18MCA44E SOFTWARE TESTING

UNIT I - INTRODUCTION

FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,

Postgraduate Department of Computer Applications,

Government Arts College (Autonomous),

Coimbatore-641018.

Year	Subject Title	Semester	Sub. Code
2018 - 2019 Onwards	Elective 1.2: SOFTWARE TESTING	IV	18MCA44E

Objective: To motivate the students as well as enrich their knowledge about the concepts of testing and its documentation.

UNIT I: Developing a test approach - Addressing software system business risk - Defining a software system strategy - Developing software system testing tactics - Testing a software using a life cycle methodology - Requirements phase testing.

UNIT II: Design phase testing - Program phase testing - Desk debugging and program peer view test tools - Evaluating test results - Installation phase testing - Acceptance testing.

UNIT III: Testing methodology for software maintenance - Testing the correctness of the installing a software change - Testing the validity of a software cost estimate - Testing the progress of the software system - Inspecting test plan and test cases.

UNIT IV: Accessing Client-Server and LAN risks - A testing strategy for a rapid prototyping - Testing techniques - Testing tools.

UNIT V: Test documentation - Reporting test results - Final test reporting - Evaluating test effectiveness - Use of testing metrices - Improving test process.

TEXT BOOKS:

1. William Perry, “Effective Methods for Software Testing”, John Wiley & Sons, Inc., 1995.

REFERENCE BOOKS:

1. Renu & Pradeep “Software Testing: Methodologies, Tools and Processes”, Tata McGraw Hill Publishing Co. Ltd.

Software Testing

- **What is it?**

"Software Testing the process to validate the outcome of software product by identifying defects to meet with expected results and specified requirements of the customer"

- **What are the techniques involved in software testing?**

- **Verification (QA)** is set of activities for ensuring **quality in process** by which products are developed. **Aim** to prevent defects with **focus on process** used to make the products.

Method of Verifications: Walkthrough, Inspection, Review (Static Testing)

- **Validation (QC)** is set of activities for ensuring **quality in products itself**. Finding defects in developed product. **Aim** to **identify defects/errors** in the finished products.

Method of Validation: Testing, End Users (Dynamic Testing)

