clear box/structural testing.
3	End users, testers, and developers.	Normally done by testers and developers.
4	This can only be done by a trial and error method.	Data domains and internal boundaries can be better tested.

TESTING TECHNIQUE

- **Functional testing**: It is a type of software testing that validates the software systems against the functional requirements. It is performed to check whether the application is working as per the software's functional requirements or not. Various types of functional testing are Unit testing, Integration testing, System testing, Smoke testing, and so on.
- **Non-functional testing**: It is a type of software testing that checks the application for non-functional requirements like performance, scalability, portability, stress, etc. Various types of non-functional testing are Performance testing, Stress testing, Usability Testing, and so on.

TESTING TECHNIQUE

- **Unit testing**: It a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
- **Integration testing**: It is a level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
- **System testing**: It is a level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
- **Acceptance testing**: It is a level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

BENEFITS OF SOFTWARE TESTING

- **Product quality:** Testing ensures the delivery of a high-quality product as the errors are discovered and fixed early in the development cycle.
- **Customer satisfaction:** Software testing aims to detect the errors or vulnerabilities in the software early in the development phase so that the detected bugs can be fixed before the delivery of the product. Usability testing is a type of software testing that checks the application for how easily usable it is for the users to use the application.
- **Cost-effective:** Testing any project on time helps to save money and time for the long term. If the bugs are caught in the early phases of software testing, it costs less to fix those errors.
- **Security:** Security testing is a type of software testing that is focused on testing the application for security vulnerabilities from internal or external sources.

NATURE OF ERRORS

- **Errors are classified into two types:**
  - **By Origin**
  - **By Nature**

NATURE OF ERRORS

- **Classification by Origin:**
  - **Requirements Errors:** These occur due to misunderstandings or misinterpretations of the software requirements, leading to features not working as intended or being entirely missing.
  - **Design Errors:** These are flaws in the software's architecture or design, causing issues with the software's overall structure or logic.
  - **Coding Errors:** These are mistakes in the source code, including typos, syntax errors, logical errors, or incorrect algorithms, resulting in unexpected behavior or crashes.
  - **Testing Errors:** These are mistakes made during the testing process itself, such as inadequate test coverage, incorrect test cases, or flawed test data.

NATURE OF ERRORS

- **Classification by Nature:**
  - **Functional Errors:** These errors occur when the software fails to perform its intended functions or does not meet the specified requirements.
  - **Performance Errors:** These errors relate to the software's speed, responsiveness, or resource usage under various workloads. Performance issues can lead to slow response times, crashes, or unresponsiveness.
  - **Usability Errors:** These are issues with the software's user interface, user experience, or overall ease of use. Usability errors can make it difficult or frustrating for users to interact with the software effectively.
  - **Compatibility Errors:** These occur when the software doesn't function correctly on different platforms, operating systems, browsers, or hardware configurations.
  - **Security Errors:** These are vulnerabilities in the software that could be exploited by malicious actors to gain unauthorized access, steal data, or disrupt operations.
  - **Error Handling Errors:** These occur when the software fails to handle unexpected errors or exceptions gracefully, leading to crashes or unexpected behavior.

NATURE OF ERRORS

- **Examples of Common Errors:**
  - **Calculation Errors:** Incorrect formulas, arithmetic operator errors, data type mismatches leading to incorrect calculation results.
  - **Control Flow Errors:** Incorrect branching logic (if/else statements), infinite loops, or incorrect loop termination conditions.
  - **Logic Errors:** Incorrect algorithms, incorrect comparisons, or incorrect handling of boundary conditions.
  - **Integration Errors:** Problems arise when combining different software modules or components due to interface mismatches, incorrect data exchange, or incompatible versions.
  - **Boundary Condition Errors:** Errors occur when the software doesn't handle extreme values or edge cases correctly (e.g., dividing by zero, exceeding array bounds).

DEFINITIONS

- **Quality (in Software)**
  - **Definition:** The degree to which a software product meets or exceeds customer expectations and requirements. This includes aspects like functionality, reliability, usability, performance, security, maintainability, and portability.
  - **Importance:** High-quality software is essential for customer satisfaction, brand reputation, and minimizing long-term costs due to defects and maintenance.
- **Quality Assurance (QA)**
  - **Definition:** A set of planned and systematic activities implemented within the software development process to ensure that the final product meets specified quality requirements.
  - **Focus:** Preventing defects through proactive measures like process definition, standards establishment, reviews, and audits.
  - **Goal:** Build quality into the product from the start, rather than just finding and fixing defects later.

