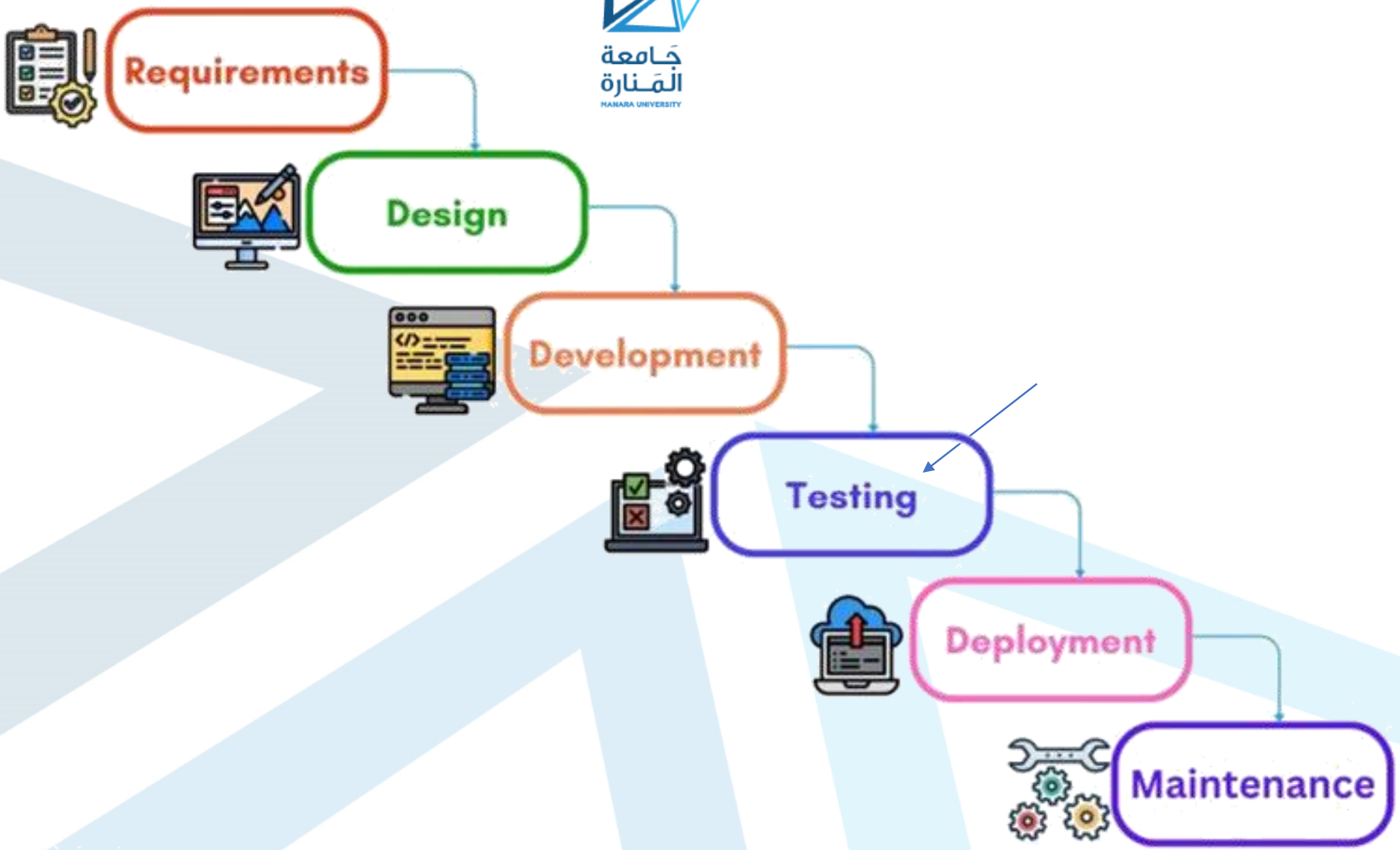
# Software Engineering -2-

## Lecture -5-

Dr. Inas Laila

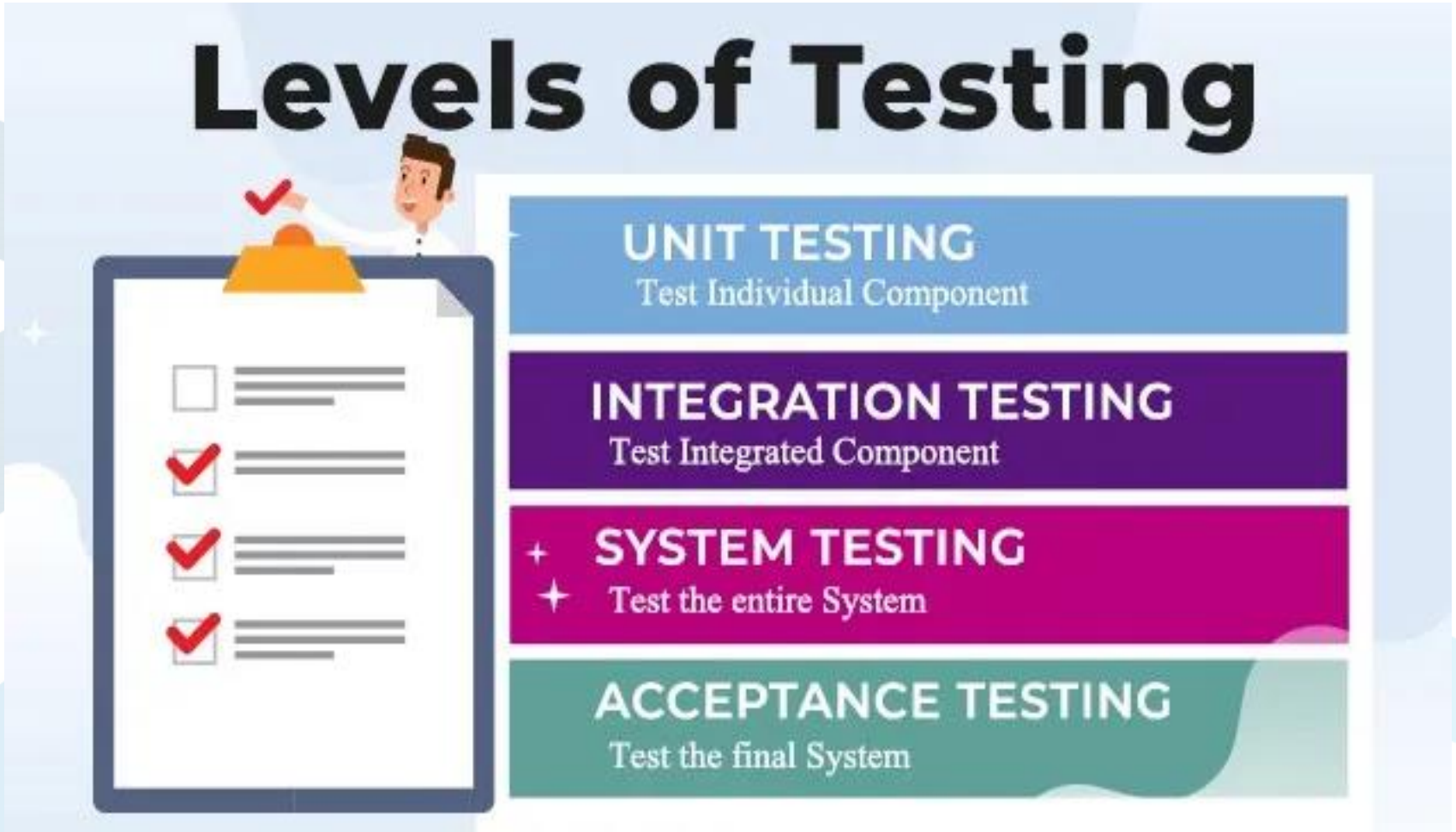

Requirements
Design
Development
Testing
Deployment
Maintenance

# Testing Within the Software Lifecycle

1. Test Levels