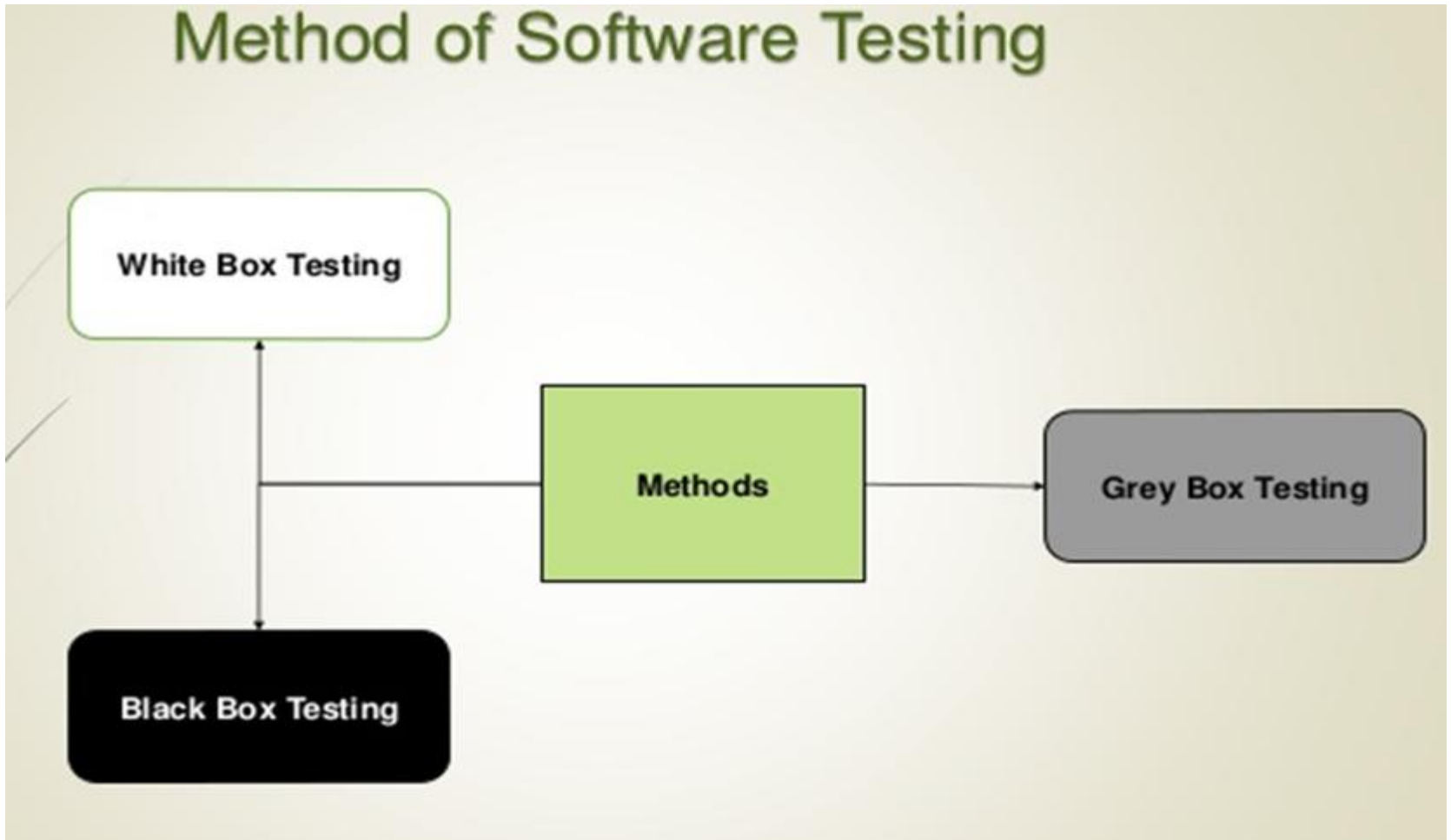
Software Testing

- **Why we need it?**

- ✓ Software testing is really required to point out the **defects and errors** that were made during the development phases.
- ✓ It's essential since it makes sure of the **Customer's reliability** and their **satisfaction** in the application.
- ✓ It is very important to ensure the **Quality of the product**. Quality product delivered to the customers helps in **gaining their confidence.**
- ✓ Testing is required for an **effective performance** of software application or product.
- ✓ It's important to ensure that the application **should not result into any failures** because it can be **very expensive in the future** or in the later stages of the development.
- ✓ It's required to **stay in the business**.

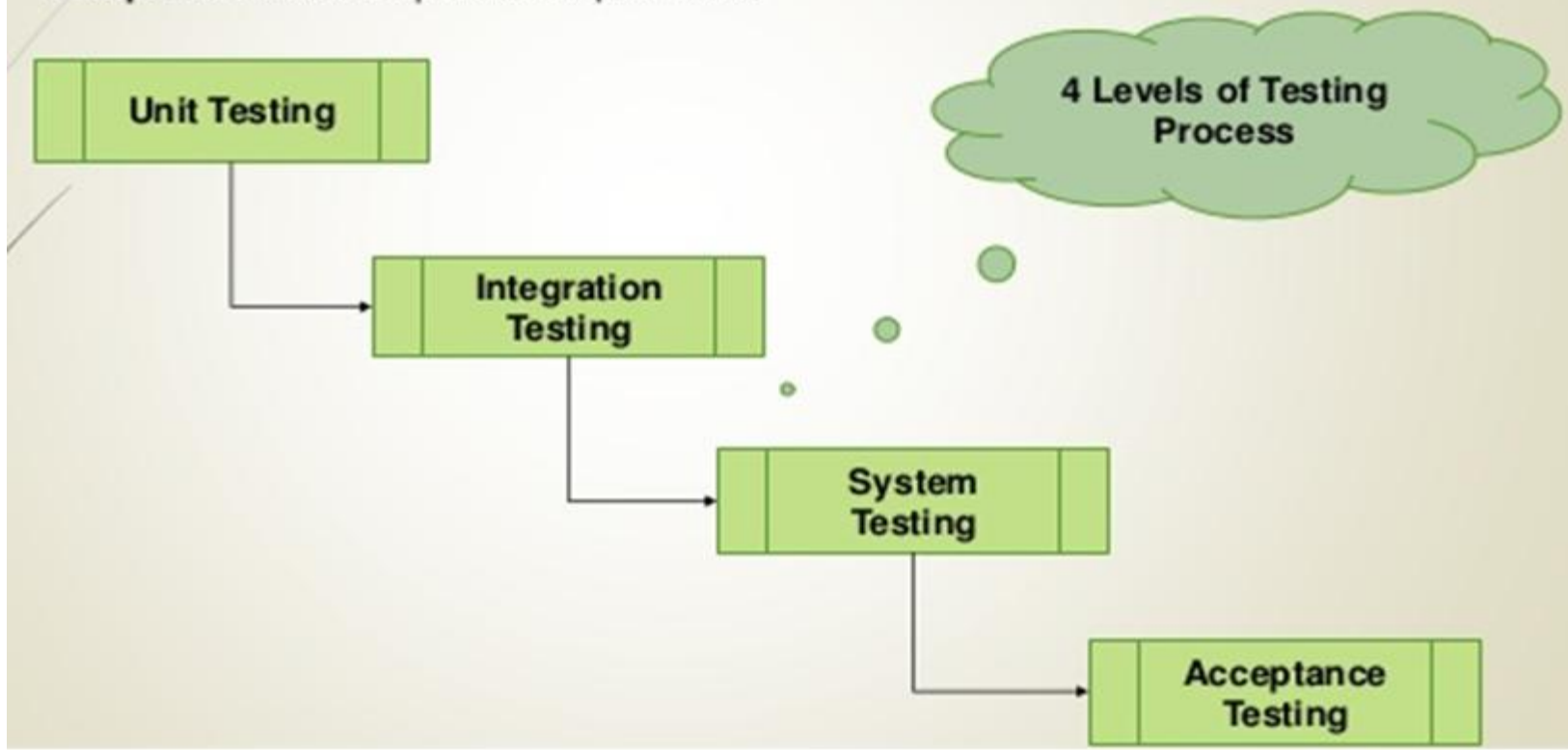
Skills Required for Software Testing

- Analytical and logical thinking
- The ability to imagine business situations
- A sense of logical curiosity and creativity
- A "Glocal" approach
- Ability to apply basic and fundamental knowledge
- Continue to learn
- Respect for truth and intellectual integrity
- Planning, time management skills
- Effective communication skills