DEFINITIONS

- **Quality Control (QC)**
  - **Definition:** A set of activities focused on identifying defects in the software product.
  - **Focus:** Detecting and correcting defects through testing and inspection.
  - **Goal:** Ensure that the product meets quality standards before it is released to customers.
- **Quality Management (QM)**
  - **Definition:** The overarching discipline encompassing all activities and processes related to achieving and maintaining quality. It includes both QA and QC.
  - **Focus:** Establishing a quality policy, setting quality objectives, and implementing a quality management system.
  - **Goal:** Create a culture of quality within the organization, where everyone is committed to delivering high-quality products and services.

DEFINITIONS

- **Software Quality Assurance (SQA)**
  - **Definition:** Specifically refers to the application of quality assurance principles and practices to software development.
  - **Focus:** Same as QA, but tailored to the unique challenges and requirements of software development.
  - **Goal:** Ensure that software products are developed and delivered with the highest possible quality.

DEFINITIONS - KEY DIFFERENCE

Term	Focus	Goal	Activities
Quality	Attribute of a product	Meet customer expectations	N/A (result of QA/QC efforts)
QA	Prevention of defects	Build quality into the process	Process definition, standards, reviews, audits
QC	Detection and correction of defects	Meet quality standards	Testing, inspection
QM	Overall quality management	Create a culture of quality	Establish quality policy, objectives, system
SQA	QA specific to software	Ensure high-quality software	Same as QA, but tailored to software

VERIFICATION AND VALIDATION

- Software quality factors are the attributes that contribute to the overall quality of a software product. Verification and validation are two essential processes used to ensure that these quality factors are met throughout the software development lifecycle.

VERIFICATION AND VALIDATION

- Verification: Verification is the process of evaluating a software product at each stage of development to ensure that it meets the specified requirements and design specifications. It is often referred to as "Are we building the product right?". Verification activities include:
  - **Reviews:** Examining documents like requirements specifications, design documents, and code to identify errors, inconsistencies, or deviations from standards.
  - **Inspections:** Formalized reviews involving a team of experts who systematically analyze documents and code to identify defects.
  - **Walkthroughs:** Less formal reviews where a developer explains the code or design to a team to gather feedback and identify potential issues.
  - **Static Analysis:** Using automated tools to analyze code without executing it, to identify potential defects like coding standard violations, security vulnerabilities, or performance bottlenecks.

VERIFICATION AND VALIDATION

- Validation: Validation is the process of evaluating a software product after it is built to ensure that it meets the customer's needs and expectations. It is often referred to as "Are we building the right product?". Validation activities include:
  - **Testing:** Executing the software with various inputs and conditions to identify defects, assess its functionality, reliability, performance, security, usability, and other quality factors.
  - **User Acceptance Testing (UAT):** Involving end-users to test the software in a real-world environment to ensure it meets their needs and expectations.
  - **Beta Testing:** Releasing a pre-release version of the software to a limited group of users to gather feedback and identify any remaining issues.

VERIFICATION AND VALIDATION IN RELATION TO SOFTWARE QUALITY FACTORS

- Verification and validation play a crucial role in ensuring that software quality factors are met. For example:
  - **Functionality:** Verification ensures that the software functions according to the specified requirements, while validation ensures that the functions meet the user's needs.
  - **Reliability:** Verification checks for errors and inconsistencies in the code and design, while validation assesses the software's ability to perform consistently without failures.
  - **Usability:** Verification reviews the user interface design, while validation tests the software's ease of use with actual users.
  - **Performance:** Verification analyzes the code for potential performance bottlenecks, while validation measures the software's response time and resource utilization under different loads.
  - **Security:** Verification examines the code for security vulnerabilities, while validation tests the software's ability to withstand attacks and protect sensitive data.

VERIFICATION MECHANISMS

- **Reviews:**
  - **Peer Reviews:** Informal reviews where developers examine each other's work (code, design, etc.) to identify errors and suggest improvements.
  - **Walkthroughs:** Led by the author, the team steps through the work product (e.g., code) to understand its logic and identify potential issues.
  - **Inspections:** A more formal review process with defined roles (moderator, author, reader, etc.) and a focus on finding defects through checklists.
  - **Technical Reviews:** Formal evaluation of the technical aspects of the software by experts to ensure it meets the specified requirements.

