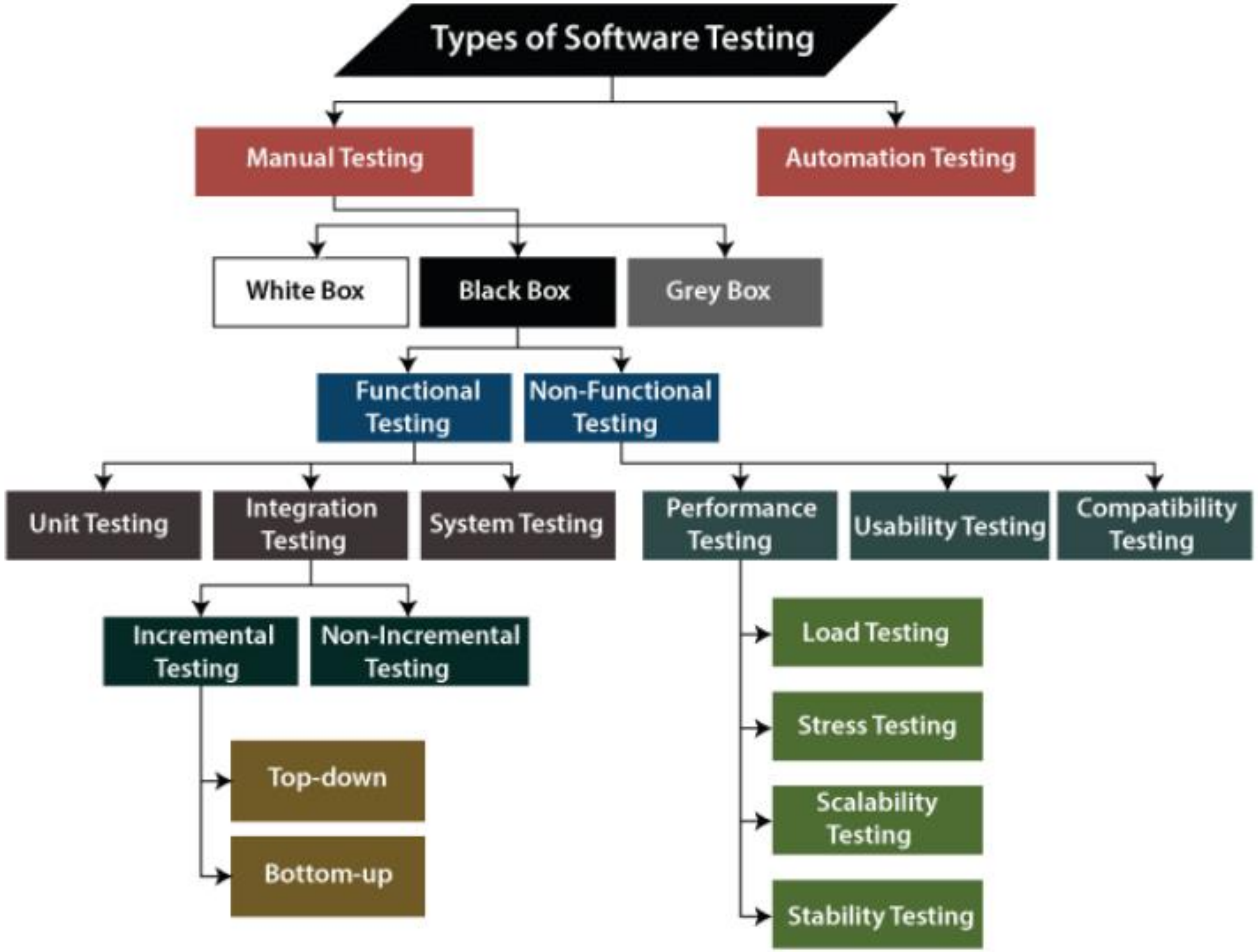
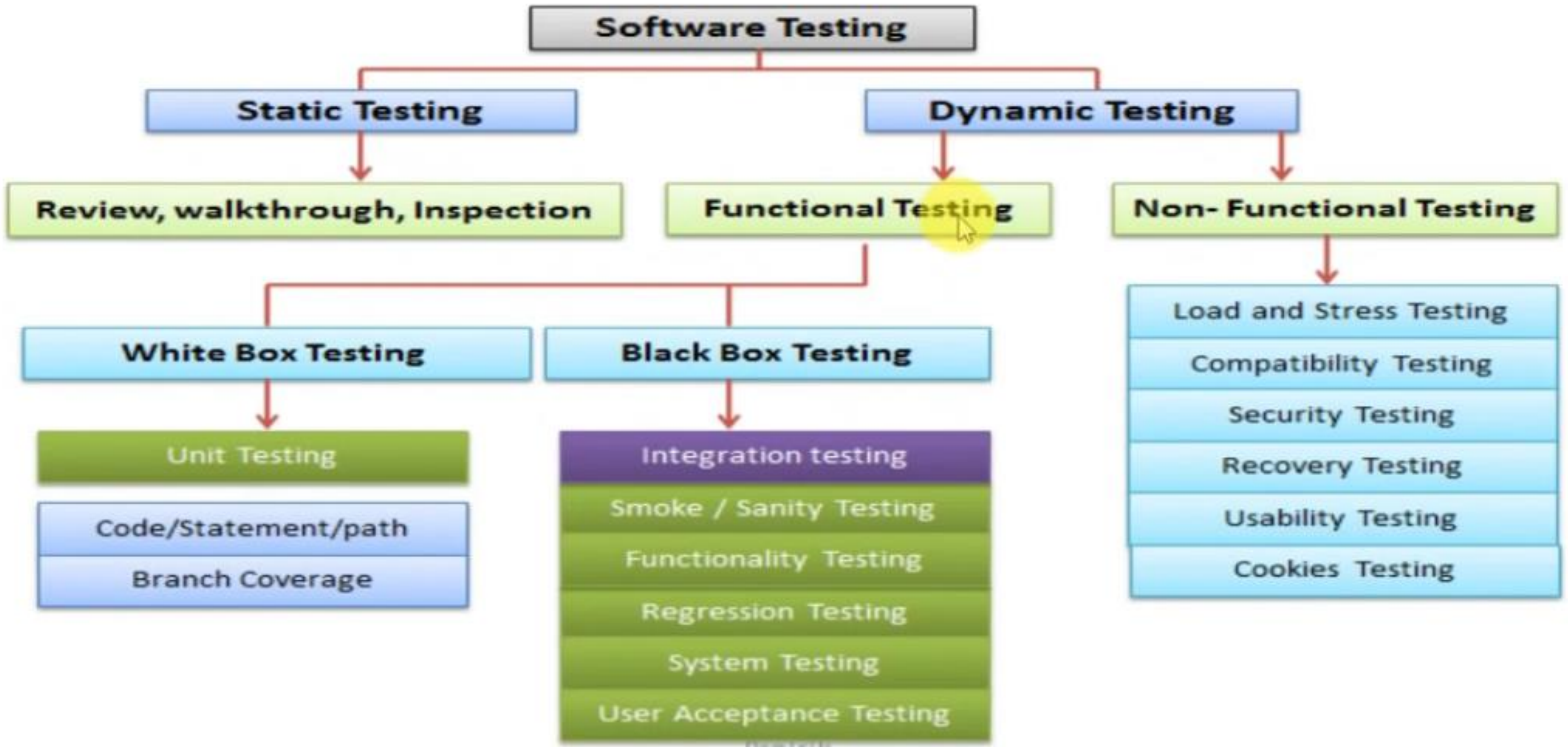
Levels of Testing

major phases of testing

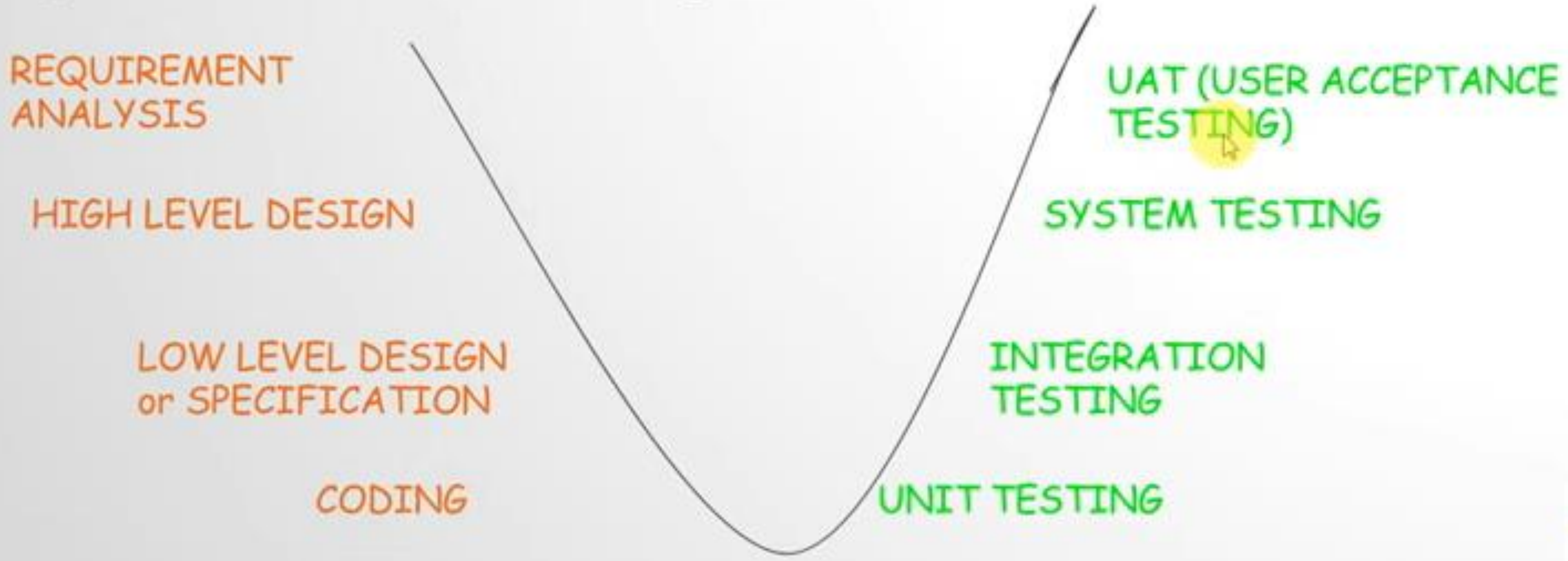

Types of Software Testing
Manual Testing
Automation Testing
White Box
Black Box
Grey Box
Functional Testing
Non-Functional Testing
Unit Testing
Integration Testing
System Testing
Performance Testing
Usability Testing
Compatibility Testing
Incremental Testing
Non-Incremental Testing
Top-down
Bottom-up
Load Testing
Stress Testing
Scalability Testing
Stability Testing

Types of Software Testing:

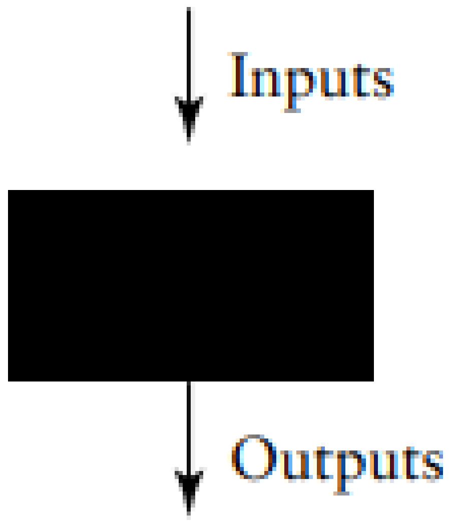
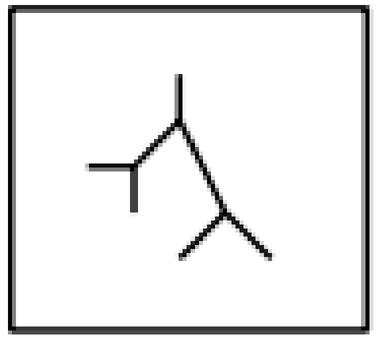


V Model:

"In V model of SDLC the real development phases and testing plans goes side by side as It can be interpreted as in a 'V' shape"