Method of Software Testing
White Box Testing
Methods
Grey Box Testing
Black Box Testing

Levels of Software Testing

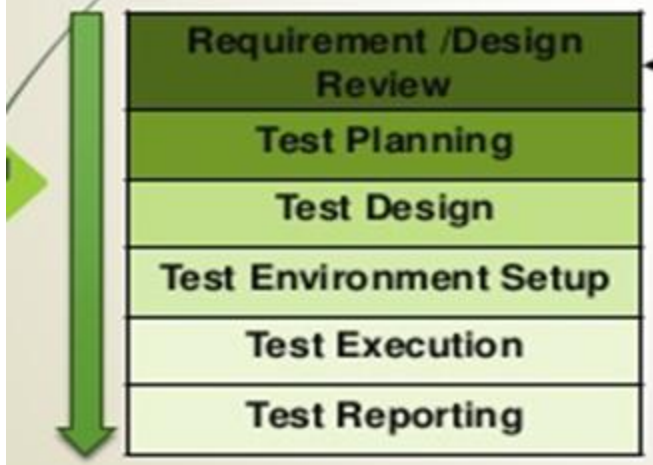
"A level of the software testing process where individual units/components of a software/system are tested, The purpose of this test is to **evaluate the system's compliance** with the specified requirements"



SDLC vs STLC

❑ **SDLC:** Software Development **Life Cycle**

- ✓ It is a **systematic approach** to develop software.
- ✓ It is a **conceptual model** used in project management that describes the stages involved in an information system development project, from an initial feasibility study through maintenance of the completed application.



❑ **STLC:** Software Test **Life Cycle**

- ✓ A **software testing life cycle** (**STLC**) is a set of steps used to test software products. Software testing is a critical part of preparing software for use, and a **STLC** helps make this process more sophisticated, consistent and effective

Test Management

- **What is Configuration Management (CM)?**

"**Configuration Management (CM)** is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life."

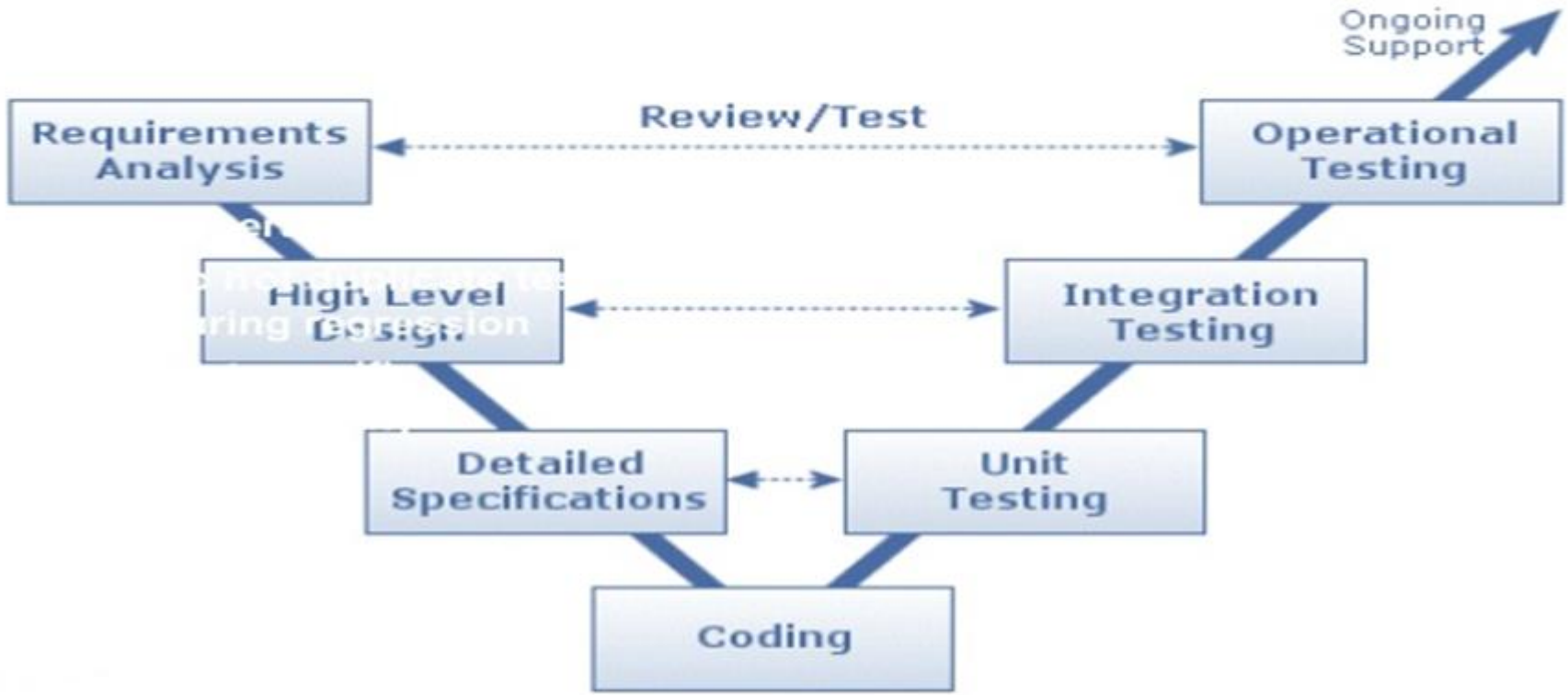
- **Why it is important in software testing?**

- It facilitates the ability to communicate status of documents and code as well as changes that have been made to them. High-quality of the software that has been tested and used, makes it a reusable asset and saves development costs.
- Increased efficiencies, stability and control by improving visibility and tracking.
- The ability to define and enforce formal policies and procedures that govern asset identification, status monitoring, and auditing.