VERIFICATION MECHANISMS

- **Static Analysis:**
  - **Code Analysis:** Automated tools analyze the code for potential errors, security vulnerabilities, performance bottlenecks, and adherence to coding standards.
  - **Data Flow Analysis:** Examines how data is used and modified throughout the program to detect potential issues like uninitialized variables or unused values.
  - **Control Flow Analysis:** Analyzes the possible paths of execution through the program to identify potential dead code, infinite loops, or unreachable code

VERIFICATION MECHANISMS

- **Formal Verification:**
  - **Model Checking:** Uses mathematical models to verify that a system meets specified properties, often used in safety-critical applications.
  - **Theorem Proving:** Employs formal logic to prove the correctness of algorithms or system properties.

VALIDATION MECHANISMS

- **Testing:**
  - **Unit Testing:** Tests individual components or modules in isolation.
  - **Integration Testing:** Tests the interaction between integrated components or modules.
  - **System Testing:** Tests the entire system as a whole to ensure it meets requirements.
  - **Acceptance Testing:** Tests conducted by users or customers to determine if the system meets their needs and expectations.
  - **Regression Testing:** Retesting after changes to ensure that existing functionality still works correctly.

VALIDATION MECHANISMS

- **Dynamic Analysis:**
  - **Code Coverage Analysis:** Measures how much of the code is executed during testing to identify areas that need more testing.
  - **Performance Testing:** Evaluates the system's speed, responsiveness, and stability under different workloads.
  - **Security Testing:** Attempts to identify vulnerabilities and weaknesses in the system's security mechanisms.
  - **Usability Testing:** Assesses how easy and intuitive the system is to use by real users.

WHICH TO CHOOSE

- The specific V&V mechanisms used will depend on various factors, including the nature of the software, its criticality, the development methodology, budget and time constraints, and regulatory requirements. A combination of verification and validation techniques is typically employed to ensure comprehensive coverage and increase confidence in the software's quality.

WHICH TO CHOOSE

- The specific V&V mechanisms used will depend on various factors, including the nature of the software, its criticality, the development methodology, budget and time constraints, and regulatory requirements. A combination of verification and validation techniques is typically employed to ensure comprehensive coverage and increase confidence in the software's quality.