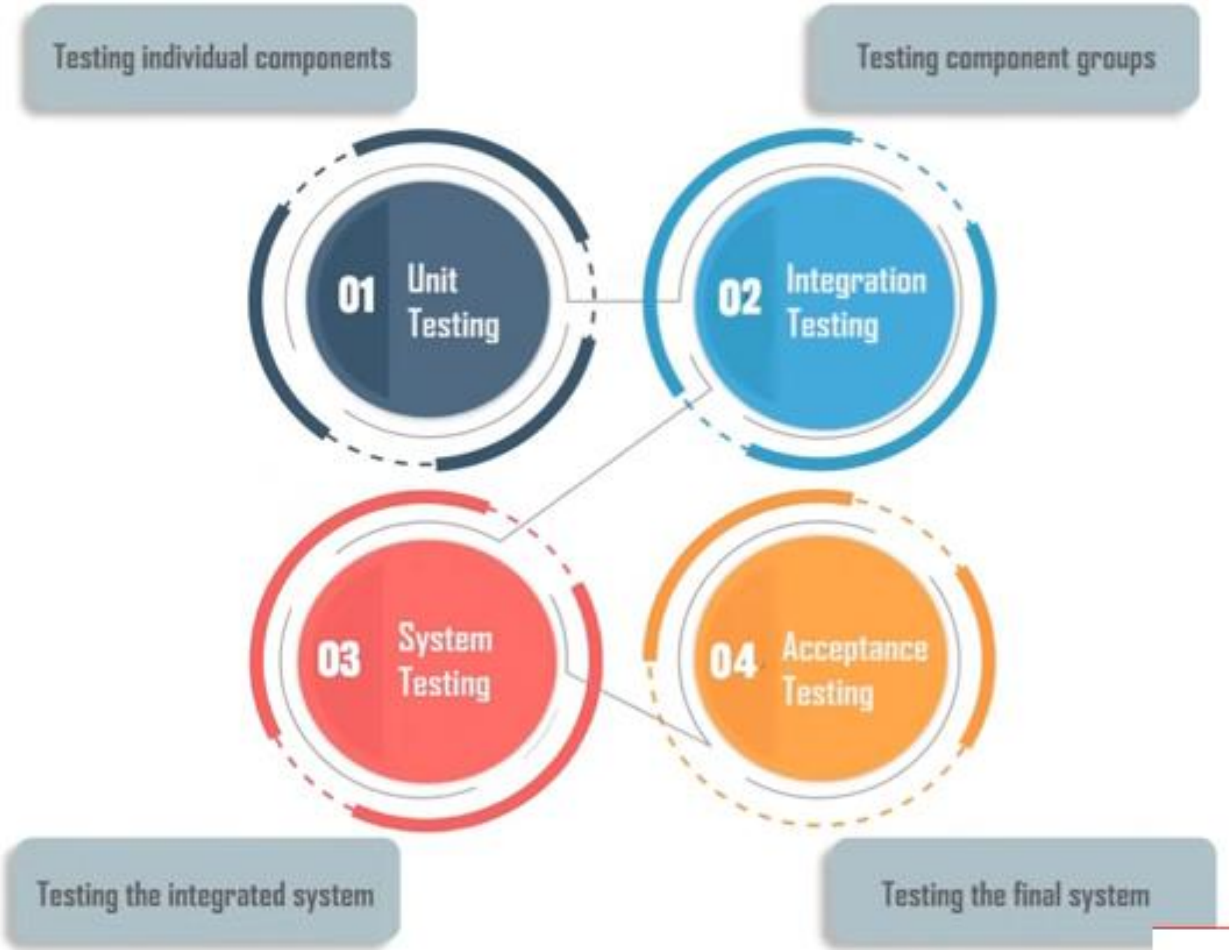


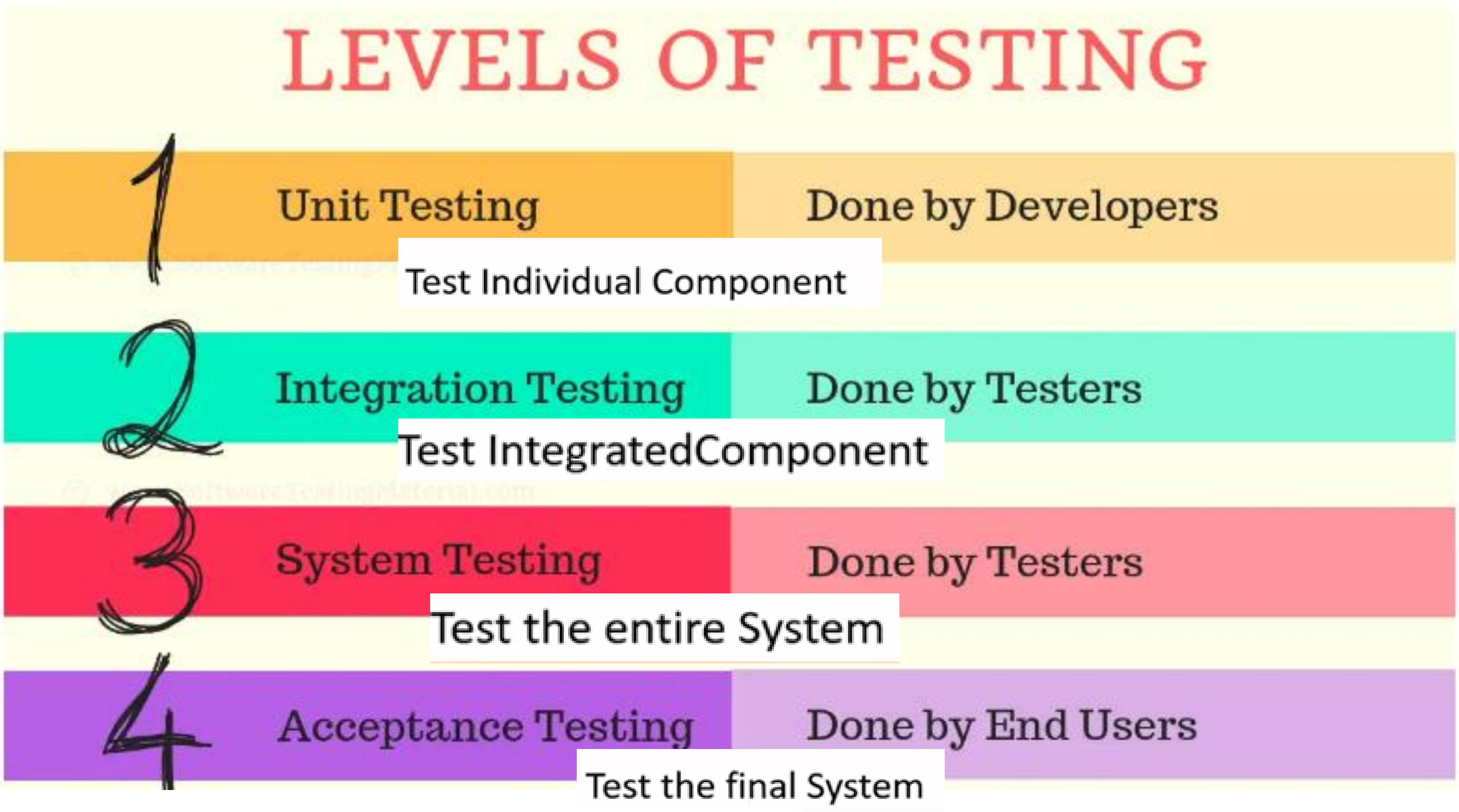
The two basic testing strategies.

Test Strategy	Tester's View	Knowledge Sources	Methods
Black box		Requirements document Specifications Domain knowledge Defect analysis data	Equivalence class partitioning Boundary value analysis State transition testing Cause and effect graphing Error guessing
White box		High-level design Detailed design Control flow graphs Cyclomatic complexity	Statement testing Branch testing Path testing Data flow testing Mutation testing Loop testing

Test Strategy	Tester's View	Knowledge Sources	Methods
Black box		Requirements document Specifications	Equivalence class Partitioning Boundary value analysis State transition testing Cause and effect graphing
White box		High-levign Detailed design	Statement testing Branch testing Path testing Data flow testing Mutation testing