2. Test Types
3. Test Technique
4. Regression Testing

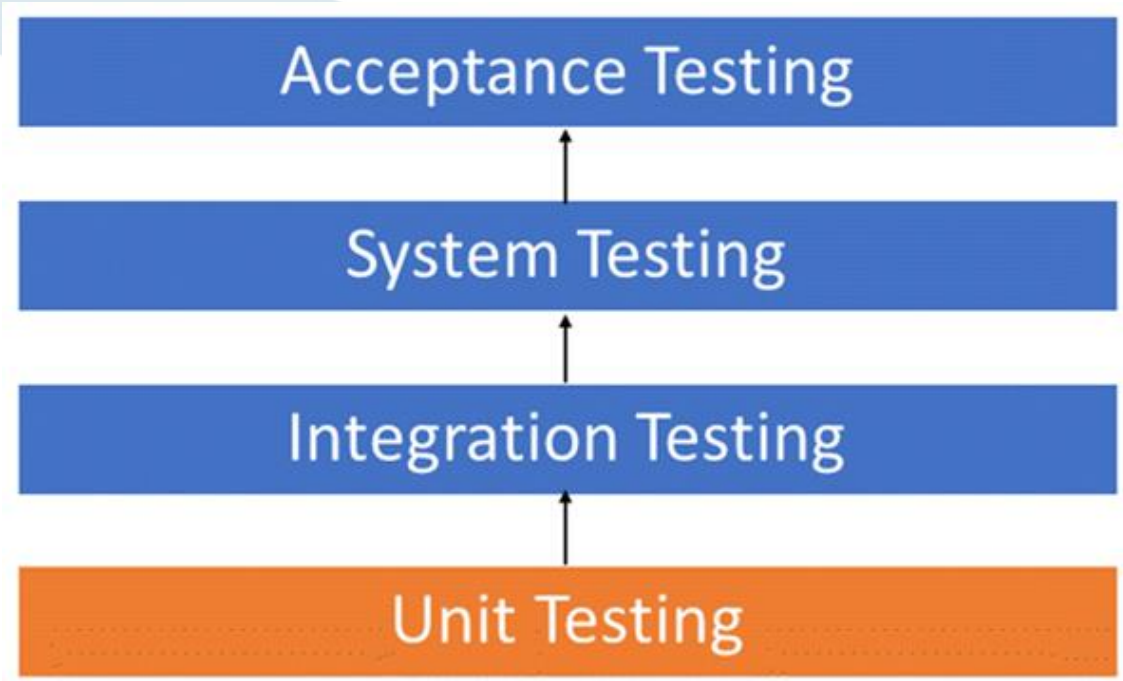
# Test Levels

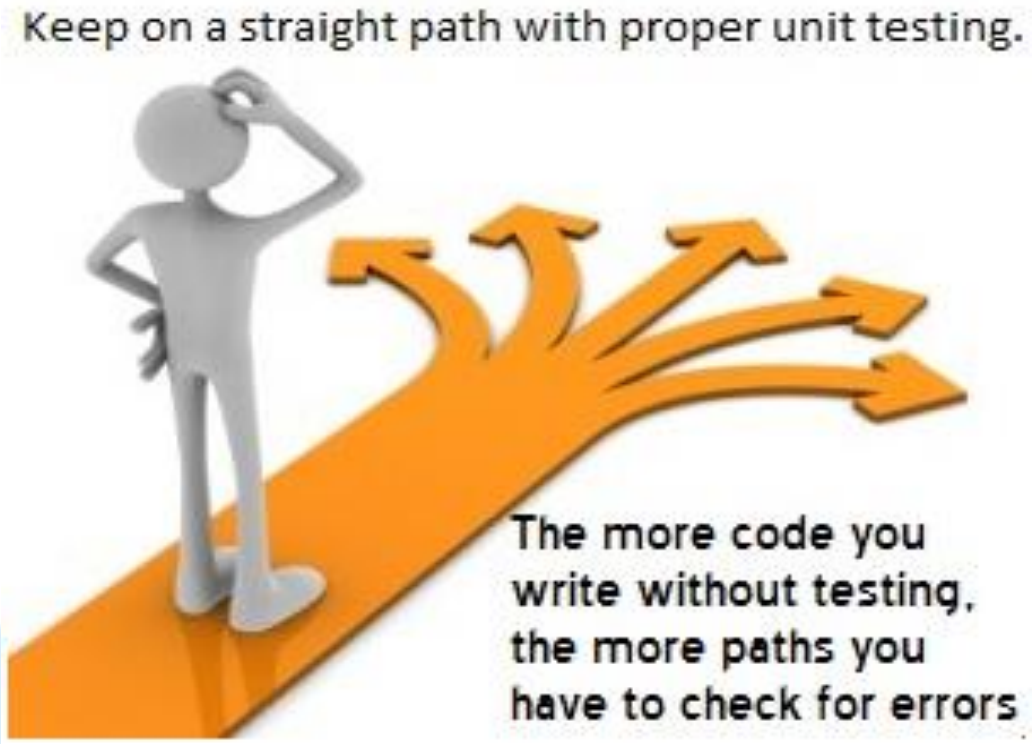
there are four levels of testing:
unit testing, integration testing, system testing, and acceptance testing.



