SOFTWARE ENGINEERING LAB

Assignment 1: Part 1

Group No. : 19

* + Alok Ranjan(2022CSB091)
  + Ranveer Kumar(2022CSB092)
  + Karan Kumar(2022CSB093)
  + Rajesh Kumar(2022CSB094)

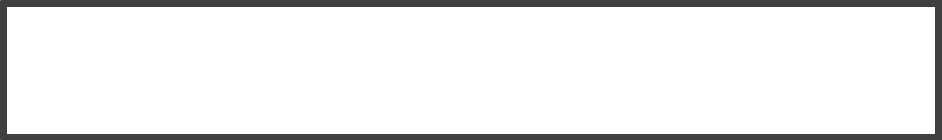


**SOFTWARE DEVELOPMENT LIFE CYCLE**

**Software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software.** SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step. The **goal of the SDLC life cycle model** is to deliver high-quality, maintainable software that meets the user’s requirements. SDLC in software engineering models outlines the plan for each stage so that each stage of the software development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users’ requirements. In this article we will see Software Development Life Cycle (SDLC) in detail.



SDLC specifies the task(s) to be performed at various stages by a software engineer or developer. It ensures that the end product is able to meet the customer’s expectations and fits within the overall budget. Hence, it’s vital for a software developer to have prior knowledge of this software development process. SDLC is a collection of these six stages, and the stages of SDLC are as follows:



**FEASIBILITY STUDY**

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of the important four stages of the Software Project Management Process.

As the name suggests, a feasibility study is the feasibility analysis or it is a measure of the software product in terms of how beneficial product development will be for the organization from a practical point of view.

Feasibility study is carried out based on many purposes to analyze whether a software product will be right in terms of development, implementation, contribution of project to the organization etc.

**Types of Feasibility Study-**

**Technical Feasibility**: In Technical Feasibility current resources both hardware software along with required technology are analysed /assessed to develop the project.

**Operational Feasibility**: It ensures the system is practical, user-friendly, and aligns with business operations.

**Economic Feasibility**: In Economic Feasibility study cost and benefit of the project is analysed.

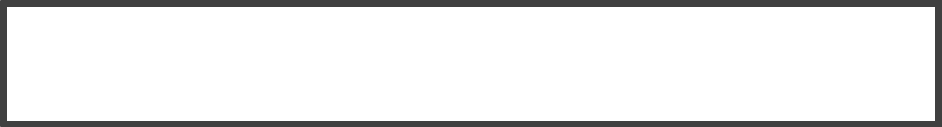
**Legal Feasibility**: In Legal Feasibility study project is analysed in legality point of view.

**Cultural and Political Feasibility**: This section assesses how the software project will affect the political environment and organizational culture.

**Market Feasibility**: This refers to evaluating the market’s willingness and ability to accept the suggested software system.

**Resource Feasibility**: This method evaluates if the resources needed to complete the software project successfully are adequate and readily available.

**Schedule Feasibility**: In Schedule Feasibility Study mainly deadlines are analyzed for proposed projects which includes how much time teams will take to complete the final project.

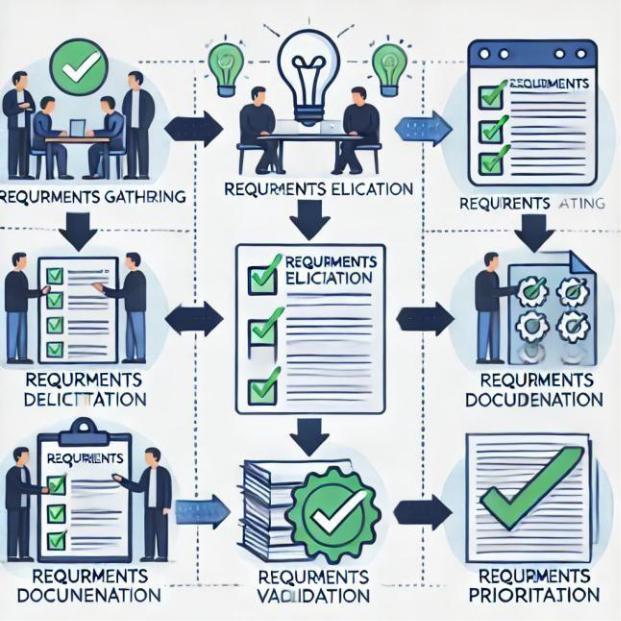
**REQUIREMENT ANALYSIS**

Requirement Analysis is one of the most crucial phases in the Software Development Life Cycle (SDLC). It involves gathering, analysing, validating, and documenting the needs of stakeholders to ensure that the developed software aligns with their expectations and business goals. The process sets the foundation for all subsequent stages of development, making it essential for the project’s success.