Levels of Software Testing

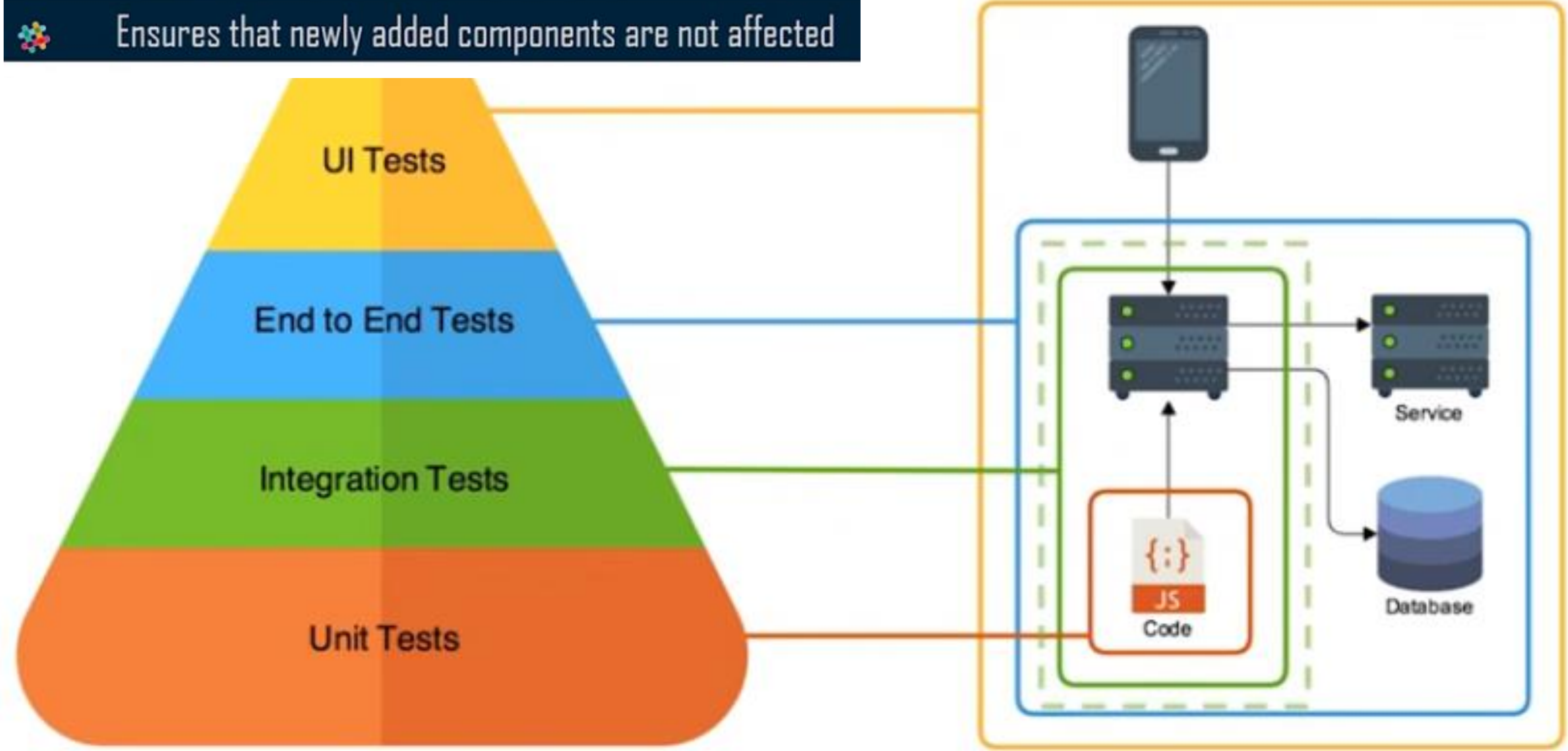


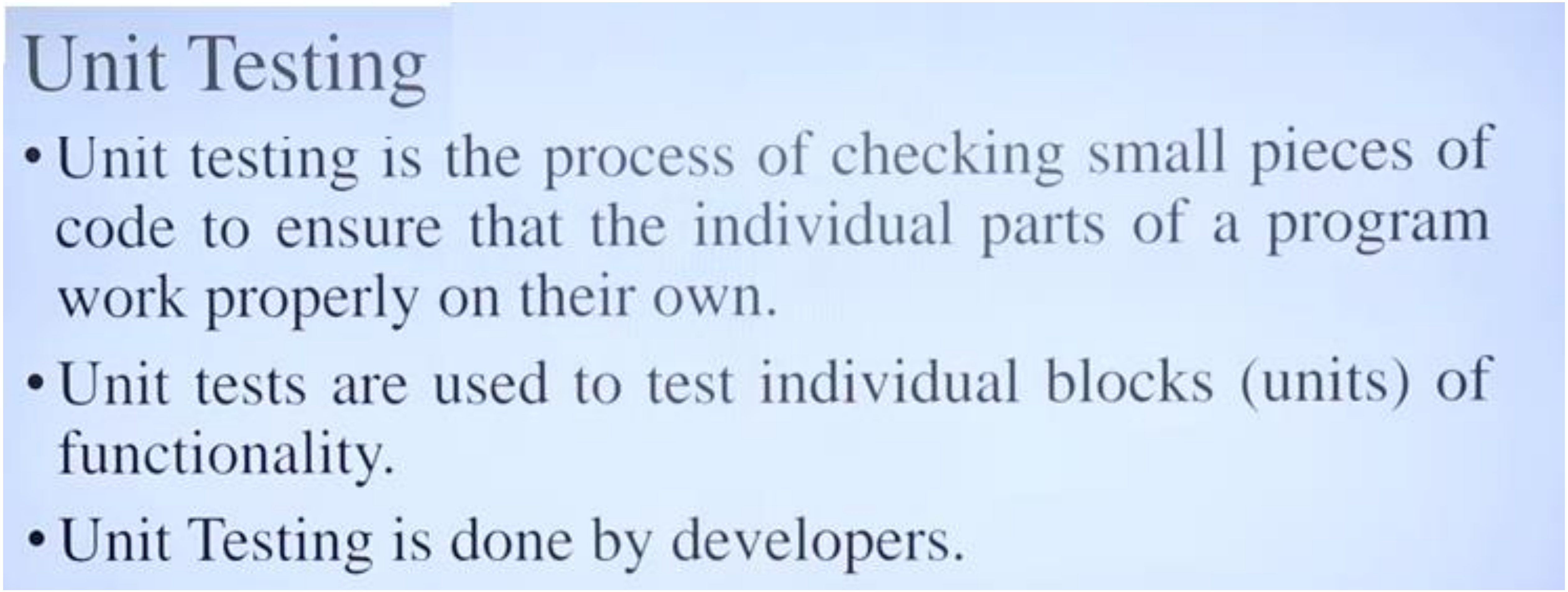


LEVELS OF TESTING

1	Unit Testing	Done by Developers	Test Individual Component
2	Integration Testing	Done by Testers	Test IntegratedComponent
3	System Testing	Done by Testers	Test the entire System
4	Acceptance Testing	Done by End Users	Test the final System

- Ensures modules work together properly
- Helps to uncover errors that lie in interfaces
- Ensures that newly added components are not affected

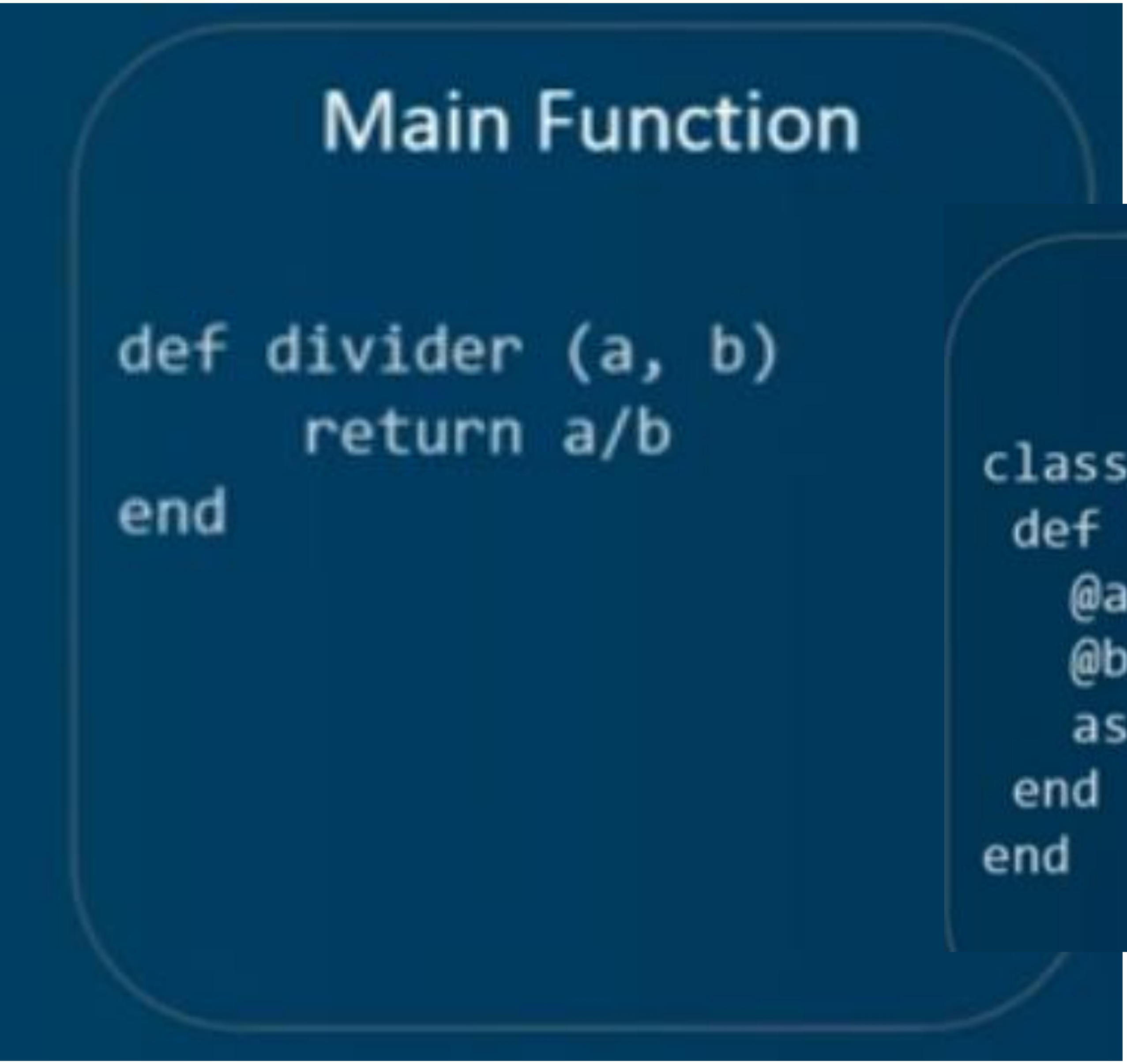




Unit Testing

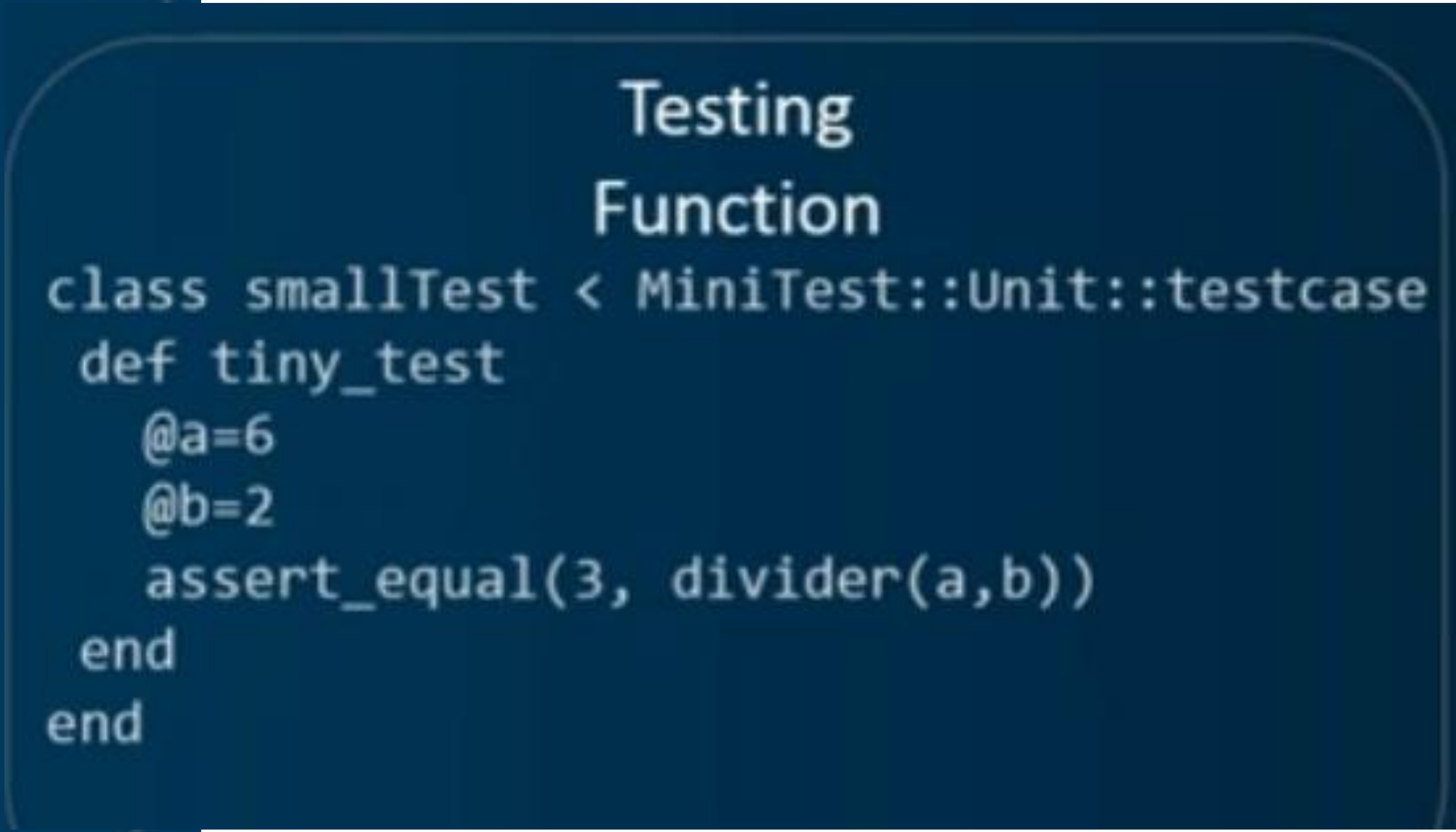
- Unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own.
- Unit tests are used to test individual blocks (units) of functionality.
- Unit Testing is done by developers.

Unit testing is a way of testing the smallest piece of code referred to as a **unit** that can be logically isolated in a system. It is mainly focused on the functional correctness of standalone modules.