# 1- Unit Testing ( Component Test ):

- Unit testing of software applications is done during the coding of an application, Unit Testing is typically performed by the developer.
- A Unit is a smallest testable portion of system or application which can be compiled, and executed. This kind of testing helps to test each module separately. The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not.
- The goal of unit testing is to isolate (عزل) each part of the program and show that the individual parts are correct.


Acceptance Testing
System Testing
Integration Testing
Unit Testing


Keep on a straight path with proper unit testing.
The more code you write without testing, the more paths you have to check for errors

# How to do Unit Testing

Unit testing can be performed either **manually or through automation**, with automated testing being the more common and efficient approach in modern development practices.

under the automated approach: 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.

**Unit Testing Tools:** There are several automated tools available to assist with unit testing. examples :

| | |
|---|---|
| Manual testing is time-consuming. | Automation testing is faster than manual testing. |

Junit: Junit is a free to use testing tool used for Java programming language.

NUnit: NUnit is widely used unit-testing framework use for all .net languages

PHPUnit: PHPUnit is a unit testing tool for PHP programme

# Integration Testing (units Integration)

The goal of Integration testing is to integrate some dependent units to check correctness of communications and data formats among them . Integration Testing is a Software Testing Technique that focuses on verifying the interactions and data exchange between different components or modules of a Software Application.

The goal of Integration Testing is to identify any problems or bugs that arise when different components are combined and interact with each other. Integration testing is done by the testing team.

Integration Test Case differs from other test cases in the sense it focuses mainly on the interfaces & flow of data/information between the modules.

Here priority is to be given for the integrating links rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page. Similarly Mail Box: Check its integration to the Delete Mails Module.

# Approaches, Strategies, Methodologies of Integration Testing

Software Engineering defines variety of **strategies to execute Integration testing:**

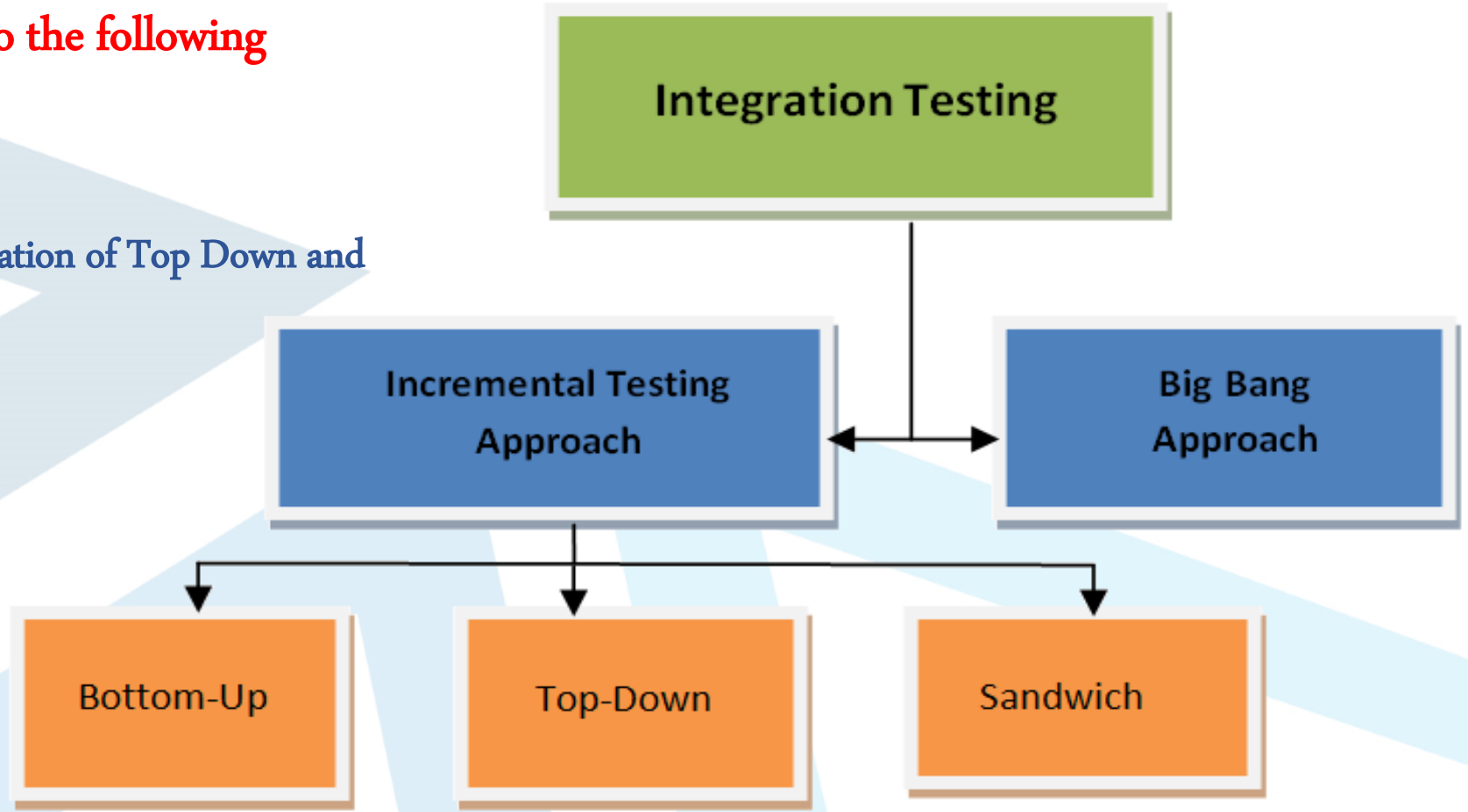
**Big Bang Approach:** In simple words, all the modules of the system are simply put together and tested. It Convenient for small systems. No stubs/drivers – Reduces extra development effort.

Disadvantages: **Fault Localization is difficult, Late defect detection : Bugs found only after full integration.**

**Incremental Approach:** In this approach, testing is done by joining two or more modules that are logically related. Then the other related modules are added and tested for the proper functioning. The process continues until all of the modules are joined and tested successfully. Earlier bug detection because modules are integrated and tested as soon as they are developed.