**Activities in Requirement Analysis**

* **Requirements gathering**: Collect requirements from stakeholders, including users, business executives, and developers.
* **Requirements Elicitation**: Turning vague business ideas into structured, detailed, and information.
* **Requirements Documentation**: Translate gathered and elicited requirements into a formal document for clarity and reference.
* **Requirements Validation**
* **Requirements Prioritization**: Classify and prioritize requirements based on their importance to stakeholders, complexity, and technical feasibility.

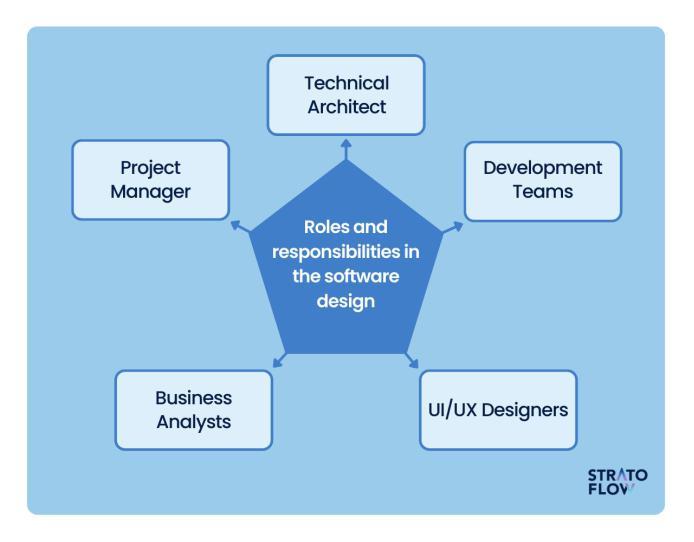
**Importance of requirement analysis**

* Prevention of Scope Creep: Clearly defined and documented requirements prevent deviations from original project scope, reducing delays and cost overruns.
* Improved Communication: Eﬀective communication between business stakeholders and developers is fostered via well-structured documentation.
* Minimized rework: Clear understanding and validation of requirements early on reduce the risk of major corrections later.
* Foundation for Design and Development: Serves as the blueprint for software architects, developers, and testers, ensuring alignment with business goals.

**DESIGN**

The design phase of the Software Development Life Cycle (SDLC) is when the development team creates a framework for the software. It is like a product roadmap; technical spec documents are a roadmap of the entire dev process of a software.

The design phase includes a detailed analysis of new software according to the requirement phase. This is the high priority phase in the development life cycle of a system because the logical designing of the system is converted into physical designing. The output of the requirement phase is a collection of things that are required, and the design phase gives the way to accomplish these requirements. The decision of all required essential tools such as **programming languages** like Java, .NET, PHP, a **database** like Oracle, MySQL, a combination of hardware and software to provide a platform on which software can run without any problem is taken in this phase.



**What happens during the Design Phase**

* The development team creates the software's structure, navigation, user interfaces, and database design.
* The team determines how data will be stored and flow.
* The team creates documents like the architecture document, maintenance manual, and user manual.
* The team selects the programming language, model, and framework.
* The team creates designs like flow charts, sketches, prototypes, and UML schemas.
* The team assesses the functional feasibility of the designs.