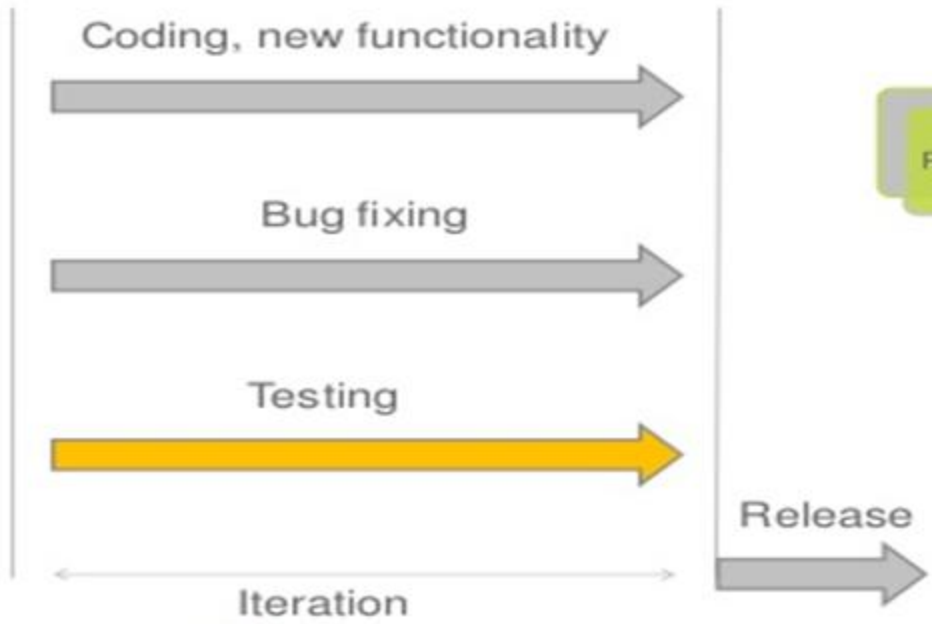
Deliverables of Software Testing

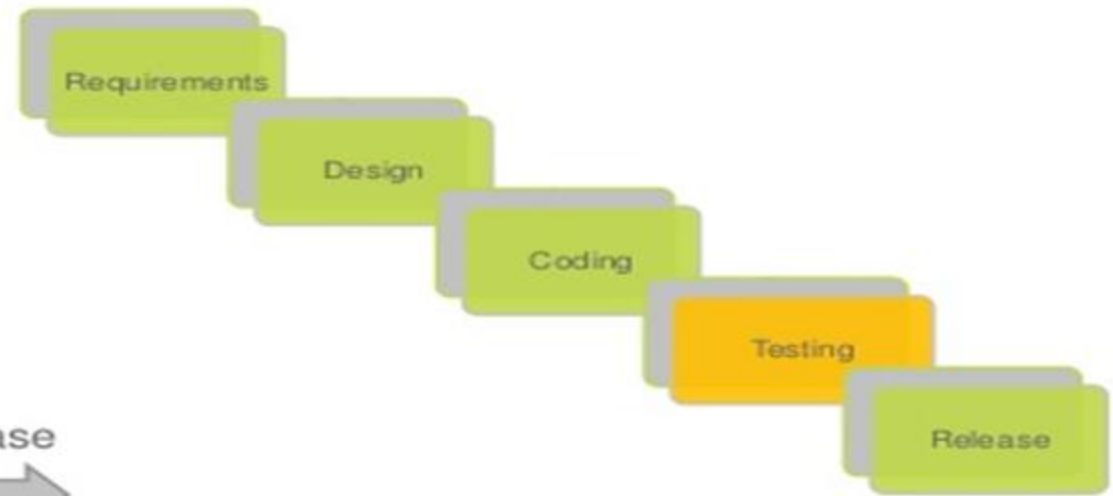
Test Plan	A test plan is a detailed document that outlines the test strategy, testing objectives, resources (manpower, software, hardware) required for testing, test schedule, test estimation and test deliverables.
Test Strategy	A test strategy is an outline that describes the testing approach of the software development cycle. It is created to inform project managers, testers, and developers about some key issues of the testing process.
Test Scenario	A Test Scenario is any functionality that can be tested. It is also called Test Condition or Test Possibility. As a tester, you may put yourself in the end user's shoes and figure out the real-world scenarios and use cases of the Application Under Test.
Test Case	A test case is a document, which has a set of test data, preconditions, expected results and post conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

V Model


Ongoing Support
Requirements Analysis
Review/Test
Operational Testing
High Level Design
Integration Testing
Detailed Specifications
Unit Testing
Coding