## Incremental Approach divided into the following

- Top Down Approach
- Bottom Up Approach
- Sandwich Approach : Combination of Top Down and Bottom Up



## What are Stubs and Drivers in Integration Testing?

Stubs and drivers are essential dummy programs that enable integration testing when not all modules are available simultaneously. These test simulate missing components, allowing testing to proceed without waiting for complete system development.

Stubs are dummy modules that replace lower-level components not yet developed or integrated. Simulate lower-level module behavior and Used in top-down integration testing

Drivers are dummy programs that call the module being tested, simulating higher-level components. They pass test data to lower-level modules and collect results. It Used in bottom-up integration testing

For instance, when testing a database module, a driver simulates the business logic layer, sending queries.

| Component | Use Stub | Use Driver |
|---|---|---|
| Testing Approach | Top-down testing | Bottom-up testing |
| Replaces | Lower-level modules | Higher-level modules |
| Function | Returns dummy data | Sends test data |

## Bottom-up Integration

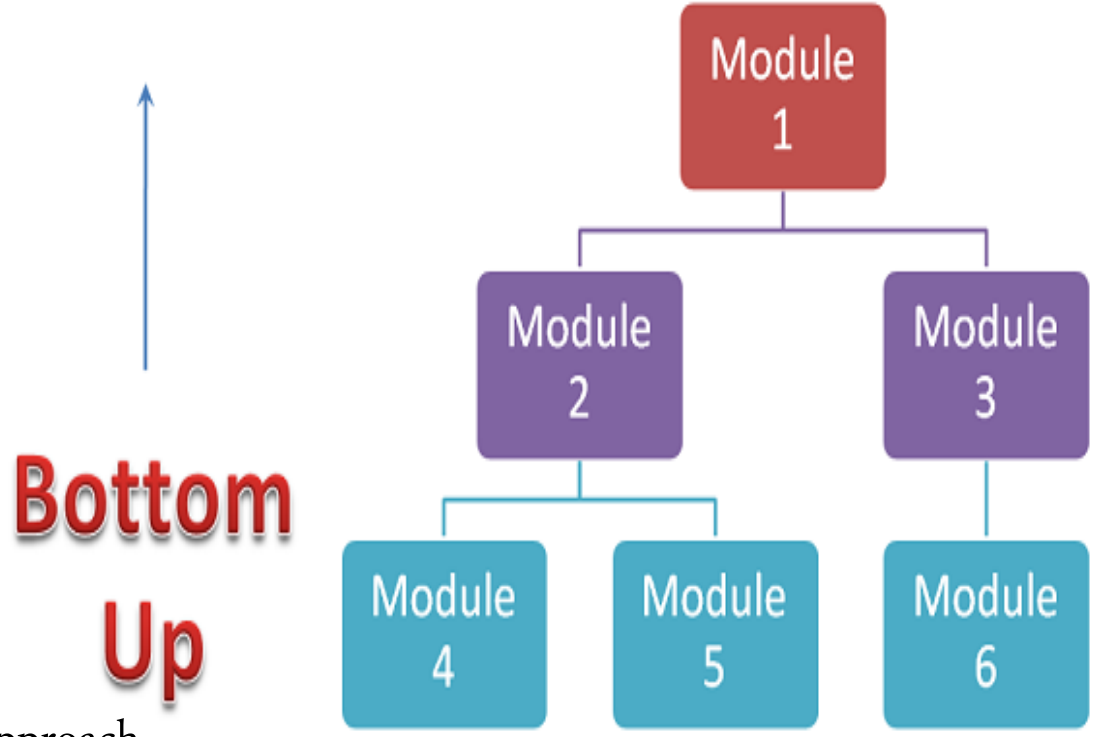
Best for applications that uses bottom up design approach.

In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested.

Unavailable Components or systems are substituted by Drivers.

The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem.

This integration testing **uses test drivers** to drive and pass appropriate data to the lower-level modules. In bottom-up testing, no stubs are required




**Advantages**: Fault localization is easier than Big Bang Approach.

No time is wasted waiting for all modules to be developed unlike Big-bang approach

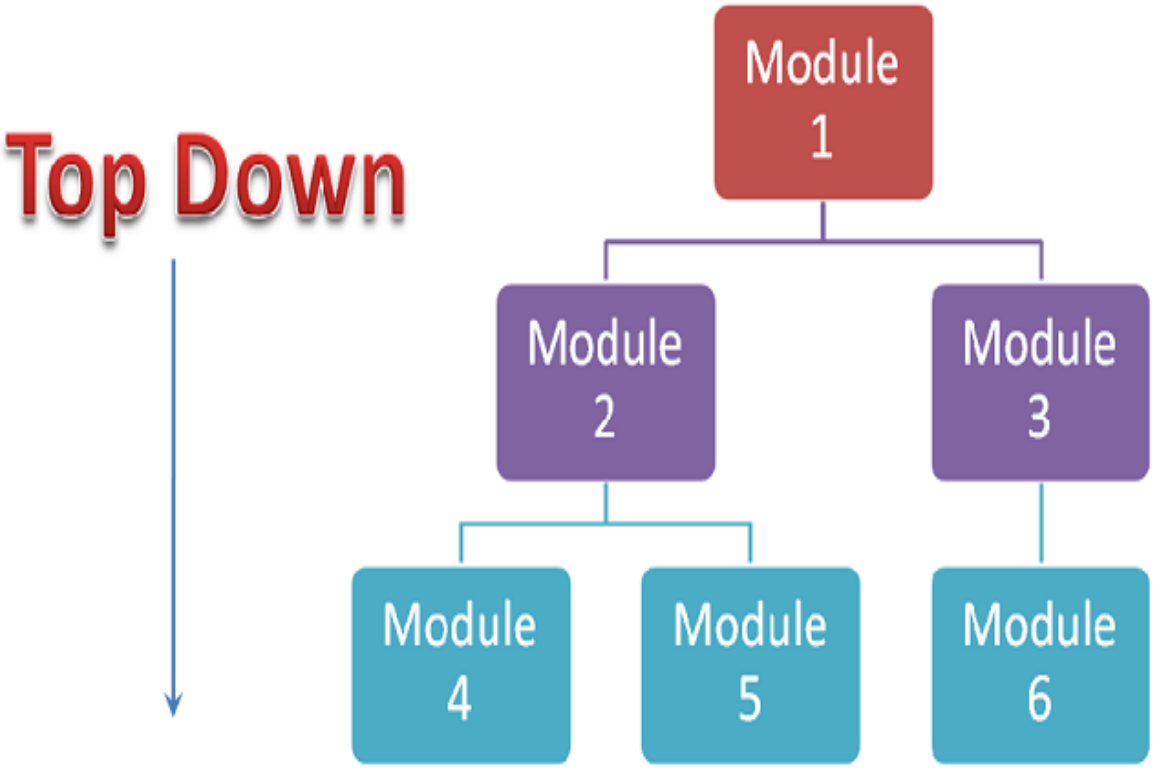
**disadvantages**

Driver modules must be produced.