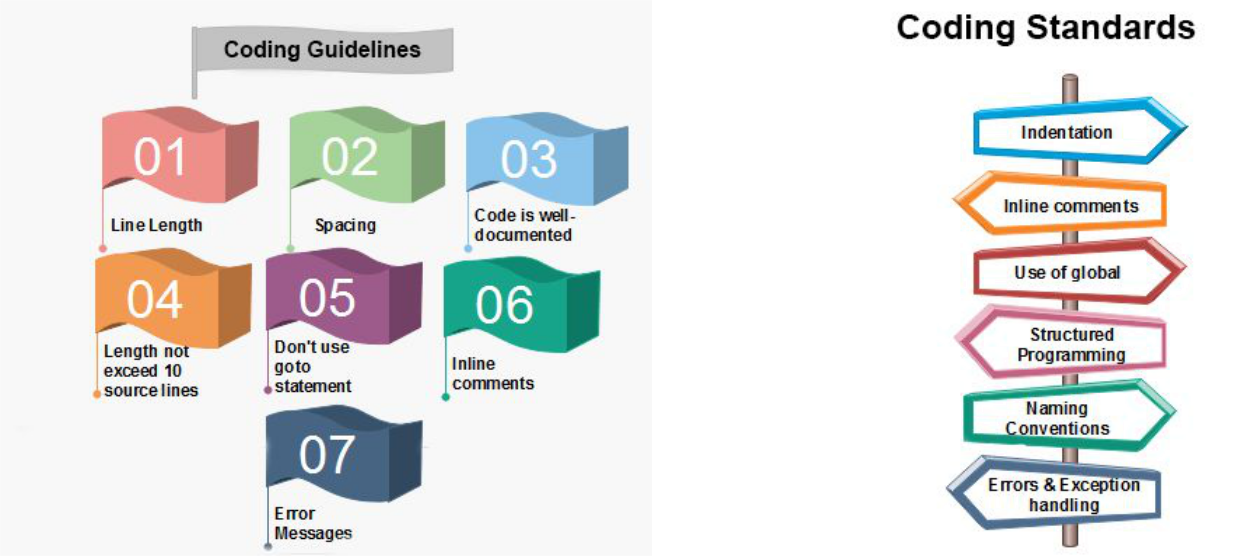
An organized document repository is developed for any critical project information, so the **Development Team** members can access, store, and refer to project documents and other deliverables from all life cycle phases. Periodic deliverable reviews are implemented to correct inaccuracy, ambiguities, and incompleteness.

**The database team** assists with architecture design and data conversion strategy. **The Technical Architect** plays the most crucial role and aims to describe the required functions and operations like screen layout, business rules, system database layout in detail and provides the architectural plan down to physical level.

**CODING PHASE**

The coding phase in the Software Development Life Cycle (SDLC) involves converting the system design into functional code using a programming language. This stage is the backbone of software development, as it translates theoretical designs into a working product. The quality of coding significantly influences the efficiency, performance, and maintainability of the software.

Proper coding ensures seamless integration with subsequent stages, particularly testing and maintenance.

Unlike the design phase, coding is typically performed by developers or programmers, who may not be directly involved in the design process. Their primary objective is to implement the design accurately while adhering to best practices and standards. The focus is not only on the coding itself but also on minimizing the cost of debugging, testing, and long-term maintenance.

**Objectives of the Coding Phase**

* **Turn Design into Functionality:** The primary aim is to transform a well- documented system design into a fully functional program.
* **Minimize Bugs Early:** Writing clean, efficient code reduces errors and makes testing simpler. Early detection and elimination of defects reduce long-term costs.
* **Optimize Performance:** Well-written code ensures efficient execution and enhances the system’s responsiveness and scalability.
* **Ease of Understanding:** The code should be understandable by other developers. This not only helps in debugging but also ensures ease of future modifications.
* **Facilitate Testing:** By adhering to coding guidelines and structuring programs modularly, developers make testing and debugging less complex and more systematic.