Agile against waterfall

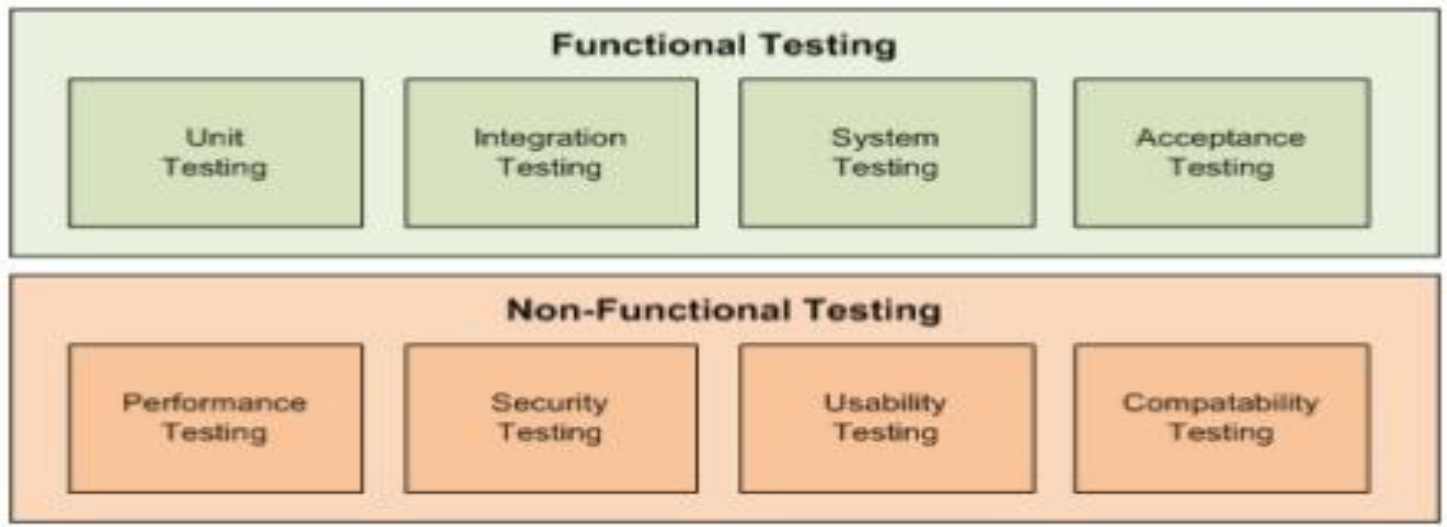




What are Software Testing Methodologies?

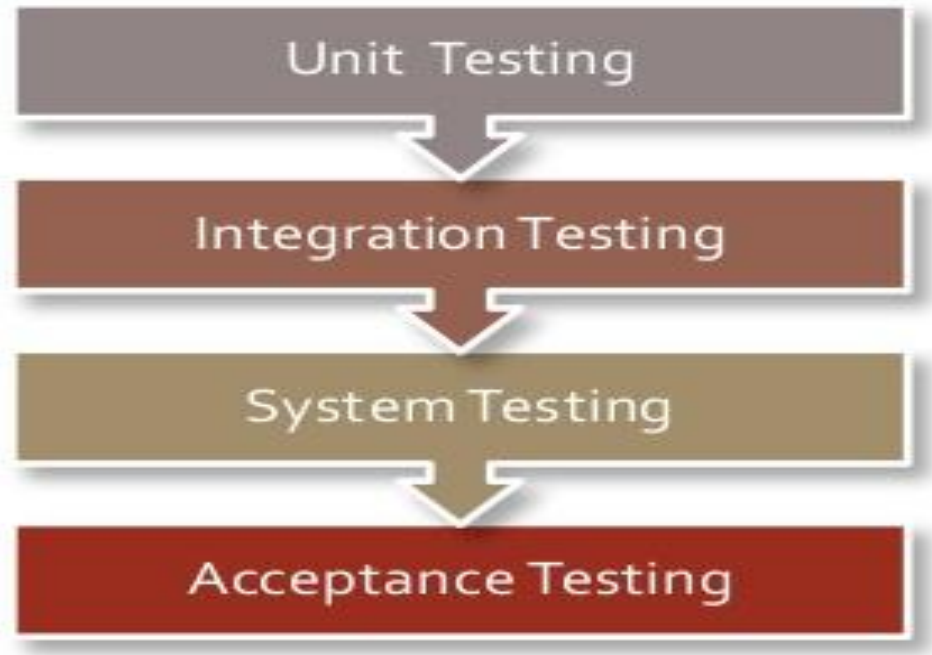
Software testing methodologies are the different approaches and ways of ensuring that a software application is fully tested.