it can be time-consuming since drivers have to be developed for performing these tests

## Top-down Integration:

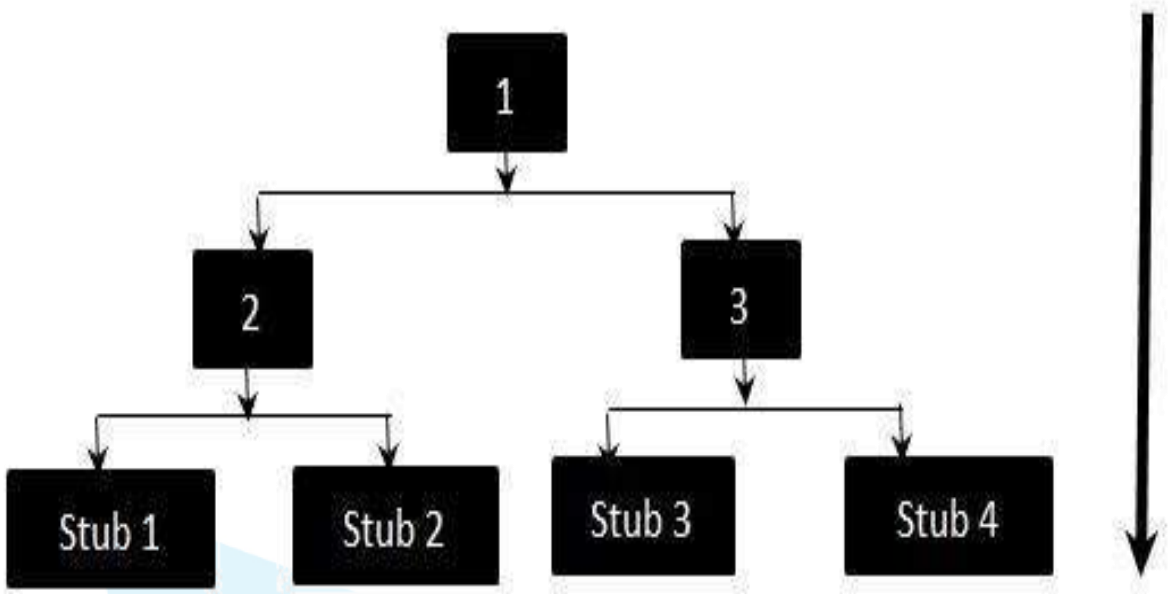
Top-down testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through control flow of architecture structure. **Unavailable Components or systems are substituted by stubs.**




Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated ➔ Needs many Stubs

## Top-Down Integration Testing

Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.




The above diagrams clearly states that Modules 1, 2 and 3 are available for integration, whereas, below modules are still under development that cannot be integrated at this point of time. Hence, Stubs are used to test the modules. The order of Integration will be:

```
+ Firstly, the integration between the modules 1,2 and 3
+ Test the integration between the module 2 and stub 1,stub 2
+ Test the integration between the module 3 and stub 3,stub 4
```

```
1,2
1,3
2,Stub 1
2,Stub 2
3,Stub 3
3,Stub 4
```

| | Unit Testing | Integration Testing |
|---|---|---|
| | In unit testing, each module of the software is tested separately. | In integration testing, all modules of the software are tested combined. |
| | Unit testing is performed by the developer. | Integration testing is performed by the tester. |
| | Unit testing is performed first of all testing processes. | Integration testing is performed after unit testing and before system testing. |
| | Unit testing is white box testing. | Integration testing is black box testing. |