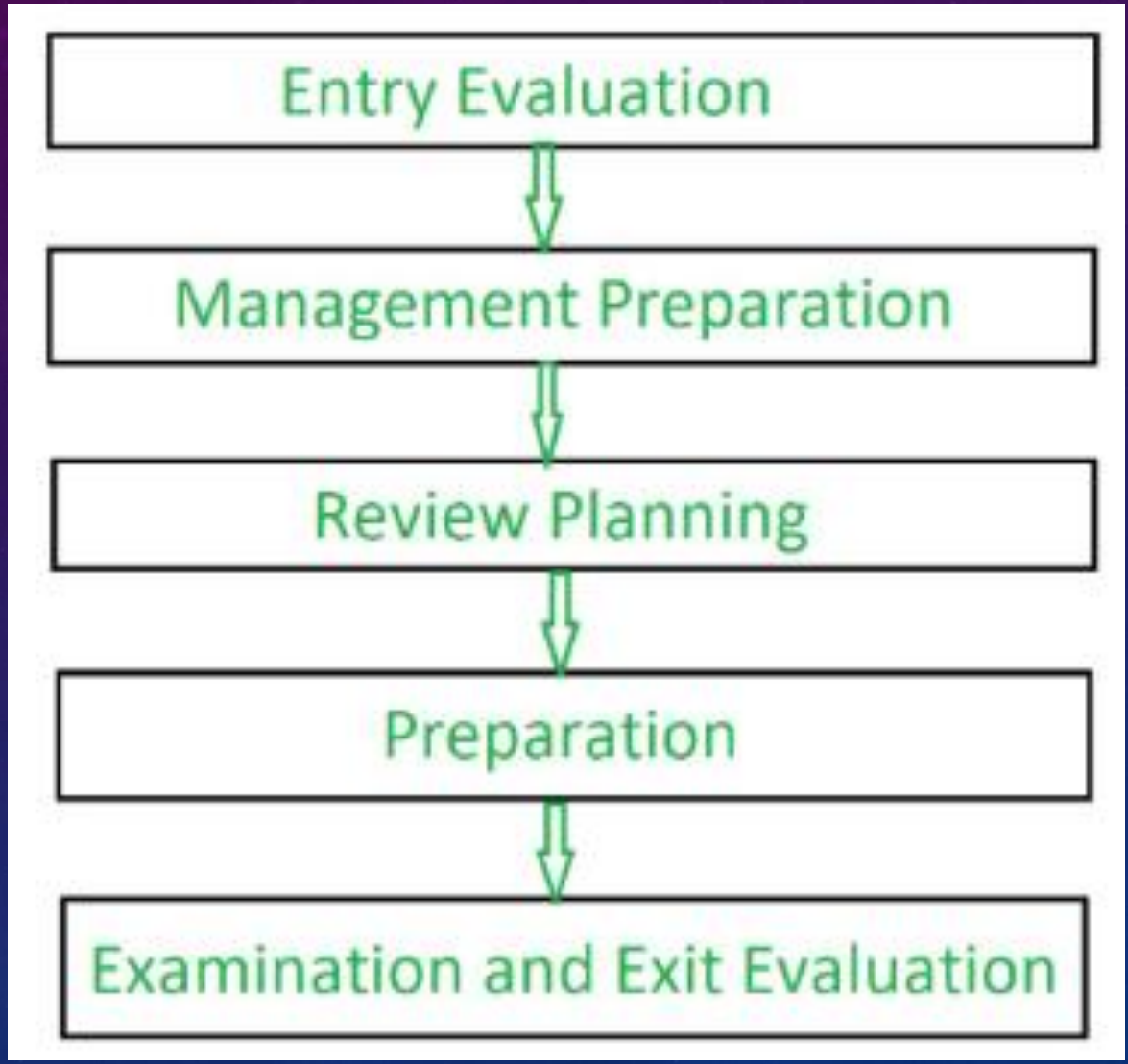
SOFTWARE REVIEW

- **Software Review** is a systematic inspection of software by one or more individuals who work together to find and resolve errors and defects in the software during the early stages of the Software Development Life Cycle (SDLC). A software review is an essential part of the Software Development Life Cycle (SDLC) that helps software engineers in validating the quality, functionality, and other vital features and components of the software. It is a whole process that includes testing the software product and it makes sure that it meets the requirements stated by the client.
- Usually performed manually, software review is used to verify various documents like requirements, system designs, codes, test plans, and test cases.

SOFTWARE REVIEW - OBJECTIVE

- The objective of the software review is:
  - To improve the productivity of the development team.
  - To make the testing process time and cost-effective.
  - To make the final software with fewer defects.
  - To eliminate the inadequacies.

SOFTWARE REVIEW - PROCESS



SOFTWARE REVIEW - PROCESS

- **Entry Evaluation:** By confirming documentation, fulfilling entry requirements and assessing stakeholder and team preparation, you can determine the software's availability.
- **Management Preparation**: To get ready for the review process, assign roles, gather resources and provide brief management.
- **Review Planning:** Establish the review's goals and scope, invite relevant parties and set a time for the meeting.
- **Preparation**: Distribute appropriate resources, give reviewers time to get familiar and promote issue identification to help them prepare.
- **Examination and Exit Evaluation:** Reviewers should collaborate to examine the results, record concerns, and encourage candid communication in meetings.
  It assess the results, make remedial plans based on flaws that have been reported and assess the process's overall efficacy.

SOFTWARE REVIEW - TYPES

- **Software Peer Review**
  - Peer review is the process of assessing the technical content and quality of the product and it is usually conducted by the author of the work product along with some other developers.
    Peer review is performed in order to examine or resolve the defects in the software, whose quality is also checked by other members of the team.

SOFTWARE REVIEW - TYPES

- **Software Peer Review**
- Peer Review has following types:
  - **Code Review:** Computer source code is examined in a systematic way.
  - **Pair Programming:** It is a code review where two developers develop code together at the same platform.
  - **Walkthrough:** Members of the development team is guided by author and other interested parties and the participants ask questions and make comments about defects.
  - **Technical Review:** A team of highly qualified individuals examines the software product for its client's use and identifies technical defects from specifications and standards.
  - **Inspection:** In inspection the reviewers follow a well-defined process to find defects.

SOFTWARE REVIEW - TYPES

- **Software Management Review**
  - Software Management Review evaluates the work status. In this section decisions regarding downstream activities are taken.
- **Software Audit Review**
  - Software Audit Review is a type of external review in which one or more critics, who are not a part of the development team, organize an independent inspection of the software product and its processes to assess their compliance with stated specifications and standards. This is done by managerial level people.