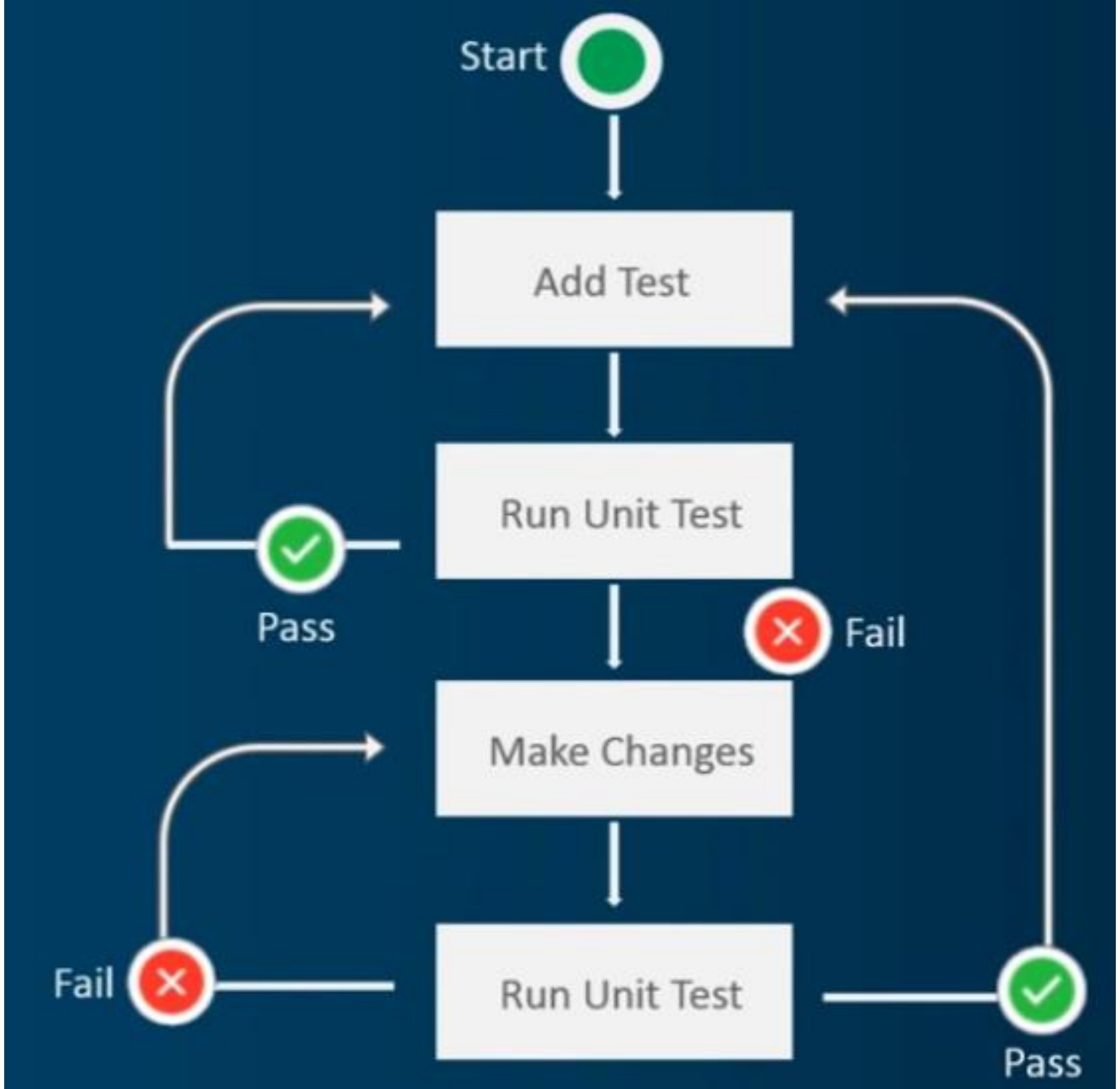
Main Function

```
def divider (a, b)
      return a/b
end
```

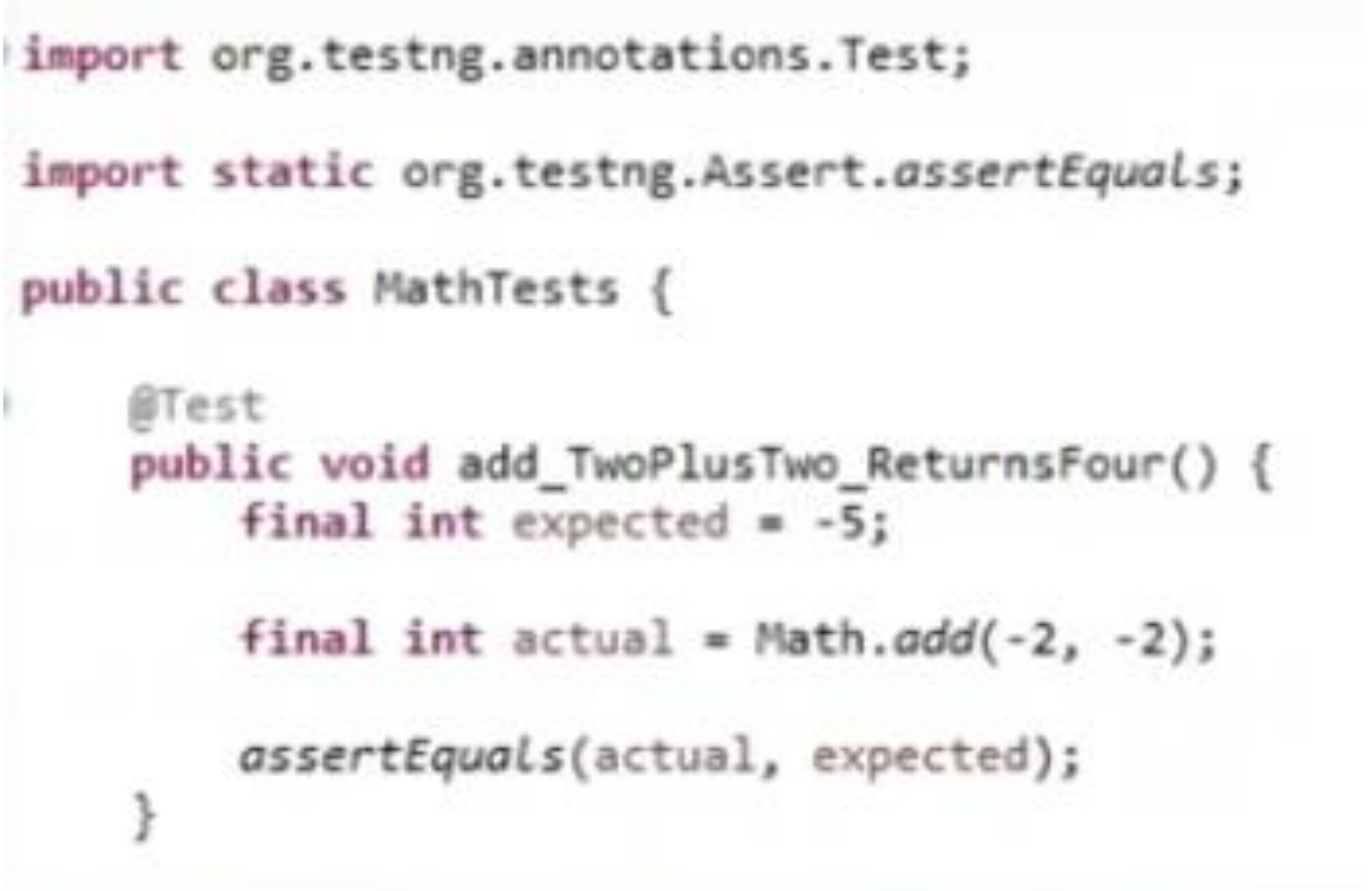


Testing Function

```
class smallTest < MiniTest::Unit::testcase
 def tiny_test
   @a=6
   @b=2
   assert_equal(3, divider(a,b))
 end
end
```



Unit Testing Algorithm



```
import org.testng.annotations.Test;

import static org.testng.Assert.assertEquals;

public class MathTests {

    @Test
    public void add_TwoPlusTwo_ReturnsFour() {
        final int expected = -5;

        final int actual = Math.add(-2, -2);

        assertEquals(actual, expected);
    }
```

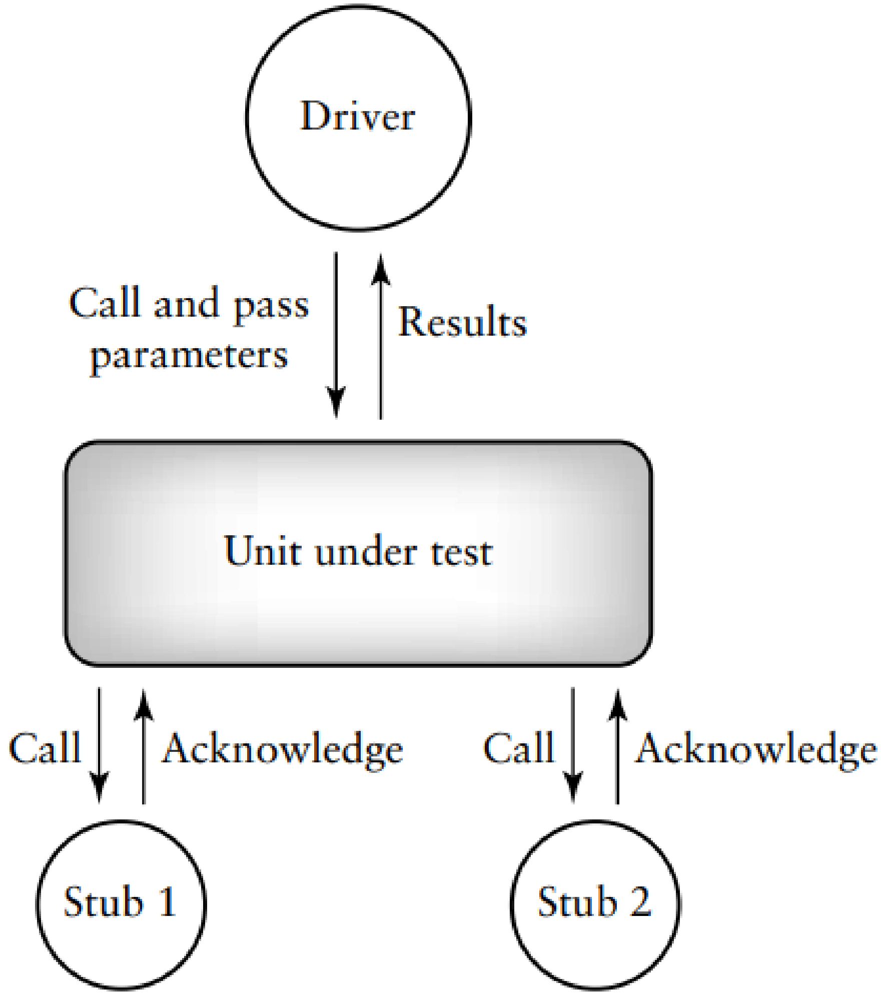
Unit testing framework for C#: NUnit

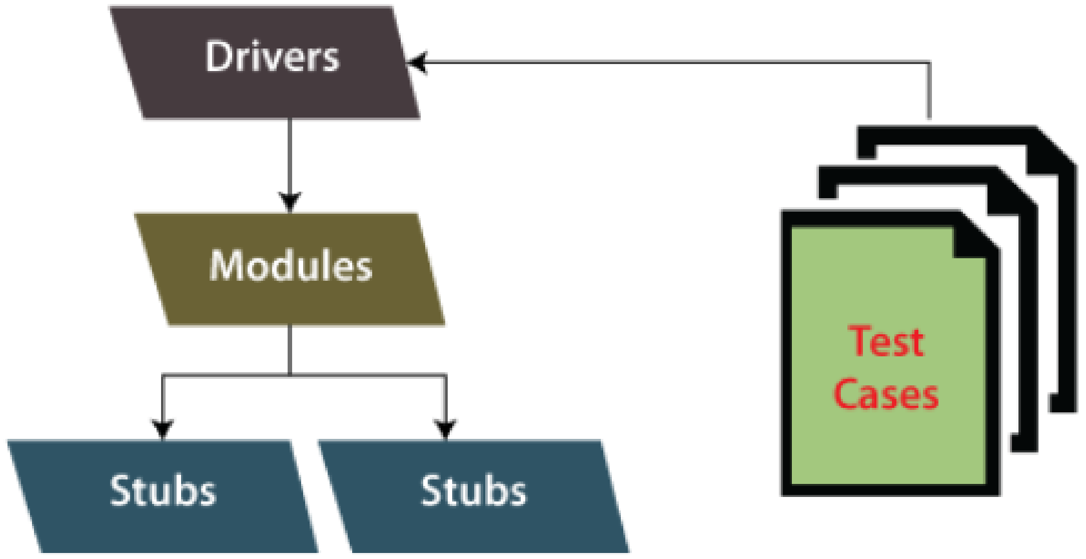
Unit testing framework for Java: JUnit, TestNG

Unit testing framework for C & C++: Embunit

The test harness.

The auxiliary code developed to support testing of units and components is called a test harness. The harness consists of drivers that call the target code and stubs that represent modules it calls.