Software testing methodologies encompass in two parts:



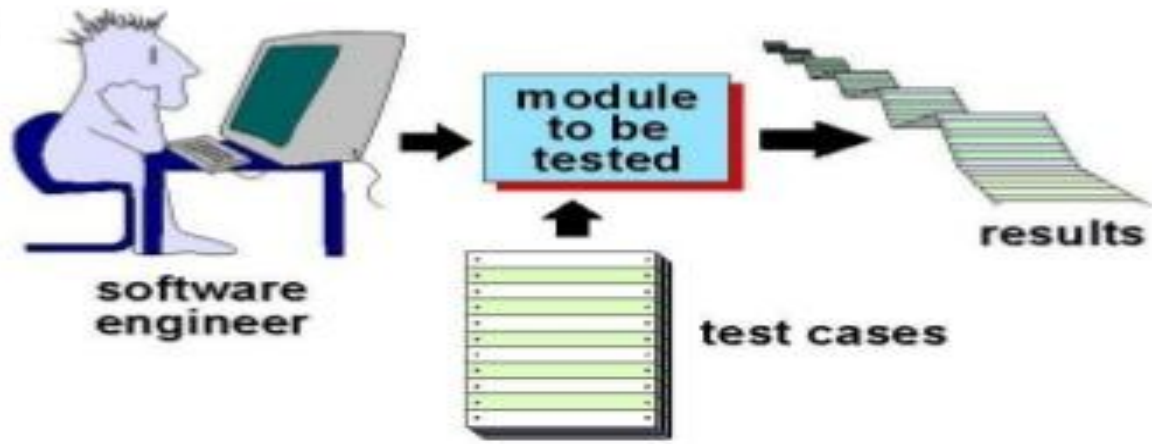
Functional Testing

The functional testing part of a testing methodology is typically broken down into four components - unit testing, integration testing, system testing and acceptance testing – usually executed in this order.



Unit Testing

- **Purpose:** To verify that the component/module functions work properly.
- **Check:**
  - internal data structures
  - Logic
  - boundary conditions for input/output data
- **Method:** White box testing
- **Done by:** Developers



Integration Testing

- **Purpose:** To verify that modules/components witch have been successfully unit tested when integrated together to perform specific tasks and activities work properly.
- This testing is usually done with a combination of automated functional tests and manual testing
- **Method:** Black box testing
- **Done by:** Independent Test Team

System Testing

- **Purpose:** Verifies that all system elements work properly and that overall system function and performance has been achieved.
- This test is carried out by interfacing the hardware and software components of the entire system (that have been previously unit tested and integration tested), and then testing it as a whole.
- **Method:** Black box testing
- **Done by:** Independent Test Team

Acceptance Testing

- **Purpose:** To ensure that the software that has been developed operates as expected and meets all user requirements.
- There are two types of acceptance testing:
  - **Alpha Testing:** It is carried out by the members of the development team, known as internal acceptance testing.
  - **Beta Testing**: It is carried out by the customer, known as external acceptance testing.
- **Method:** Black box testing

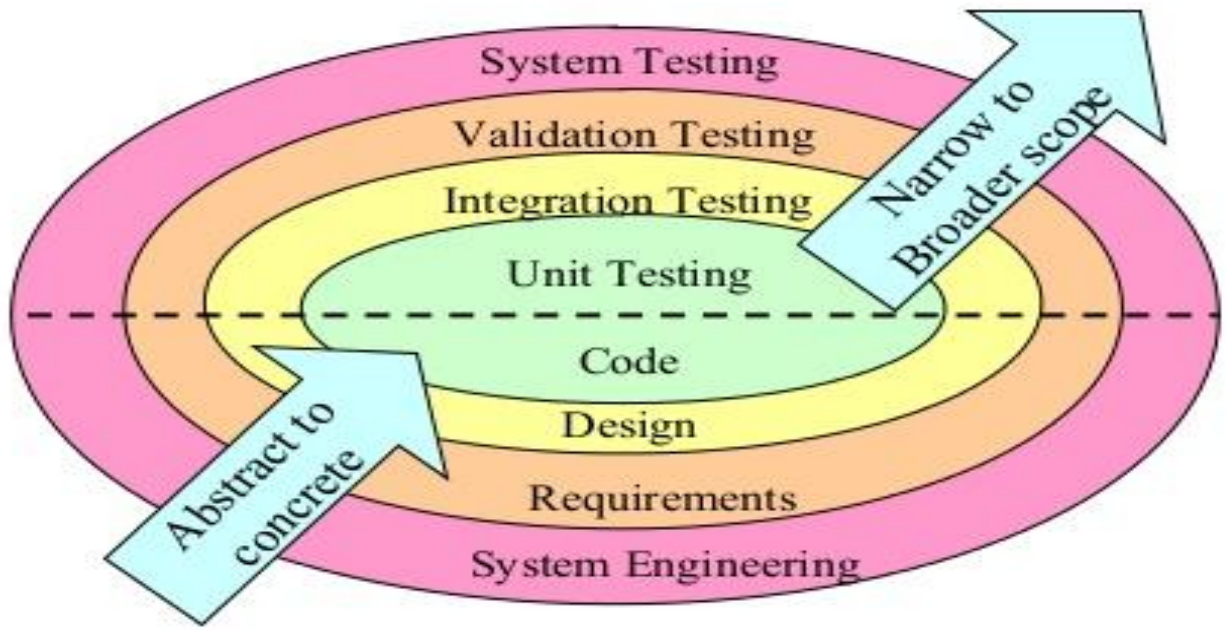
General Characteristics of Strategic Testing

- To perform effective testing, a software team should conduct effective formal technical reviews
- Testing begins at the component level and work outward toward the integration of the entire computer-based system
- Different testing techniques are appropriate at different points in time
- Testing is conducted by the developer of the software and (for large projects) by an independent test group
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy