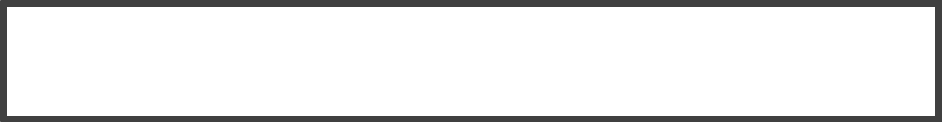
**Programming Language Selection**

Choosing the right programming language is critical in the coding phase. Factors influencing the choice include:

* **Application Requirements:** The language should align with the technical requirements of the project. For example, Python is suitable for AI/ML applications, while C++ is preferred for system- level programming.
* **Developer Expertise:** The team’s familiarity with the language can significantly affect productivity and quality.
* **Performance Needs:** High-performance applications may require low-level languages like C, while general-purpose applications can use higher-level languages like Java or Python.
* **Community Support:** A language with a large community and extensive documentation simplifies troubleshooting and accelerates development.

**Common Challenges in Coding**

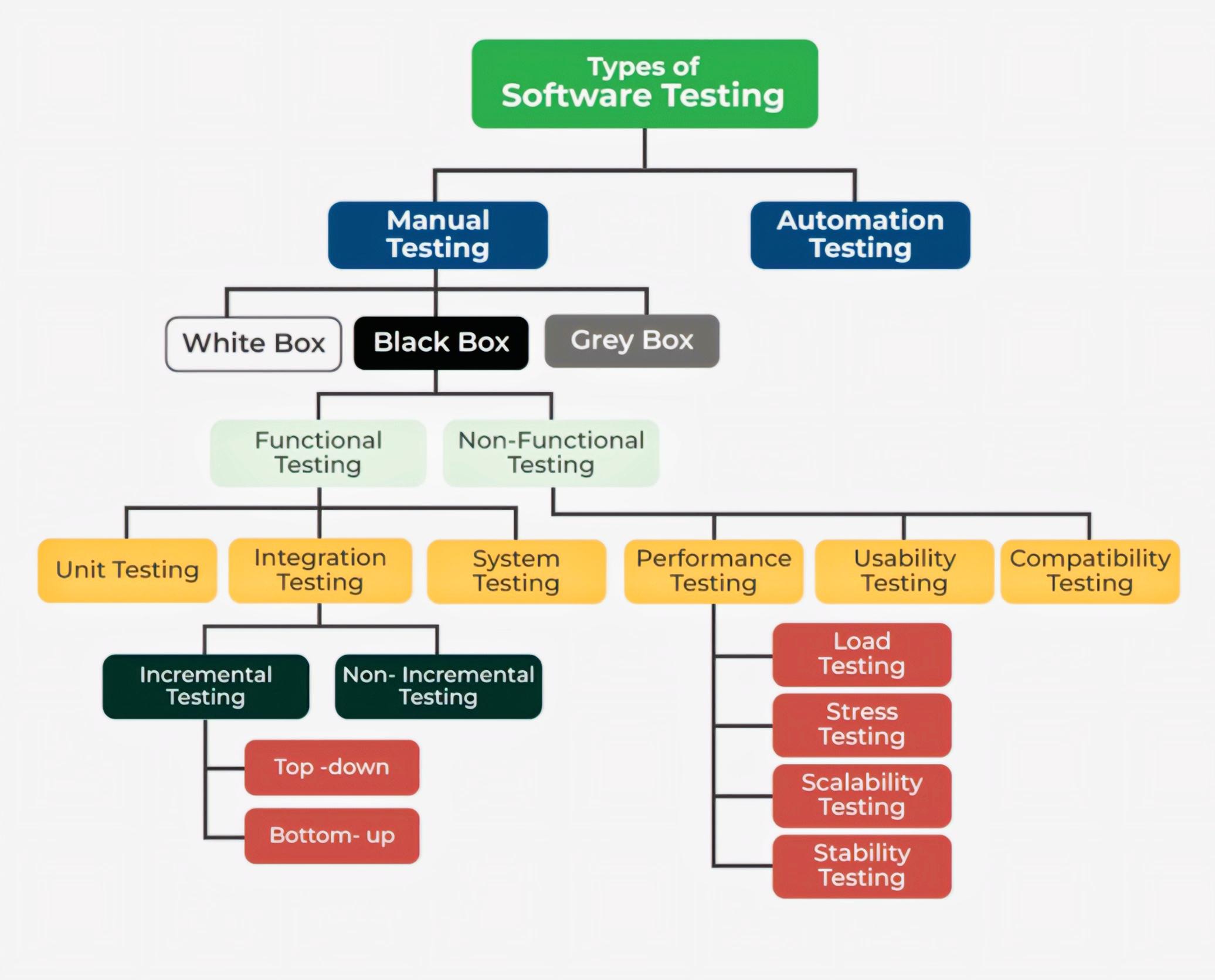
* **Unclear Requirements**: Ambiguity in requirements may lead to incorrect implementation. Frequent communication with stakeholdersis essential to avoid this.
* **Integration Issues**: Integrating independently developed modules can result in compatibility problems. Proper interface design and early integration testing can mitigate this.
* **Time Constraints:** Rushed coding often results in errors and poor-quality code. Following Agile methodologies can help manage time effectively.
* **Complex Algorithms:** Some problems require complex algorithms, which can be challenging to implement. Leveraging existing libraries or consulting domain experts can help.