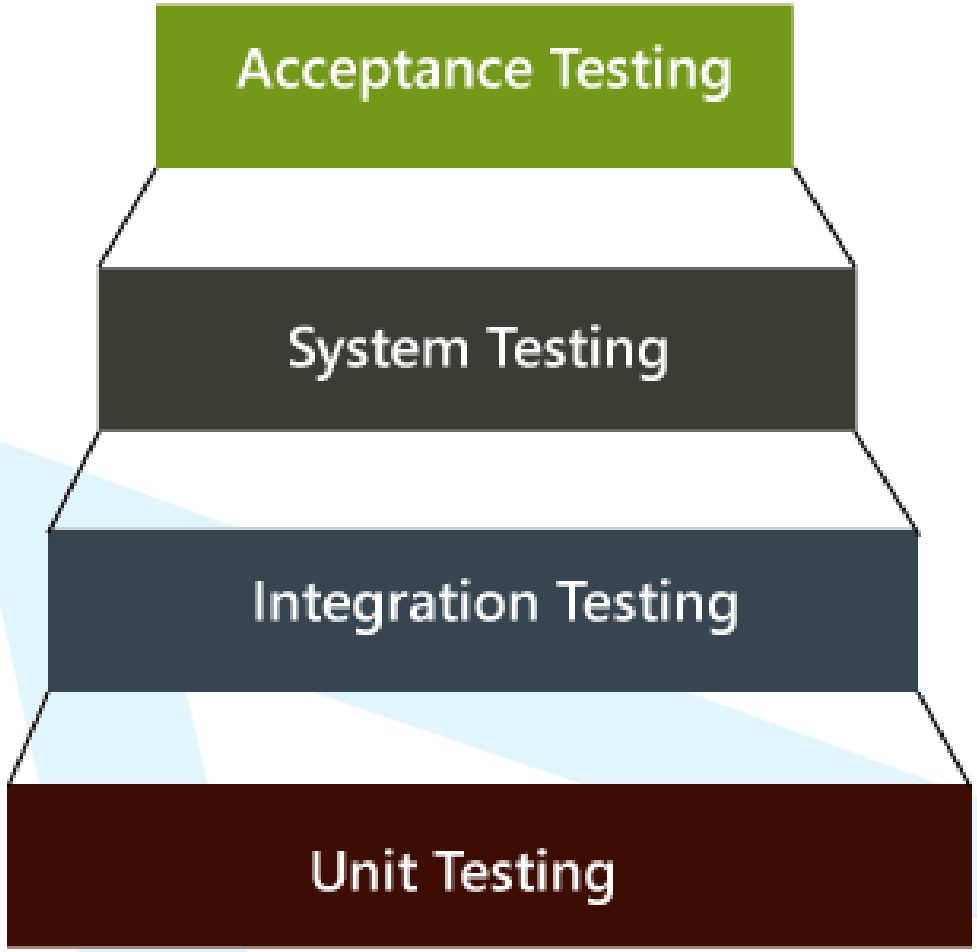
## System Integration Test

- Incase of system communicates with other systems ( external ) , then this level of testing checks correct communications , messages , data , dependencies are correct between systems.
- Usually Done by independent testers to check that customer scenarios ( business ) is fulfilled as requested ( needed ).

# System Test

Once Integration testing is completed successfully, we move on to the next level of testing, which is called **System testing**. IEEE defines the system tests as "**A testing performed on a complete and integrated system to check the system's compliance with its specific requirements**".

•***System testing*** is done by the testing team, and the approach is to focus on overall functional and non-functional behavior of the system. Typically it's done in a dedicated environment that mimics the production environment in terms of hardware and software.



## System testing can be classified as :

•**Functional System testing**: This ensures that the system is performing all the functions that were specified in the software requirement specification. E.g. On an Amazon website, a user should be able to search a product, add to cart, and finally place an order using a debit/credit card.

•**Non-Functional System testing**: This ensures that the non-functional aspects of the system are working to acceptable standards. E.g. On an amazon website, the search results should be shown to the user within 3 seconds. The  page should not crash on a simultaneous user load of 50k users. Some of the types of Non-functional requirements are :

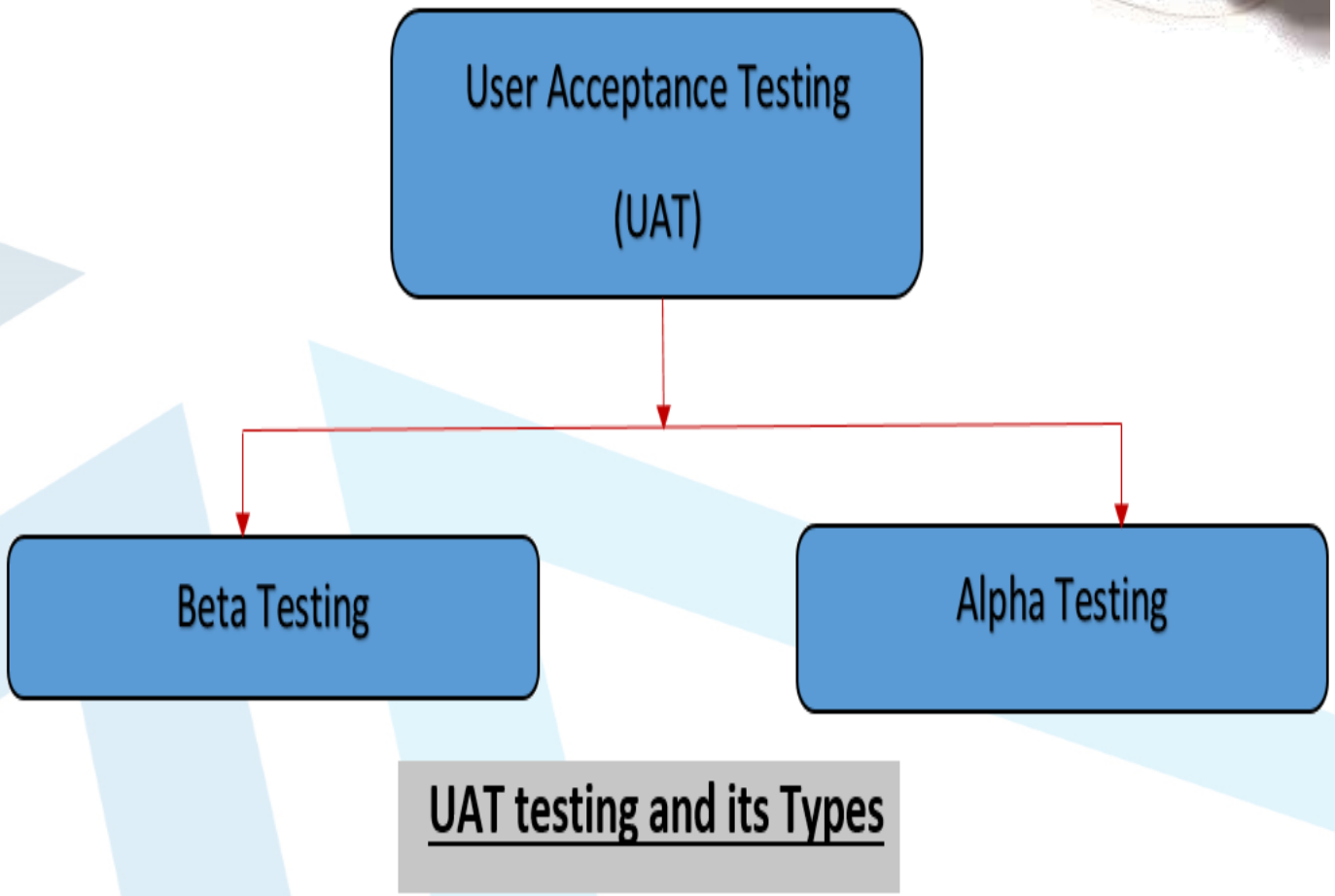
- **Performance**
- **Load Handling** - Performance of system under increased load
- **Recovery** - How the system recovers on failures. E.g. What happens when Payment gateway goes down.
- **Reliability** - Ability of software to repeatedly perform its function consistently with time.
- **Usability** - Ease with which a user can perform desired functions

# User Acceptance Testing


While all the other levels (Component, Integration, and System) are executed by the software producers, acceptance testing is the only level that is carried out by the customers and end-users.