Verification and Validation

- Software testing is part of a broader group of activities called verification and validation that are involved in software quality assurance
- Verification (Are the algorithms coded correctly?)
  - The set of activities that ensure that software correctly implements a specific function or algorithm
- Validation (Does it meet user requirements?)
  - The set of activities that ensure that the software that has been built is traceable to customer requirements

A Strategy for Testing Conventional Software



Levels of Testing for Conventional Software

- Unit testing
  - Concentrates on each component/function of the software as implemented in the source code
- Integration testing
  - Focuses on the design and construction of the software architecture
- Validation testing
  - Requirements are validated against the constructed software
- System testing
  - The software and other system elements are tested as a whole

Testing Strategy applied to Object-Oriented Software

- Must broaden testing to include detections of errors in analysis and design models
- Unit testing loses some of its meaning and integration testing changes significantly
- Use the same philosophy but different approach as in conventional software testing
- Test "in the small" and then work out to testing "in the large"
  - Testing in the small involves class attributes and operations; the main focus is on communication and collaboration within the class
  - Testing in the large involves a series of regression tests to uncover errors due to communication and collaboration among classes
- Finally, the system as a whole is tested to detect errors in fulfilling requirements

Unit Testing

- Focuses testing on the function or software module
- Concentrates on the internal processing logic and data structures
- Is simplified when a module is designed with high cohesion
  - Reduces the number of test cases
  - Allows errors to be more easily predicted and uncovered
- Concentrates on critical modules and those with **high cyclomatic complexity** when testing resources are limited

Targets for Unit Test Cases

- Module interface
  - Ensure that information flows properly into and out of the module
- Local data structures
  - Ensure that data stored temporarily maintains its integrity during all steps in an algorithm execution
- Boundary conditions
  - Ensure that the module operates properly at boundary values established to limit or restrict processing
- Independent paths (basis paths)
  - Paths are exercised to ensure that all statements in a module have been executed at least once
- Error handling paths
  - Ensure that the algorithms respond correctly to specific error conditions

Drivers and Stubs for Unit Testing

- Driver
  - A simple main program that accepts test case data, passes such data to the component being tested, and prints the returned results
- Stubs
  - Serve to replace modules that are subordinate to (called by) the component to be tested
  - It uses the module's exact interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing testing
- Drivers and stubs both represent overhead
  - Both must be written but don't constitute part of the installed software product

Integration Testing

- Defined as a systematic technique for constructing the software architecture
  - At the same time integration is occurring, conduct tests to uncover errors associated with interfaces
- Objective is to take unit tested modules and build a program structure based on the prescribed design
- Two Approaches
  - Non-incremental Integration Testing
  - Incremental Integration Testing

Non-incremental Integration Testing

- Commonly called the "Big Bang" approach
- All components are combined in advance
- The entire program is tested as a whole
- Chaos results
- Many seemingly-unrelated errors are encountered
- Correction is difficult because isolation of causes is complicated
- Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop

Incremental Integration Testing

- Three kinds
  - Top-down integration
  - Bottom-up integration
  - Sandwich integration
- The program is constructed and tested in small increments
- Errors are easier to isolate and correct
- Interfaces are more likely to be tested completely
- A systematic test approach is applied

Top-down Integration

- Modules are integrated by moving downward through the control hierarchy, beginning with the main module
- Subordinate modules are incorporated in either a depth-first or breadth-first fashion
  - DF: All modules on a major control path are integrated
  - BF: All modules directly subordinate at each level are integrated
- Advantages
  - This approach verifies major control or decision points early in the test process
- Disadvantages
  - Stubs need to be created to substitute for modules that have not been built or tested yet; this code is later discarded
  - Because stubs are used to replace lower level modules, no significant data flow can occur until much later in the integration/testing process

Bottom-up Integration

- Integration and testing starts with the most atomic modules in the control hierarchy
- Advantages
  - This approach verifies low-level data processing early in the testing process
  - Need for stubs is eliminated
- Disadvantages
  - Driver modules need to be built to test the lower-level modules; this code is later discarded or expanded into a full-featured version
  - Drivers inherently do not contain the complete algorithms that will eventually use the services of the lower-level modules; consequently, testing may be incomplete or more testing may be needed later when the upper level modules are available

Sandwich Integration

- Consists of a combination of both top-down and bottom-up integration
- Occurs both at the highest level modules and also at the lowest level modules
- Proceeds using functional groups of modules, with each group completed before the next
  - High and low-level modules are grouped based on the control and data processing they provide for a specific program feature
  - Integration within the group progresses in alternating steps between the high and low level modules of the group
  - When integration for a certain functional group is complete, integration and testing moves onto the next group
- Reaps the advantages of both types of integration while minimizing the need for drivers and stubs
- Requires a disciplined approach so that integration doesn't tend towards the "big bang" scenario

Regression Testing

- Each new addition or change to baselined software may cause problems with functions that previously worked flawlessly
- Regression testing re-executes a small subset of tests that have already been conducted
  - Ensures that changes have not propagated unintended side effects
  - Helps to ensure that changes do not introduce unintended behavior or additional errors
  - May be done manually or through the use of automated capture/playback tools
- Regression test suite contains three different classes of test cases
  - A representative sample of tests that will exercise all software functions
  - Additional tests that focus on software functions that are likely to be affected by the change
  - Tests that focus on the actual software components that have been changed