**TESTING**

#### Overview of Software Testing

Software testing is a crucial process in the software development lifecycle. It ensures that the software meets the specified requirements, functions correctly, and is free of defects.

Testing of the software is necessary to ensure its smooth execution.**Types of Software Testing**



Testing is categorized into two broad approaches:

1. **Manual testing**

Manual software testing techniques are broadly classified into three categories:

1. White box testing
2. Black box testing
3. Grey box testing
4. **Automation testing**.
5. **Manual testing**: It includes testing software manually, i.e., without using any automation tool or script.
6. **Automation testing**: It is also known as Test Automation, is when the tester writes scripts and uses another software to test the product.

#### Testing Techniques

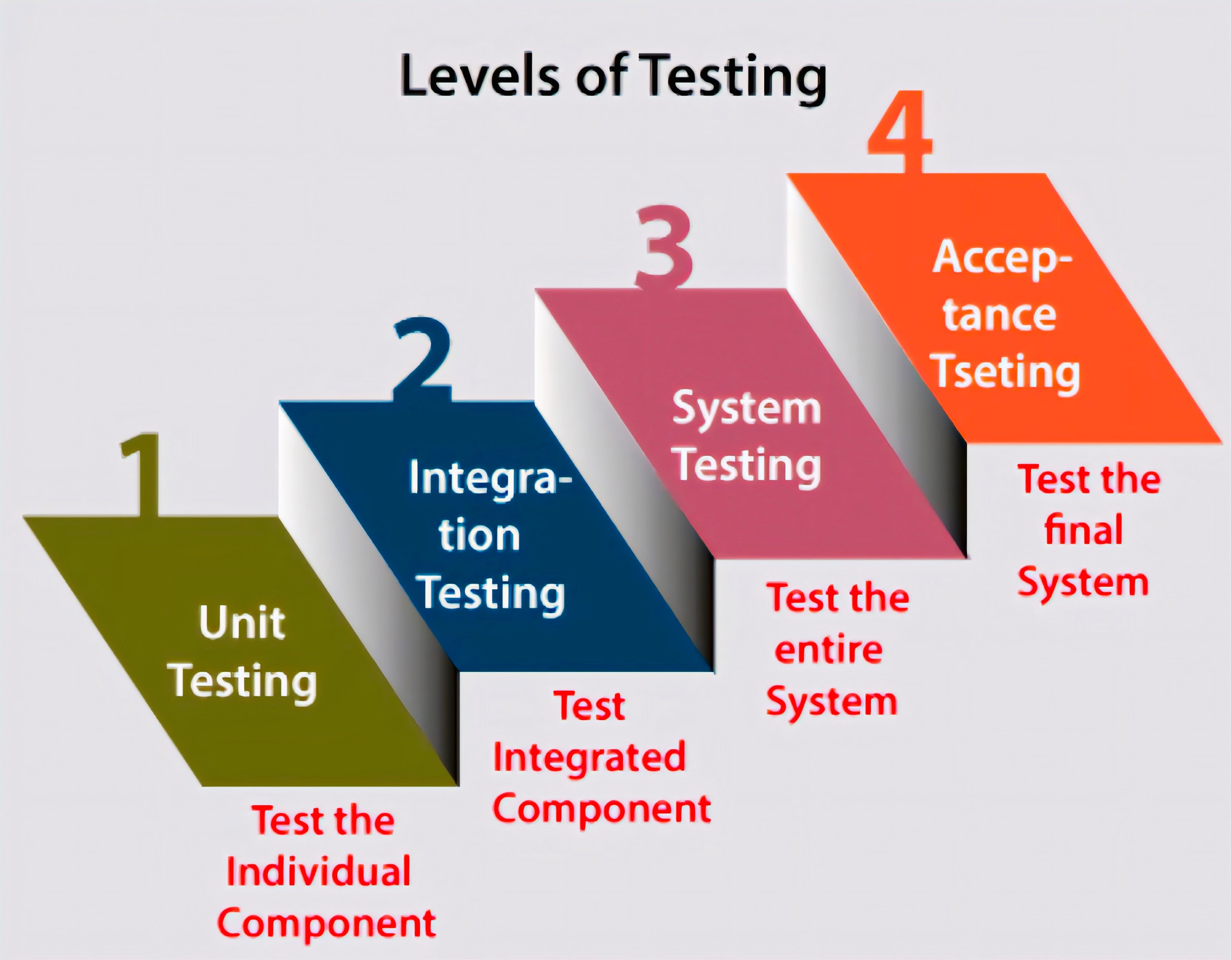
Software testing techniques are broadly classified into three categories:

1. **Black Box Testing**:
   * Tests the software’s functionality without access to the internal code.
   * Ideal for verifying inputs, outputs, and user interface behavior.
2. **White Box Testing**:
   * Involves testing internal code and logic.
   * Used to ensure complete code coverage and adherence to specifications.
3. **Grey Box Testing**:
   * Combines elements of black-box and white-box testing.
   * Testers have partial knowledge of the internal workings.

#### 1. Functional Testing

Functional testing is done in 4-5 phases (depending on the testing strategy and project requirements) as it generally involves a series of sequential testing levels to ensure the software functions behave as expected.

These phases typically include:



1. **Unit Testing**:  
   * Tests individual components or modules in isolation.
   * Performed by developers during the development phase using the white-box approach.
   * **Advantages**:
     + Early detection of defects.
     + Improves code quality and reusability.
2. **Integration Testing**:  
   * Verifies interactions between modules.
   * Checks data flow and communication between components.
   * **Methods**:
     + **Top-Down Approach**: Starts testing with high-level modules, integrating lower-level modules incrementally.
     + **Bottom-Up Approach**: Begins with testing lower-level modules before integrating higher-level ones.
3. **System Testing**:  
   * Tests the complete integrated system to ensure it meets the defined requirements.
   * Involves both functional and non-functional aspects.
   * Conducted after all components are integrated with hardware and software.
4. **Alpha Testing / User acceptance testing**:  
   * Performed internally near the end of development.
   * Conducted by developers and QA(Quality Assurance) teams to identify and fix bugs.
   * **Advantages**:
     + Enhances quality by detecting early-stage defects.
     + Improves user experience through internal feedback.
5. **Beta Testing**:
   * Beta testing is the process of testing a software product or service in a real-world environment before its official release.
   * Conducted in real-world conditions by a selected group of users.
   * It is an essential step in the software development lifecycle as it helps identify bugs and errors that may have been missed during the development process.
   * Provides valuable feedback on bugs, usability issues, and performance improvements.

#### 

#### 2. Non-Functional Testing

This type of testing evaluates aspects like performance, scalability, and security that are not directly related to functional correctness. Key types include:

1. **Performance Testing**:
   * Assesses the speed and responsiveness of the software under various loads.
2. **Stress Testing**:
   * Determines the software's stability under extreme workloads or unfavorable conditions.
3. **Recovery Testing**:
   * Verifies the system’s ability to recover from crashes, failures, or network interruptions.
   * **Example**: Testing a browser’s ability to restore sessions after a power outage.
   * **Advantages**:
     + Ensures reliability and stability.
     + Reduces risks of post-release failures.
   * **Disadvantages**:
     + Time-consuming and costly.
     + Requires skilled testers.
4. **Security Testing**:
   * Focuses on identifying vulnerabilities and ensuring data protection.
   * **Principles**:
     + Confidentiality, Integrity, Authentication, Authorization, Availability, and Non-repudiation.
   * **Advantages**:
     + Enhances system security and prevents data breaches.
     + Reduces risks of security incidents in production environments.