## User Acceptance Testing (UAT)
## (اختبار قبول المستخدم)


User Acceptance Testing (UAT) is **the final phase of the software development lifecycle**, where real end-users or business representatives test the software to verify that it meets business requirements and functions as intended in real-world scenarios.



# Levels of Testing


**Alpha testing** is a type of acceptance testing; Alpha testing is an initial phase of testing a product or software, conducted by the development or QA team to identify and fix bugs before moving on to the beta testing phase Alpha testing is carried out in a lab environment and usually the testers are internal employees of the organization. when user are using the software then developer looks they activities and if user has some Queries then developer explain them. After the testing user gives feedback about the software called alpha only because it is done early on, near the end of the development of the software, and before beta testing.

Alpha testing is the initial testing phase to ensure the product is functional and ready for the next stage of testing with a wider audience.

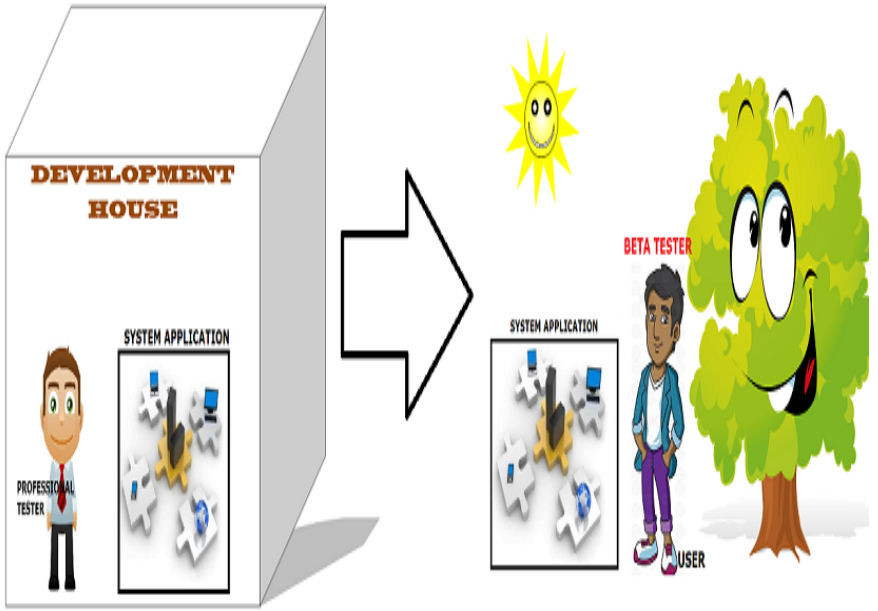
This process is crucial for enhancing software quality and ensuring a positive user experience before releasing the product to external testers or customers.

# Beta Testing

Beta Testing is performed by real users of the software application in a real environment. Beta testing can be called pre-release testing and it is one type of User Acceptance Testing. The main purpose of beta testing is to verify software compatibility with different software and hardware configurations, types of network connection, and to get the users' feedback on software usability and functionality. And get input from actual users in order to find any bugs, usability difficulties, or areas that need to be improved before the product is formally released. There are two types of beta testing:

**open beta** is available for a large group of end-users or to everyone interested

**closed beta** is available only to a limited number of users that are selected especially for beta testing. Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product.



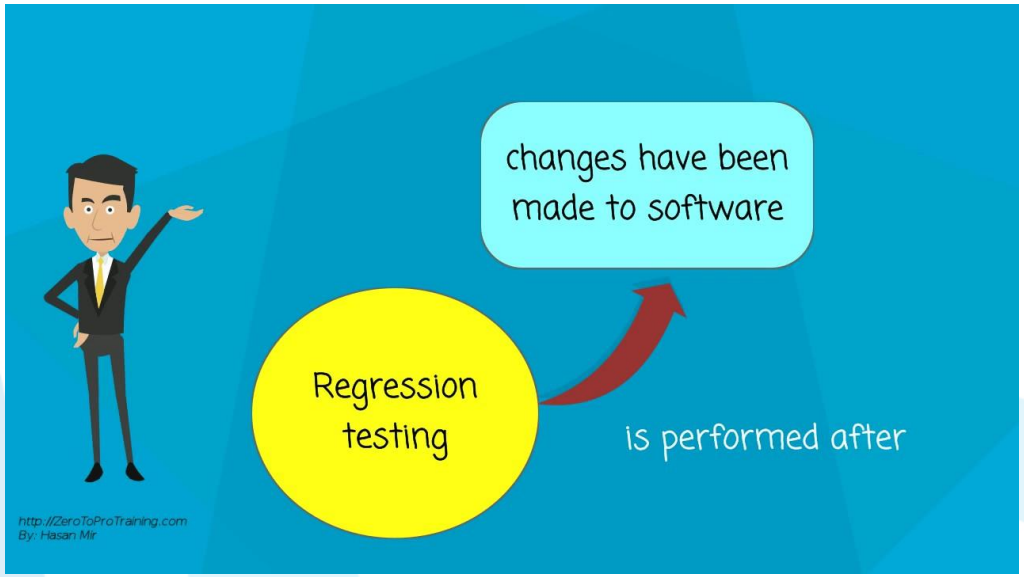

BETA TESTING OF THE PRODUCT IN REAL WORLD ENVIRONMENT

| Alpha Testing | Beta Testing |
| --- | --- |
| Alpha testing performed by Testers who are usually internal employees of the organization | Beta testing is performed by Clients or End Users who are not employees of the organization |
| Alpha Testing performed at developer's site | Beta testing is performed at a client location or end user of the product |
| Reliability and Security Testing are not performed in-depth Alpha Testing | Reliability, Security, Robustness are checked during Beta Testing |
| Alpha testing involves both the white box and black box techniques | Beta Testing typically uses Black Box Testing |
| Alpha testing requires a lab environment or testing environment | Beta testing doesn't require any lab environment or testing environment. The software is made available to the public and is said to be real time environment |
| Long execution cycle may be required for Alpha testing | Only a few weeks of execution are required for Beta testing |
| Critical issues or fixes can be addressed by developers immediately in Alpha testing | Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product |
| Alpha testing is to ensure the quality of the product before moving to Beta testing | Beta testing also concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real time users. |

# Regression Test

*Regression testing* is re-running functional and non-functional tests (retesting the program)to ensure that previously developed and tested software still performs after a change (e.g. bug fixes or new functionality)have been made.

It is performed by the testing teams.