Summary work sheet for unit test

Unit Test Worksheet

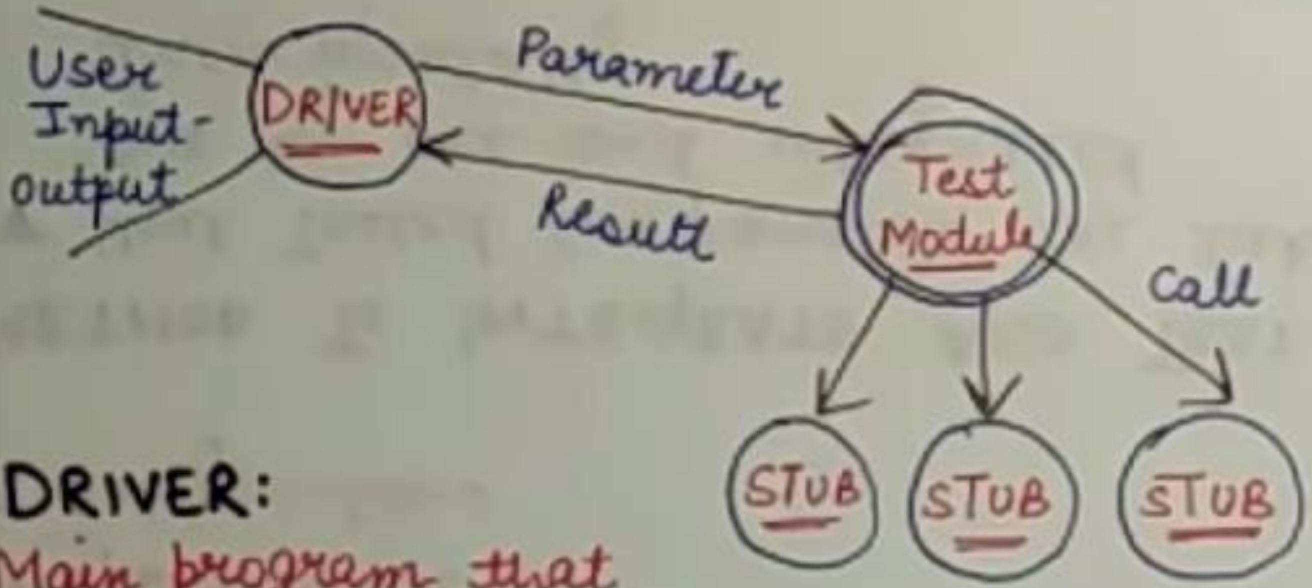
Unit Name: ______________________________

Unit Identifier: ______________________________

Tester: ______________________________

Date: ______________________________

Test case ID	Status (run/not run)	Summary of results	Pass/fail



DRIVER:

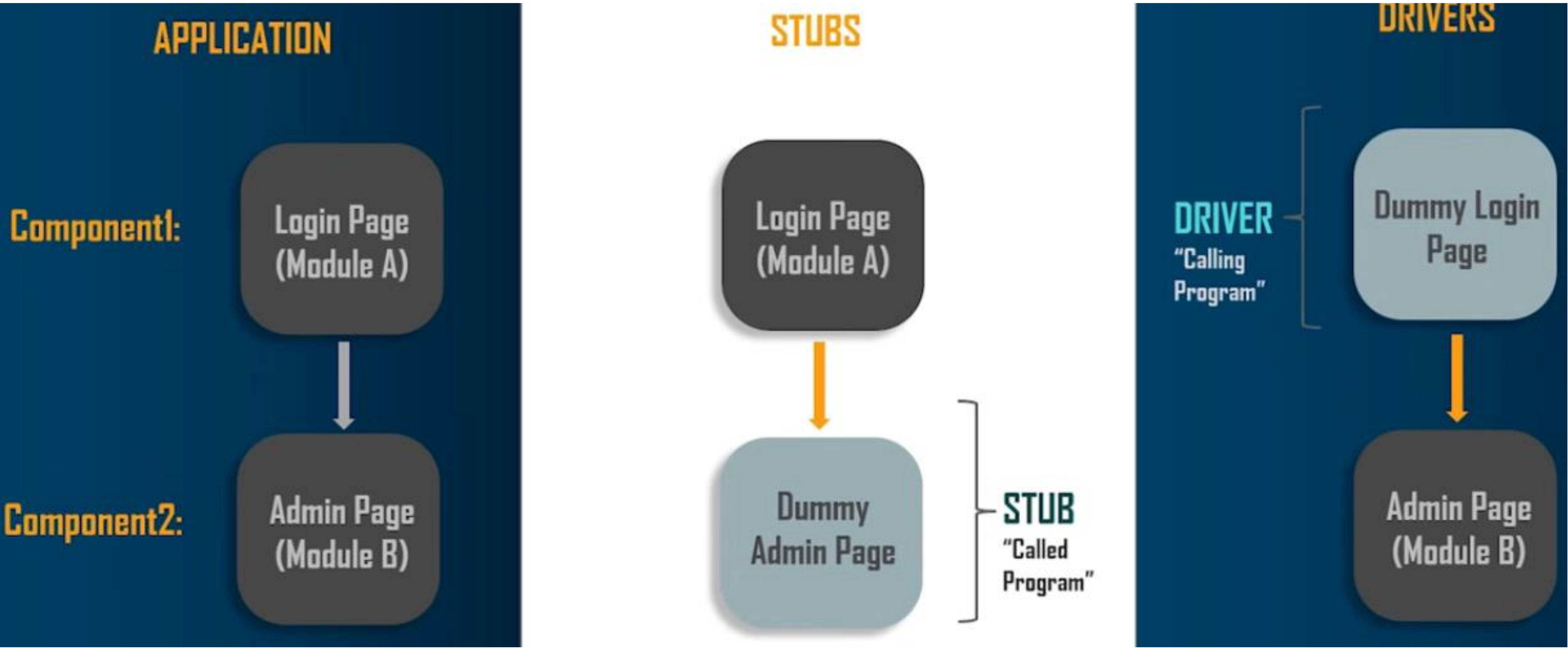
Main program that accepts test case data, passes data to the component to be tested and prints relevant results.

STUB:

Subordinate Modules that are called by the module to be tested

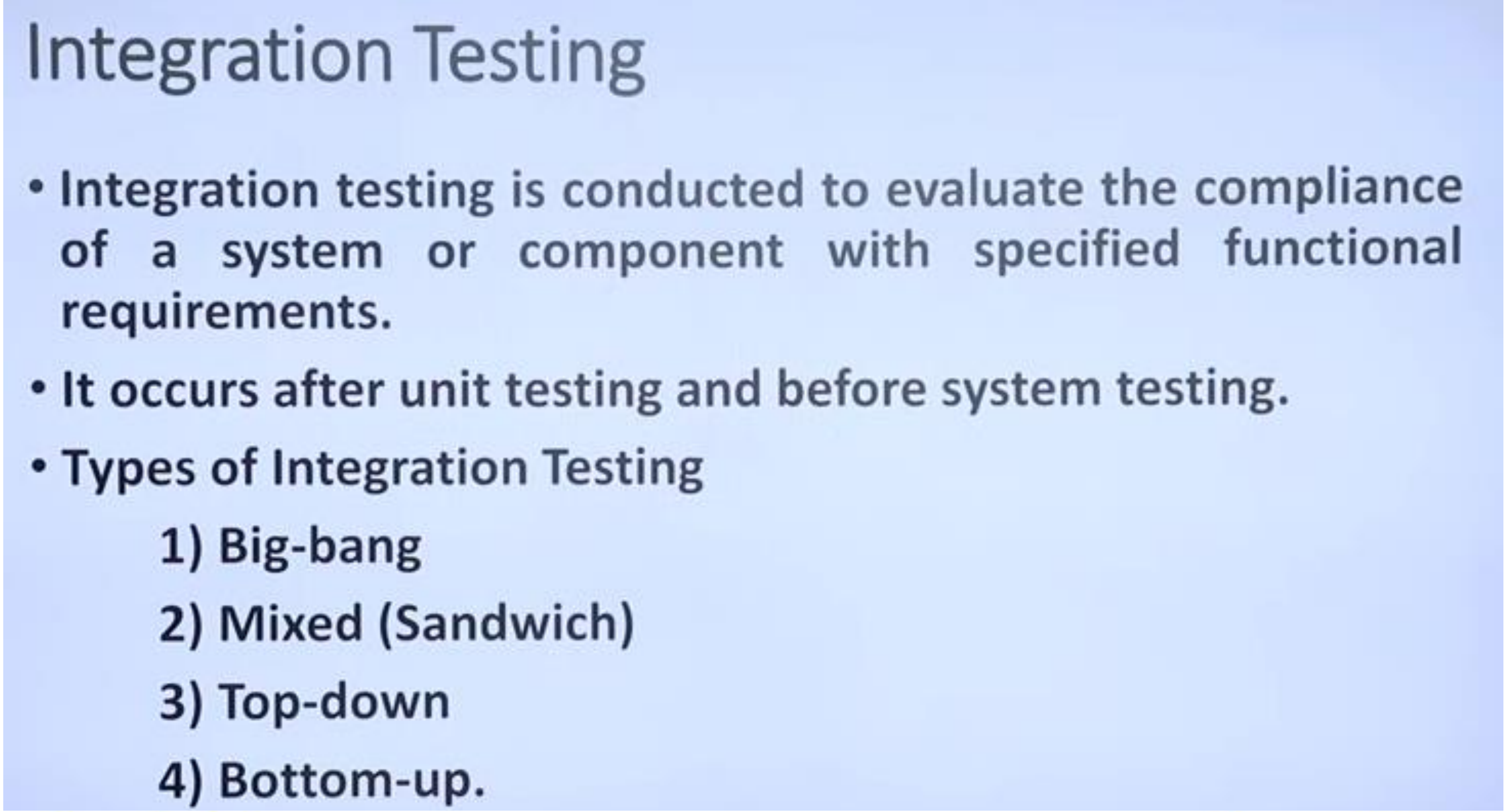
- It is a dummy sub-program that does minimal data manipulation, provides verification of entry and returns the control to module under testing.

Stubs & Drivers



Test Harness

1. Consider an e-commerce application that offers users décor items. The website consists of several modules: homepage, product pages, cart, Wishlist, and payment gateway, to name a few. You need to test every feature of the site and ensure that no issues slip away. However, it is difficult to perform the entire testing process in one single shot. With Testsigma, you can run the test cases across 3000+ real devices and browsers. Your testers do not need to spend much time writing test cases and maintaining input data. Testsigma allows you no-code testing with test data management options. And you receive reports and analytics documents to share with all your stakeholders. You can also integrate our test automation tool with multiple bug-tracking tools, CI/CD software, and collaboration platforms. And all of these different aspects would come under test harness for this particular testing.



Integration Testing

- Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.
- It occurs after unit testing and before system testing.
- Types of Integration Testing
  1) Big-bang
  2) Mixed (Sandwich)
  3) Top-down
  4) Bottom-up.

Integration Testing is a level of software testing where individual units are combined and the connectivity or data transfer between these units is tested. The main aim of this testing is to recognize the interface between the modules.