#### 3. Maintenance Testing

Maintenance testing ensures that changes or updates in the software do not introduce new bugs. **Regression Testing** is a key part of this process, verifying that modifications do not adversely affect previously working features.

#### Software Testing : Why is it important ?

Testing is indispensable for ensuring the success and reliability of software products. Below are the key reasons:

* **Early Bug Detection**: Detecting and fixing bugs early in the development process reduces overall costs and prevents complications in later stages.
* **Enhanced Quality**: Testing improves software stability, functionality, and user satisfaction by ensuring the product meets its requirements.
* **Risk Mitigation**: It minimizes the chances of critical failures or defects after deployment, reducing risks in production environments.
* **Feedback and Usability**: Alpha and beta testing provide valuable insights into real-world performance, usability, and user needs, guiding improvements before final release.

By systematically employing various testing methods, developers ensure the software meets both functional and non-functional requirements, delivering a reliable, secure, and seamless user experience.

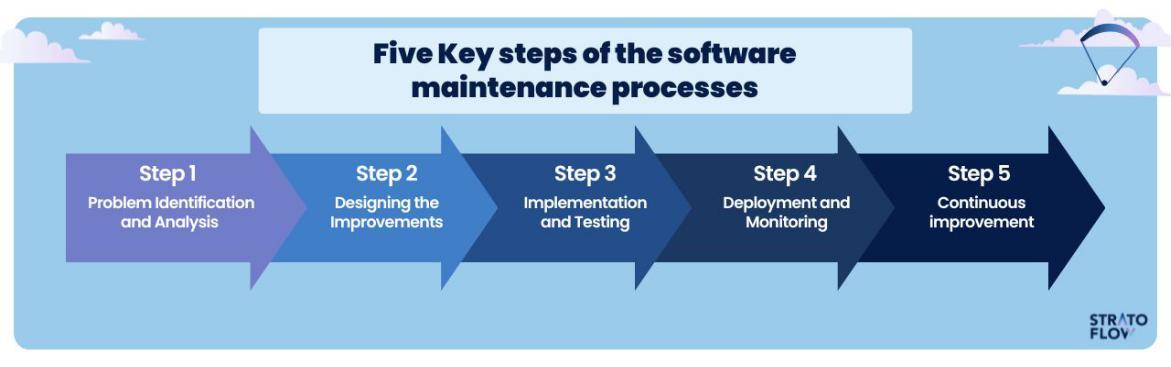
**MAINTENANCE**

Maintenance is a crucial phase of the Software Development Lifecycle (SDLC) that begins after the deployment of a software product. It involves updating, optimizing, and improving the software to ensure its functionality, security, and relevance over time. Maintenance accounts for a significant portion of the software's overall lifecycle cost.

**There are four main types of maintenance:**

1. **Corrective Maintenance:** Fixing bugs or errors that were not detected during the testing phase. For example, resolving a crash issue in a mobile app caused by an edge-case user input.
2. **Adaptive Maintenance:** Modifying software to keep it compatible with changing environments, such as operating system updates or hardware upgrades. For instance, updating a desktop application to work with a newer version of Windows.
3. **Perfective Maintenance:** Enhancing existing features or adding new ones based on user feedback to improve usability and performance. An example would be adding a dark mode option to a web application.
4. **Preventive Maintenance:** Anticipating potential issues and making changes to reduce risks, such as refactoring code to improve readability and maintainability.

Effective maintenance ensures software remains useful, secure, and efficient. For example, modernizing legacy systems like COBOL-based banking software is crucial for integrating with newer digital payment platforms. This phase is vital for long-term user satisfaction and organizational success.



**Resources:**

1. <https://www.javatpoint.com/software-engineering-coding>
2. <https://www.browserstack.com/guide/coding-standards-best-practices>
3. [https://www.geeksforgeeks.org/software-development life-cycle-sdlc/](https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/)
4. <https://stratoflow.com/sdlc-design-phase/>

**Thank You**