SOFTWARE REVIEW - ADVANTAGES

- Defects can be identified earlier stage of development (especially in formal review).
- Earlier inspection also reduces the maintenance cost of software.
- It can be used to train technical authors.
- It can be used to remove process inadequacies that encourage defects.

TEST CASE DESIGN

- Test case design is a crucial phase in software testing that involves creating a set of conditions or variables under which a tester can determine whether a software application is working correctly.
- **1. Definition and Purpose:**
  - **Test Case**: A test case is a specific set of conditions or variables developed to verify whether a software application functions correctly.
  - **Purpose**: The primary purpose is to identify defects, ensure functionality, validate requirements, and improve software quality.

TEST CASE DESIGN - COMPONENTS

2. Components of a Test Case:

- **Test Case ID**: A unique identifier for the test case.
- **Test Description**: A brief description of the test case.
- **Pre-conditions**: Any prerequisites or conditions that must be met before executing the test case.
- **Test Steps**: Step-by-step instructions to execute the test.
- **Test Data**: Data required for executing the test case.
- **Expected Result**: The expected outcome of the test.
- **Actual Result**: The actual outcome after executing the test.
- **Status**: Pass or fail based on the comparison of expected and actual results.
- **Post-conditions**: Any conditions that must be met after executing the test case.
- **Remarks/Notes**: Additional information or observations.

TEST CASE DESIGN - TECHNIQUES

Black Box Testing:

- **Equivalence Partitioning**: Dividing input data into equivalent partitions to reduce the number of test cases.
- **Boundary Value Analysis**: Testing at the boundaries between partitions.
- **Decision Table Testing**: Using decision tables to represent combinations of inputs and their corresponding outputs.
- **State Transition Testing**: Testing the software's response to state changes.
- **Use Case Testing**: Designing test cases based on use case scenarios.

TEST CASE DESIGN - TECHNIQUES

White Box Testing:

- **Statement Coverage**: Ensuring each statement in the code is executed at least once.
- **Branch Coverage**: Ensuring each branch (decision point) is executed at least once.
- **Path Coverage**: Ensuring all possible paths through the code are executed.
- **Loop Coverage**: Ensuring loops are executed with zero, one, and multiple iterations.

TEST CASE DESIGN - TECHNIQUES

Experience-Based Techniques:

- **Error Guessing**: Using experience to guess the likely locations of defects.
- **Exploratory Testing**: Simultaneous learning, test design, and execution

TEST CASE DESIGN - PROCESS

- **Requirement Analysis**: Understand and analyze the requirements and specifications.
- **Define Test Objectives**: Determine what needs to be tested.
- **Identify Test Conditions**: Identify conditions to be tested.
- **Design Test Cases**: Develop detailed test cases based on identified conditions.
- **Review and Validate**: Review test cases with stakeholders to ensure coverage and correctness.
- **Execute Test Cases**: Execute test cases and document results.
- **Traceability**: Ensure traceability between test cases and requirements

TEST CASE DESIGN – BEST PRACTICES

- **Clear and Concise**: Write test cases clearly and concisely to avoid ambiguity.
- **Reusability**: Design test cases to be reusable for future testing cycles.
- **Coverage**: Ensure comprehensive coverage of all requirements and scenarios.
- **Maintenance**: Regularly update test cases to reflect changes in requirements.
- **Automation**: Automate repetitive and regression test cases where feasible.
- **Traceability**: Maintain traceability between test cases and requirements to ensure all requirements are tested.

TEST CASE DESIGN – CHALLENGES

- **Incomplete Requirements**: Dealing with incomplete or unclear requirements.
- **Complex Scenarios**: Designing test cases for complex and interdependent scenarios.
- **Dynamic Changes**: Managing changes in requirements or functionalities during the development lifecycle.
- **Time Constraints**: Limited time available for thorough testing.
- **Tool Limitations**: Limitations of testing tools in use.

TEST CASE DESIGN – TOOLS

- **Test Management Tools**: JIRA, TestRail, HP ALM.
- **Automation Tools**: Selenium, QTP, TestComplete.
- **Requirement Management Tools**: IBM Rational DOORS, RequisitePro.
- Test case design is a meticulous process that requires a clear understanding of the application under test, meticulous planning, and attention to detail. Effective test case design can significantly enhance the quality of the software and ensure that it meets the specified requirements and expectations.