Smoke Testing

- Taken from the world of hardware
  - Power is applied and a technician checks for sparks, smoke, or other dramatic signs of fundamental failure
- Designed as a pacing mechanism for time-critical projects
  - Allows the software team to assess its project on a frequent basis
- Includes the following activities
  - The software is compiled and linked into a build
  - A series of breadth tests is designed to expose errors that will keep the build from properly performing its function
    - The goal is to uncover "show stopper" errors that have the highest likelihood of throwing the software project behind schedule
  - The build is integrated with other builds and the entire product is smoke tested daily
    - Daily testing gives managers and practitioners a realistic assessment of the progress of the integration testing
  - After a smoke test is completed, detailed test scripts are executed

Benefits of Smoke Testing

- Integration risk is minimized
  - Daily testing uncovers incompatibilities and show-stoppers early in the testing process, thereby reducing schedule impact
- The quality of the end-product is improved
  - Smoke testing is likely to uncover both functional errors and architectural and component-level design errors
- Error diagnosis and correction are simplified
  - Smoke testing will probably uncover errors in the newest components that were integrated
- Progress is easier to assess
  - As integration testing progresses, more software has been integrated and more has been demonstrated to work
  - Managers get a good indication that progress is being made

Alpha and Beta Testing

- Alpha testing
  - Conducted at the developer's site by end users
  - Software is used in a natural setting with developers watching intently
  - Testing is conducted in a controlled environment
- Beta testing
  - Conducted at end-user sites
  - Developer is generally not present
  - It serves as a live application of the software in an environment that cannot be controlled by the developer
  - The end-user records all problems that are encountered and reports these to the developers at regular intervals
- After beta testing is complete, software engineers make software modifications and prepare for release of the software product to the entire customer base

1. What is Risk

- A risk is a potential problem – it might happen and it might not, this is uncertainty.
- We don't know whether a particular event will occur or no but if it does has a negative impact on a project.
- An example would be that team is working on a project and the developer walks out of project and other person is recruited in his place and he doesn't work on the same platform and converts it into the platform he is comfortable with. Now the project has to yield the same result in the same time span. Whether they will be able to complete the project on time. That is the risk of schedule .

2. Definitions of Risks

- Risk is the probability of suffering loss.
- Risk provides an opportunity to develop the project better.
- Risk exposure= Size (loss)* probability of (loss)
- There is a difference between a Problem and Risk
- Problem is some event which has already occurred but risk is something that is unpredictable.

3. Risk Categorization

- Project risks
  - They threaten the project plan
  - If they become real, it is likely that the project schedule will slip and that costs will increase
- Technical risks
  - They threaten the quality and timeliness of the software to be produced
  - If they become real, implementation may become difficult or impossible
- Business risks
  - If they become real, they Problem occuring in the project or the product

4. Other Risk Categories

- Known risks
  - Those risks that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date)
- Predictable risks
  - Those risks that are extrapolated from past project experience (e.g., past turnover)
- Unpredictable risks
  - Those risks that can and do occur, but are extremely difficult to identify in advance

5. Negative Impact of Risk

- Diminished quality of product
- Increased cost
- Delayed completion
- Project failure

6. Risk management

- The project should be managed in such a way that the risks don't affect the project in a big way.
- Risk Management is a methodology that helps managers make best use of their available resources
- By using various paradigms, principles we can manage the risks.

7. The Principles of Risk Management

- Global Perspective: In this we look at the larger system definitions, design and implementation. We look at the opportunity and the impact the risk is going to have .
- Forward Looking View: Looking at the possible uncertainties that might creep up. We also think for the possible solutions for those risks that might occur in the future.
- Open Communication: This is to enable the free flow of communication between in the customers and the team members so that they have clarity about the risks.
- Integrated management: In this phase risk management is made an integral part of project management.
- Continous process :In this phase the risks are tracked continuously throughout the risk management paradigm.

8. Risk management paradigm

- 1.Identify: Search for the risks before they create a major problem
- 2.Analyze: understand the nature , kind of risk and gather information about the risk.
- 3.Plan: convert them into actions and implement them.
- 4.Track: we need to monitor the necessary actions.
- 5.Control: Correct the deviation and make any necessary amendments.
- 6.Communicate: Discuss about the emerging risks and the current risks and the plans to be undertaken.

9. Risk Management in Project management:

Basically project management deals with following :

- 1. Planning: Looking for the desired results, the strategies to be applied.
- 2. Organizing: Getting all the things together so that the desired results are obtained. By organizing the efficiency is increased and lot of time is saved.
- 3. Directing: Communication takes place and exchange of ideas is formatted in this phase.
- 4. Controlling: In the last phase feedback and evaluation is done.

10. How To Manage the Risks

1. Determine risk sources and Categories.
2. Determine Risk Parameters
3. Establish a Risk Management Strategy
4. Identify Risks
5. Evaluate and prioritize the risks.
6. Develop and Implement Risk mitigation plans

11. THE RMMM PLAN

- A risk management strategy can be included in the software project plan or the risk management steps can be organized into a separate ***Risk Mitigation, Monitoring and Management Plan.***
- The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.
- Some software teams do not develop a formal RMMM document. Rather, each risk is documented individually using a *risk information sheet*

THANK YOU

This content is taken from the text books and reference books prescribed in the syllabus.