# Defect Management

- Definition
- Factors of good Defect Report
- Defect Life cycle

## Definition

Defect is a deviation (انحراف) from Requirement, Design, Standard.

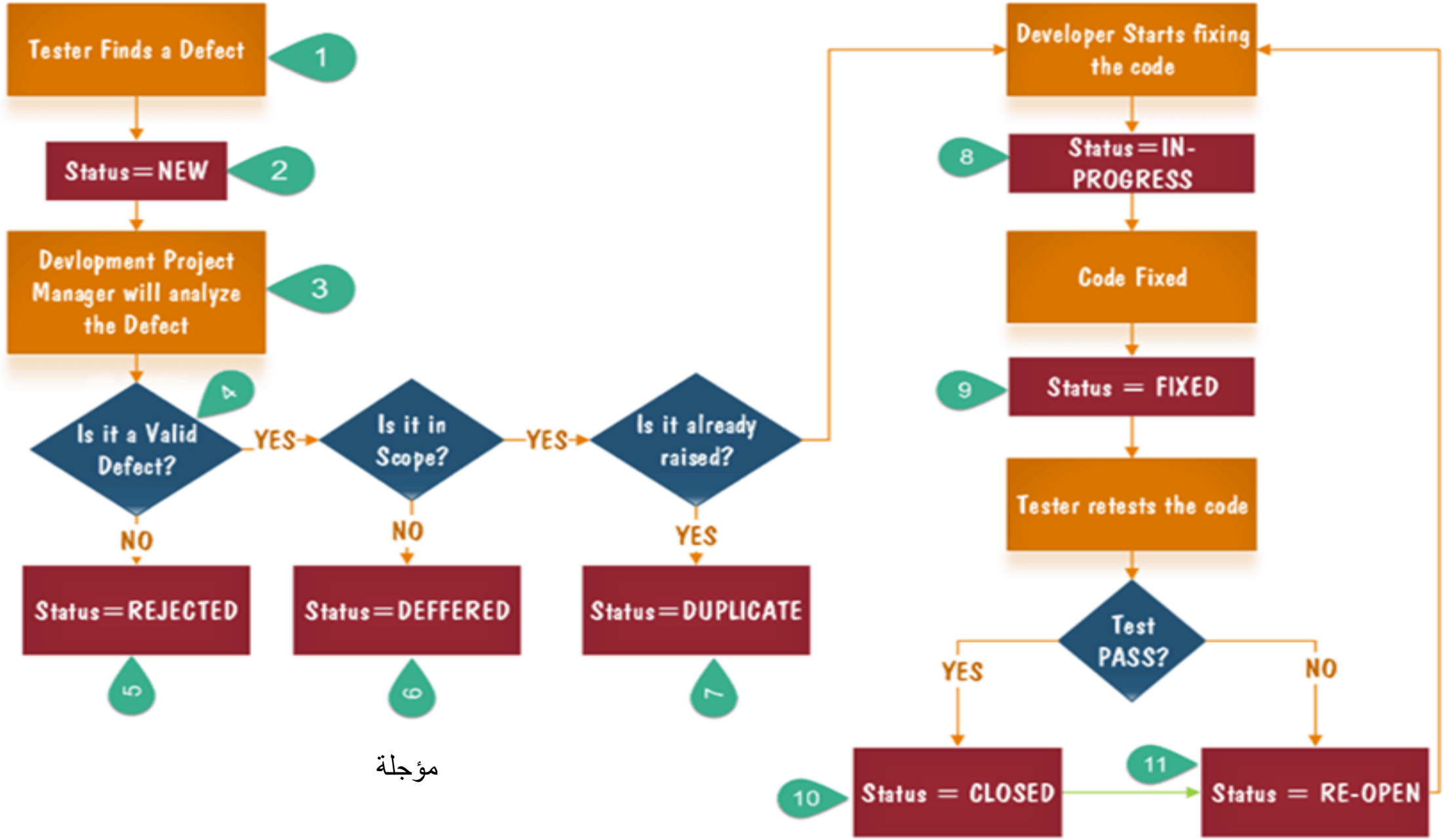
# Factors of Good Defect Report

- Title
- ID
- Description
- Reproducible(قابلة للتكرار) Detailed Ordered Steps
- Expected Result
- Actual Result
- Severity (الخطورة)
- Priority
- Enough attachment
- Test case traceable
- Requirement ( Item) traceable
- Status
- Discovered by
- Assigned to
- Environment - Build

# Defect Life cycle

**Defect life cycle**, also known as **Bug Life cycle** is the journey of a **defect cycle**, which a **defect** goes through during its lifetime.

It varies from organization to organization and also from project to project


Tester Finds a Defect
1
Status = NEW
2
Devlopment Project Manager will analyze the Defect
3
Is it a Valid Defect?
4
YES
NO
Is it in Scope?
YES
NO
Is it already raised?
YES
Status = REJECTED
5
Status = DEFFERED
6
Status = DUPLICATE
7
Developer Starts fixing the code
8
Status = IN-PROGRESS
Code Fixed
9
Status = FIXED
Tester retests the code
Test PASS?
YES
NO
10
Status = CLOSED
11
Status = RE-OPEN

مؤجلة

•Tester finds the defect

•Status assigned to defect- New

•Defect is forwarded to Project Manager for analyze

•Project Manager decides whether defect is valid

•Here the defect is not valid- status given "Rejected."

•So, project manager assigns a status **rejected**. If the defect is not rejected then the next step is to check whether it is in scope. Suppose teste find a problem But it is not a part of the current release then such defects are assigned as a **postponed or deferred** status.

•Next, manager verifies whether a similar defect was raised earlier. If yes defect is assigned a status **duplicate**.

•If no the defect is assigned to the developer who starts fixing the code. During this stage, the defect is assigned a status **in- progress.**

•Once the code is fixed. Defect is assigned a status **fixed**

•Next the tester will re-test the code. In case, the **Test Case** passes the defect is **closed.** If the test cases fails again, the defect is **re-opened** and assigned to the developer.

•Consider a situation where during the 1st release of APP a defect was found that was fixed and assigned a status closed. During the second upgrade release the same defect again re-surfaced. In such cases, a closed defect will be **re-opened.**

That's all to Bug Life Cycle

# Test Planning and Estimation

## Exit Criteria:

Run out of budget?

Run out of time?

Boss says stop?

All defects have been fixed?


HAVE ANY
QUESTIONS AND DOUBTS