Integration Testing Approaches

Top–Down Approach(TDA)

Testing takes place from top to bottom, following the control flow or architectural structure

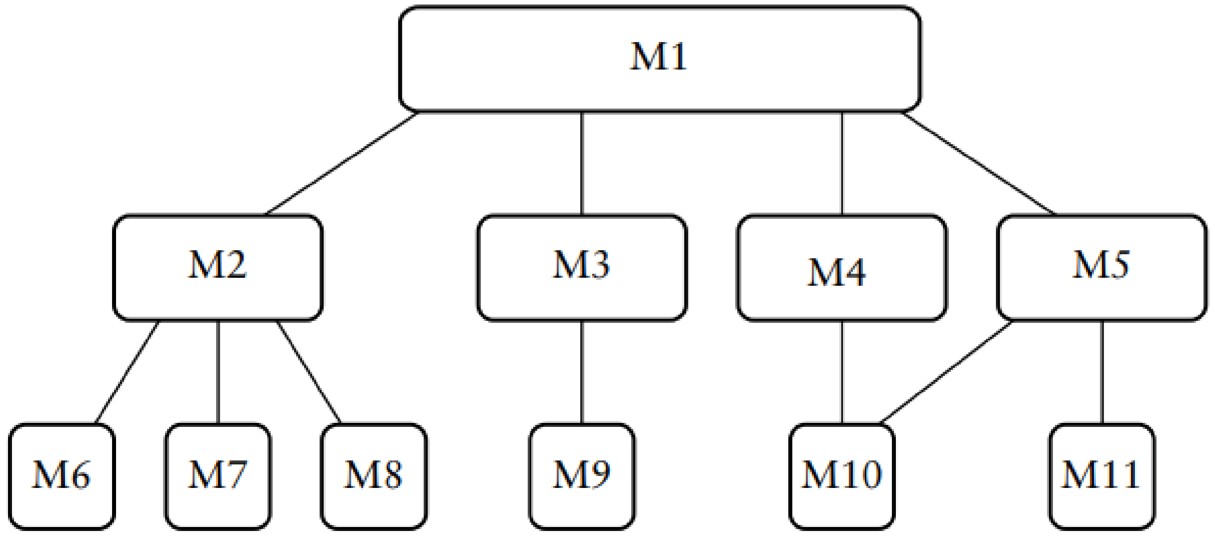
Bottom-UP Approach(BUA)

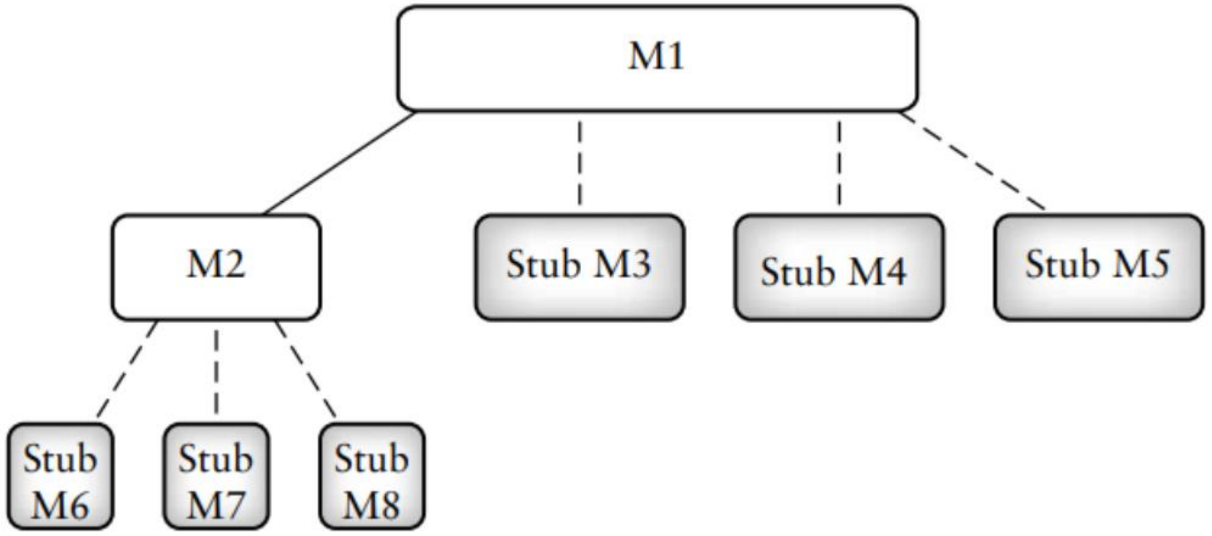
Testing takes place from the bottom of the control flow upwards

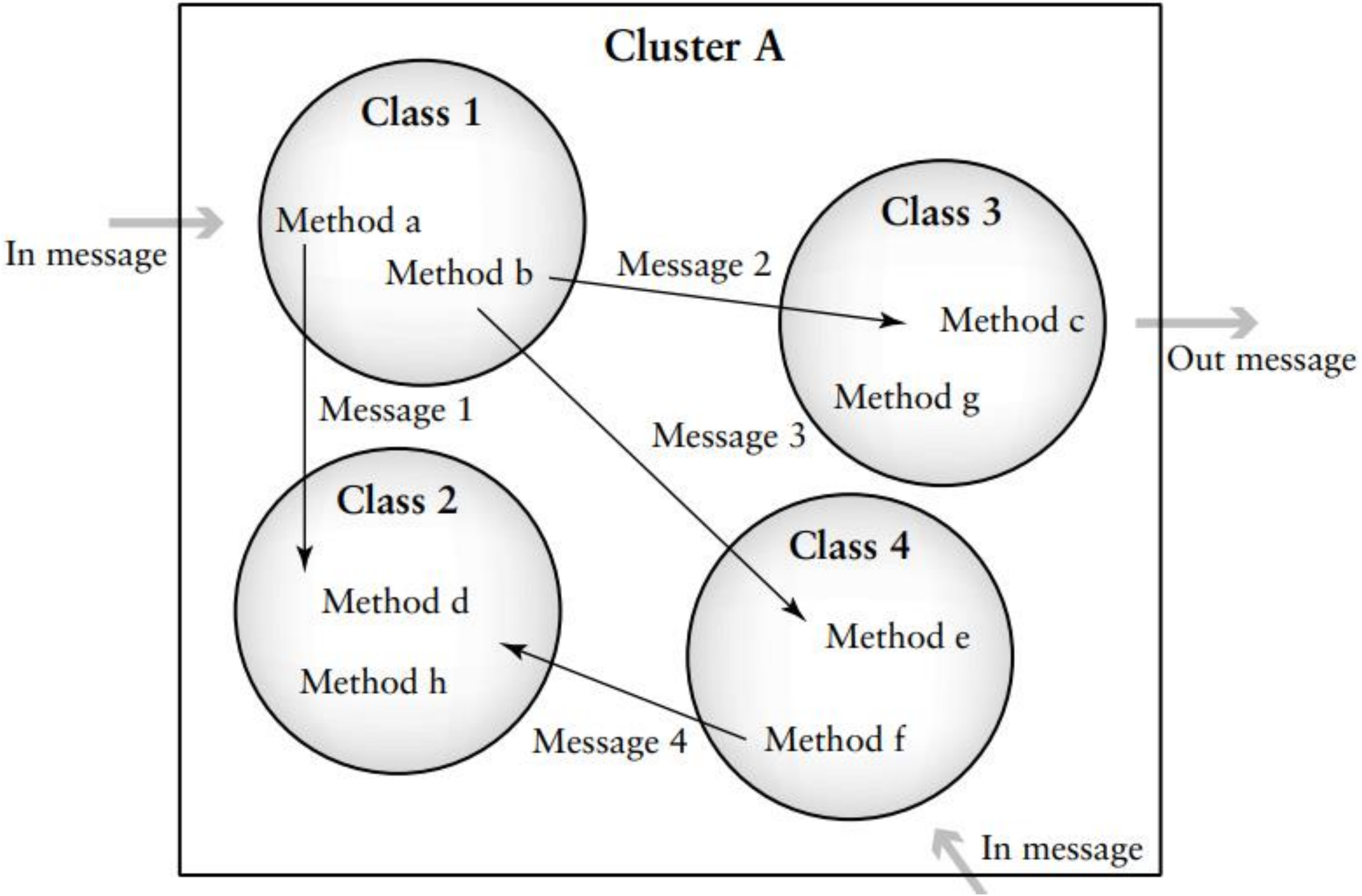
Big Bang Approach

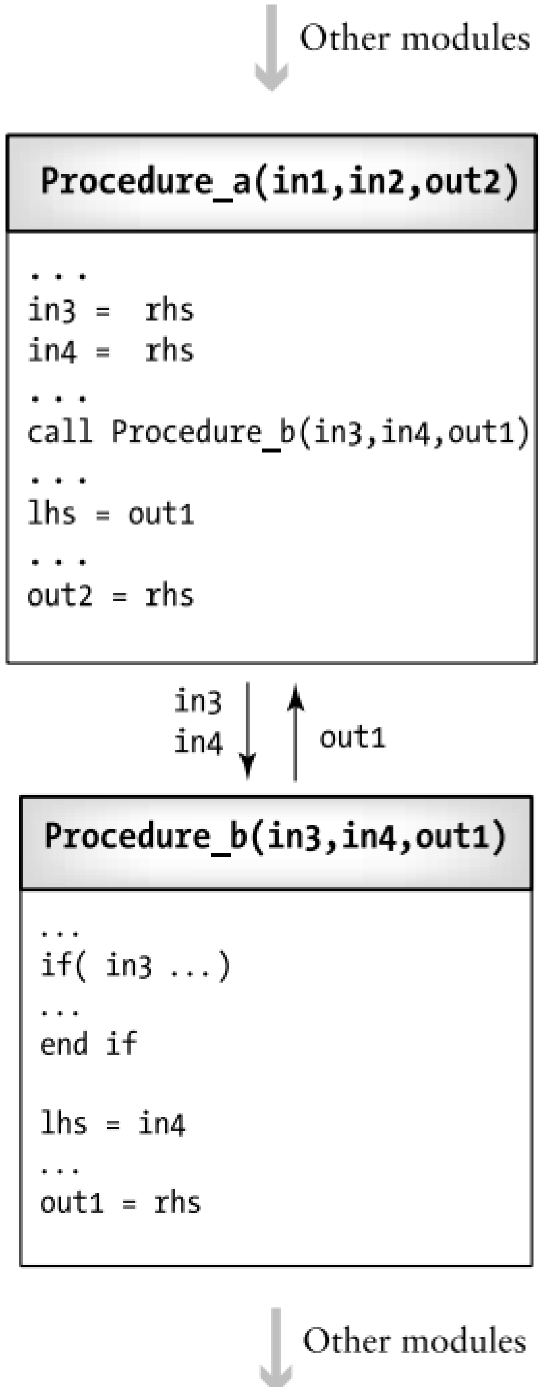
All components or modules are integrated simultaneously, after which everything is tested as a whole

Simple structure chart for integration test examples






Cluster A
Class 1
Method a
Method b
Class 2
Method d
Method h
Class 3
Method c
Method g
Class 4
Method e
Method f
Message 1
Message 2
Message 3
Message 4
In message
Out message
In message



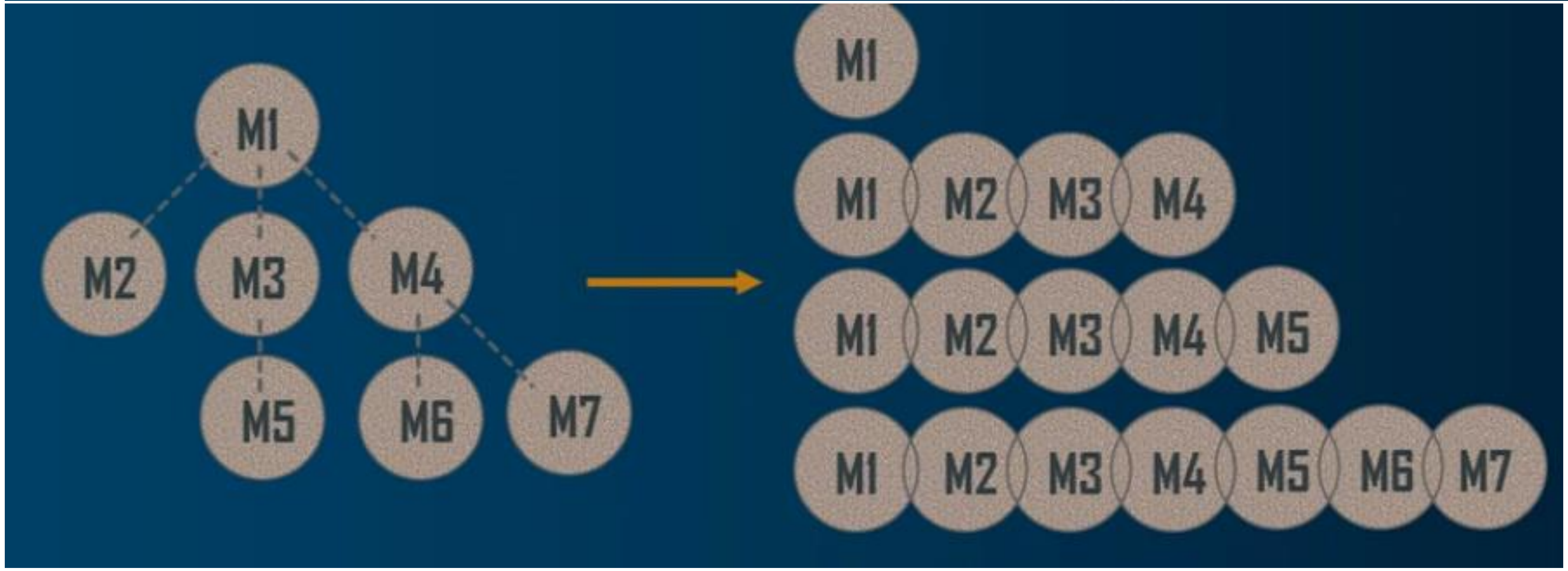
```
Procedure_a(in1,in2,out2)

. . .
in3 =  rhs
in4 =  rhs
. . .
call Procedure_b(in3,in4,out1)
. . .
lhs = out1
. . .
out2 = rhs
```

```
Procedure_b(in3,in4,out1)

...
if( in3 ...)
...
end if

lhs = in4
...
out1 = rhs
```

Example integration of two procedures.

Top-Down Approach(TDA)

Testing takes place from top to bottom, following the control flow or architectural structure



Advantages

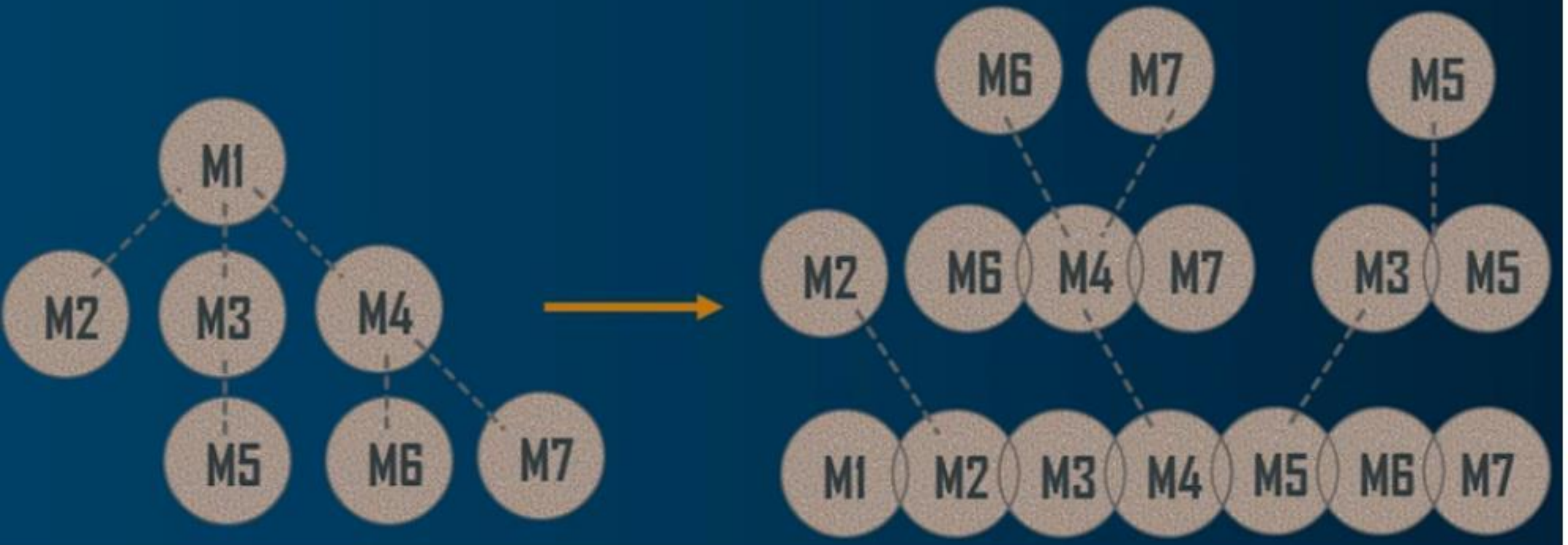
- Extremely consistent
- Less time required
- Fault localization is easier
- Detects major flaws

Disadvantages

- Requires several stubs
- Poor support for early release
- Basic functionality is tested late

Bottom-UP Approach(BUA)

Testing takes place from the bottom of the control flow, upwards



Advantages

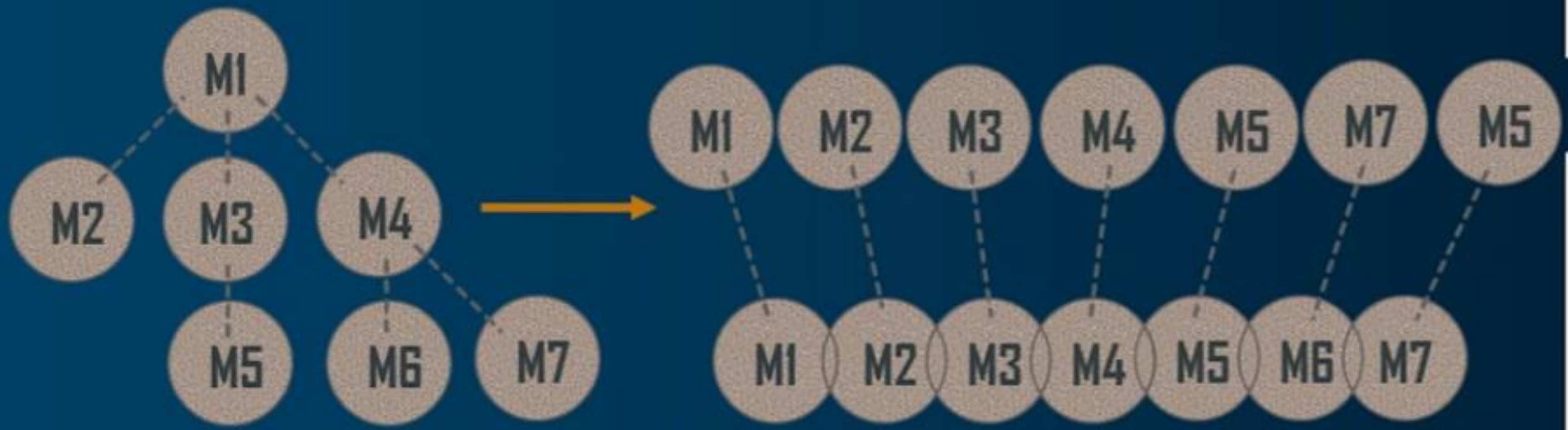
- Efficient application
- Less time requirements
- Test conditions are easier to create

Disadvantages

- Requires several drivers
- Data flow is tested late
- Poor support for early release
- Key interface defects are detected late

Big Bang Approach

All components or modules are integrated simultaneously, after which everything is tested as a whole



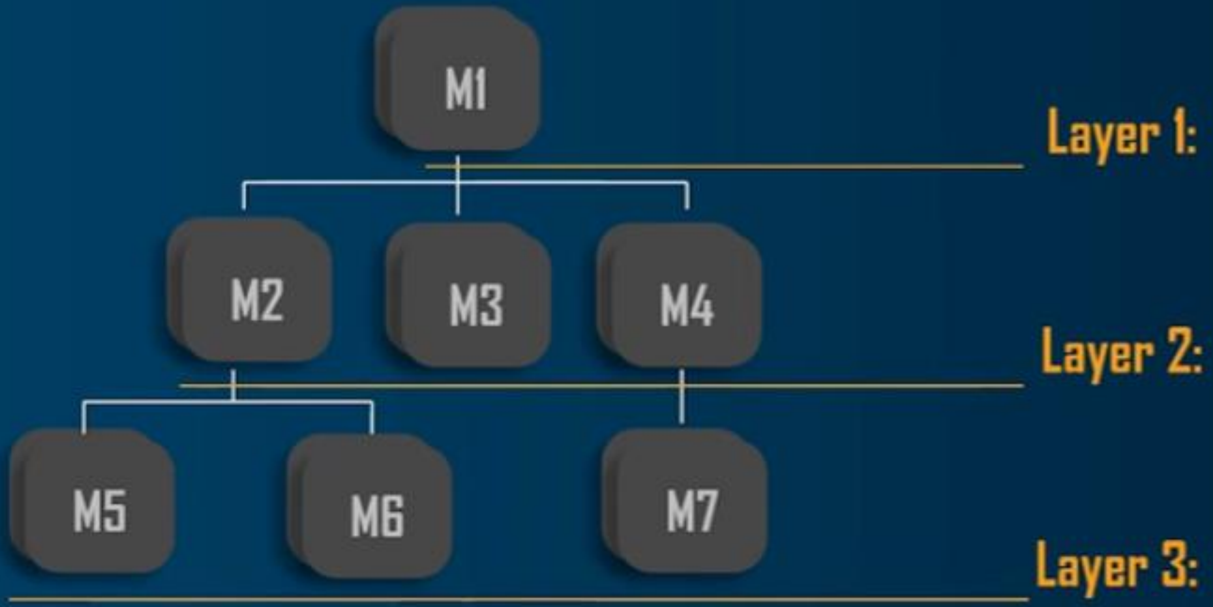
Advantages

- All components tested at once
- Convenient for small systems
- Saves testing time

Disadvantages

- Lot of delay before testing
- Difficult to trace cause of failures
- Possibility of few missing interface links
- Critical modules are not prioritized

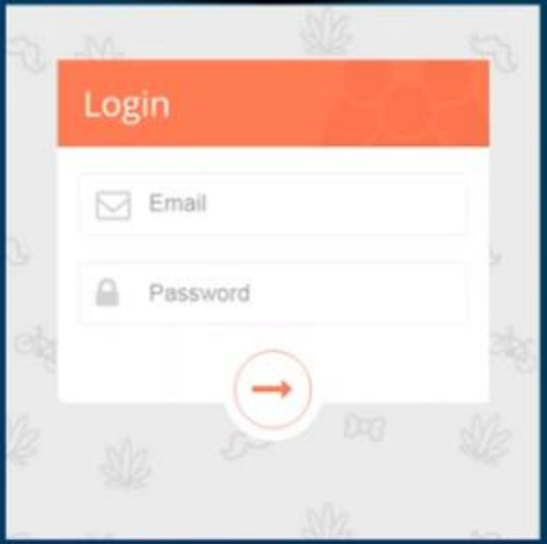
Sandwich Integration Approach



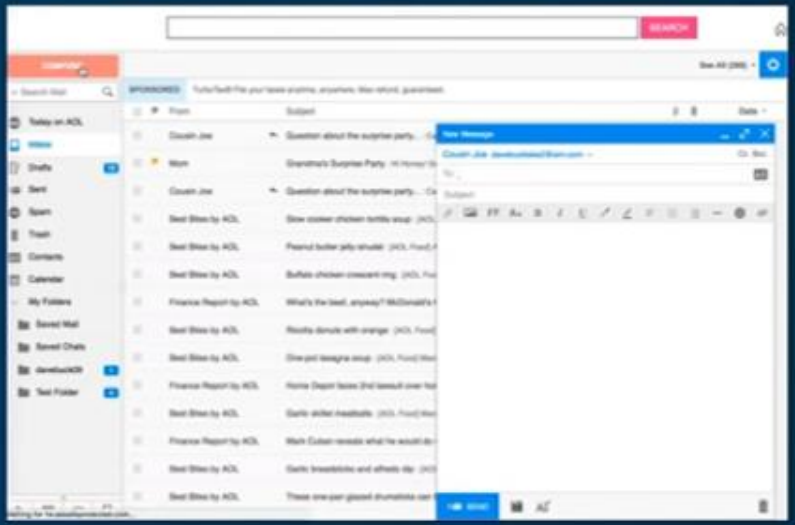
- Also called *Hybrid Integration Testing* or *Mixed Integration Testing*
- Middle layer is the target layer
- Top-Down approach is topmost layer
- Bottom-Up approach is lowermost layer
- Advantage: Both layers can be tested in parallel
- Disadvantage: High cost, big skill set, extensive testing is not done

Example of Integration Testing

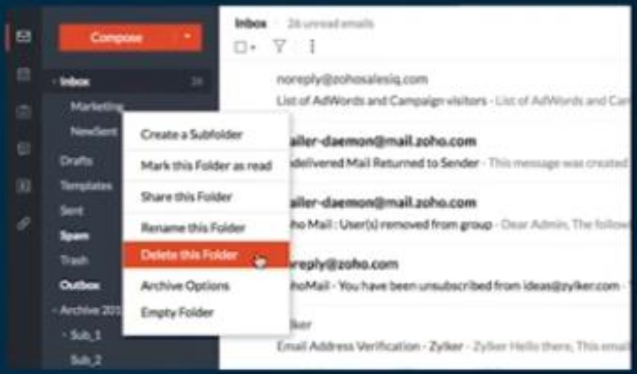
Consider an application with three modules, Login Page, Mailbox, and Delete emails. All these modules are integrated logically by programmers.



Login Page



Mail Box



Delete Emails

Example of Integration Testing

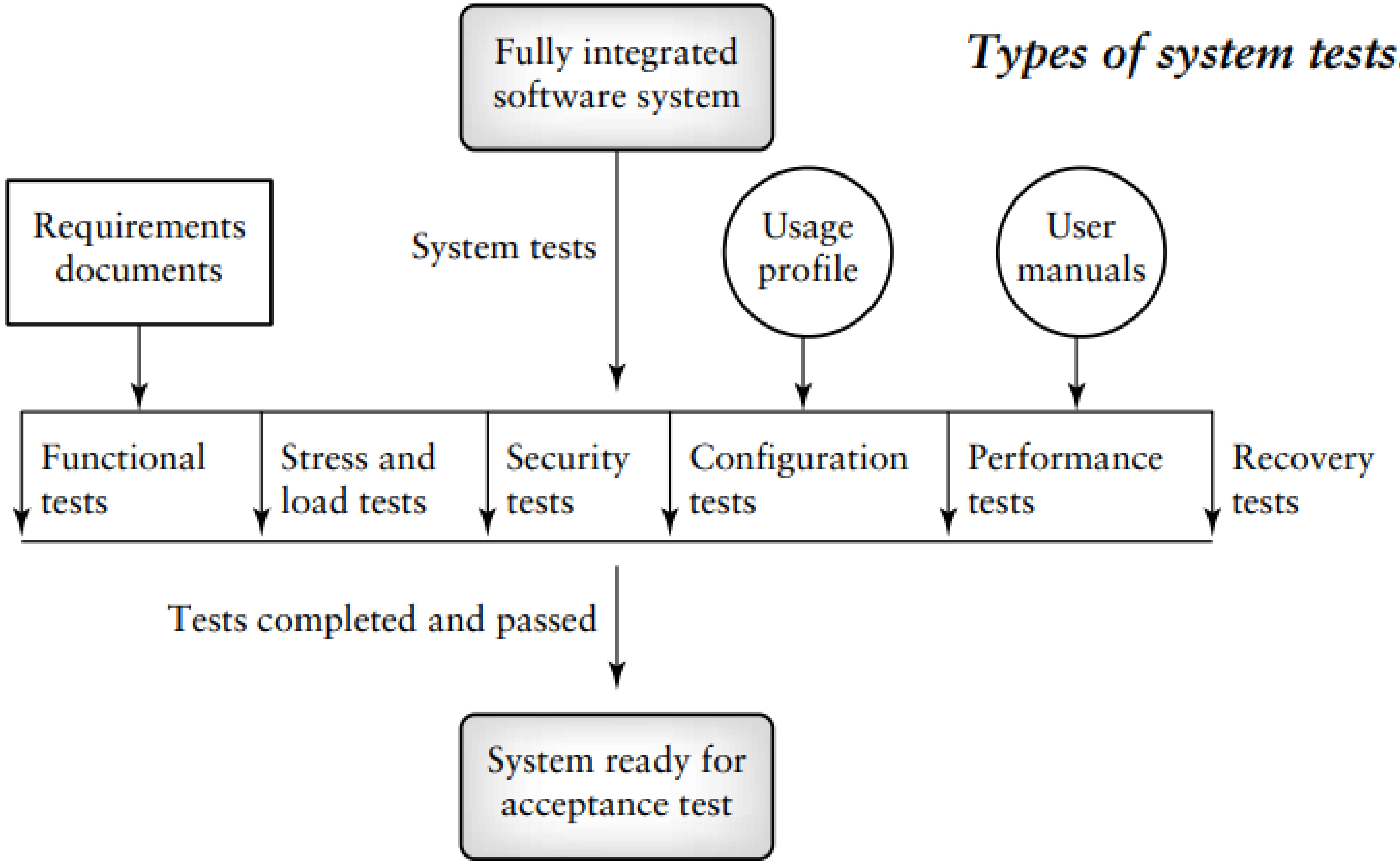
Consider an application with three modules, Login Page, Mailbox, and Delete emails. All these modules are integrated logically by programmers.

Test Case ID	Test Case Objective	Test Case Description	Expected Outcome
A	Test the interface link between Login Page and the Mail Box Page	Enter the login details & click on login button to login	You should be directed to the Mail Box Page
B	Check the interface link between Mail Box & the Delete Email Module	From mail box select the email you want to delete & click on delete	Selected email should be deleted and should appear in the Trash Folder

System Test: The Different Types

- Functional testing
- Performance testing
- Stress testing
- Configuration testing
- Security testing
- Recovery testing

Types of system tests.



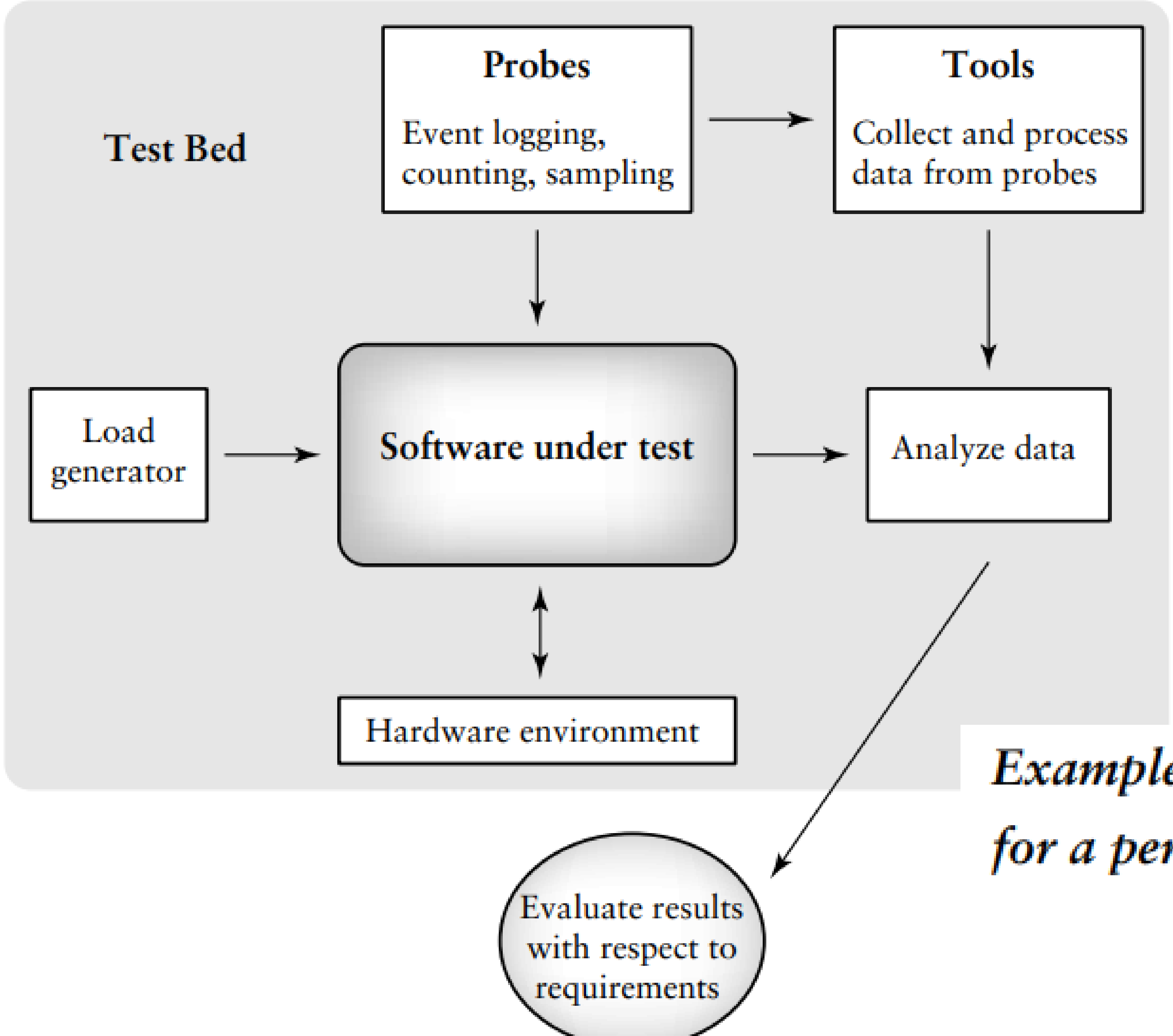
A load is a series of inputs that simulates a group of transactions.

operating system is required to handle 10 interrupts/second and the load causes 20 interrupts/second, the system is being stressed.

Configuration testing allows developers/testers to evaluate system performance and availability when hardware exchanges and reconfigurations occur.

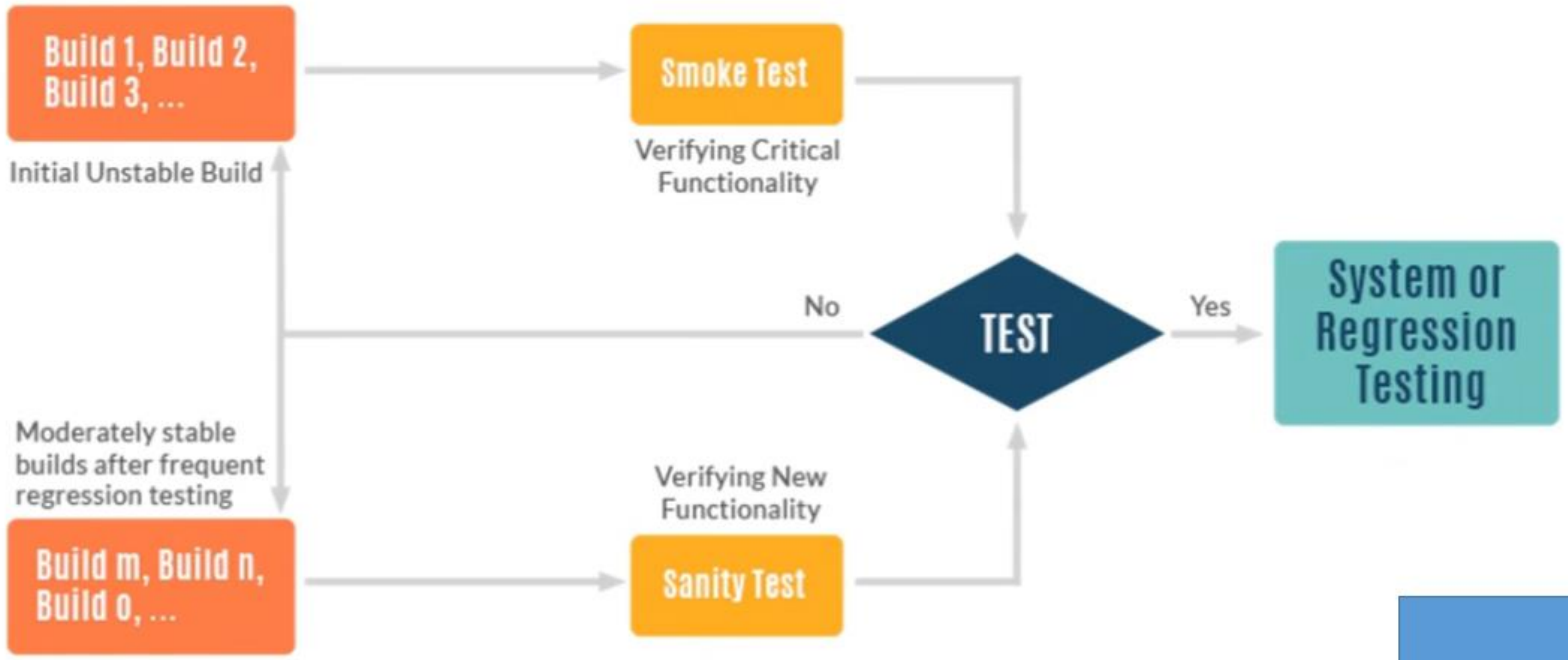
Designing and testing software systems to insure that they are safe and secure is a big issue facing software developers and test specialists.

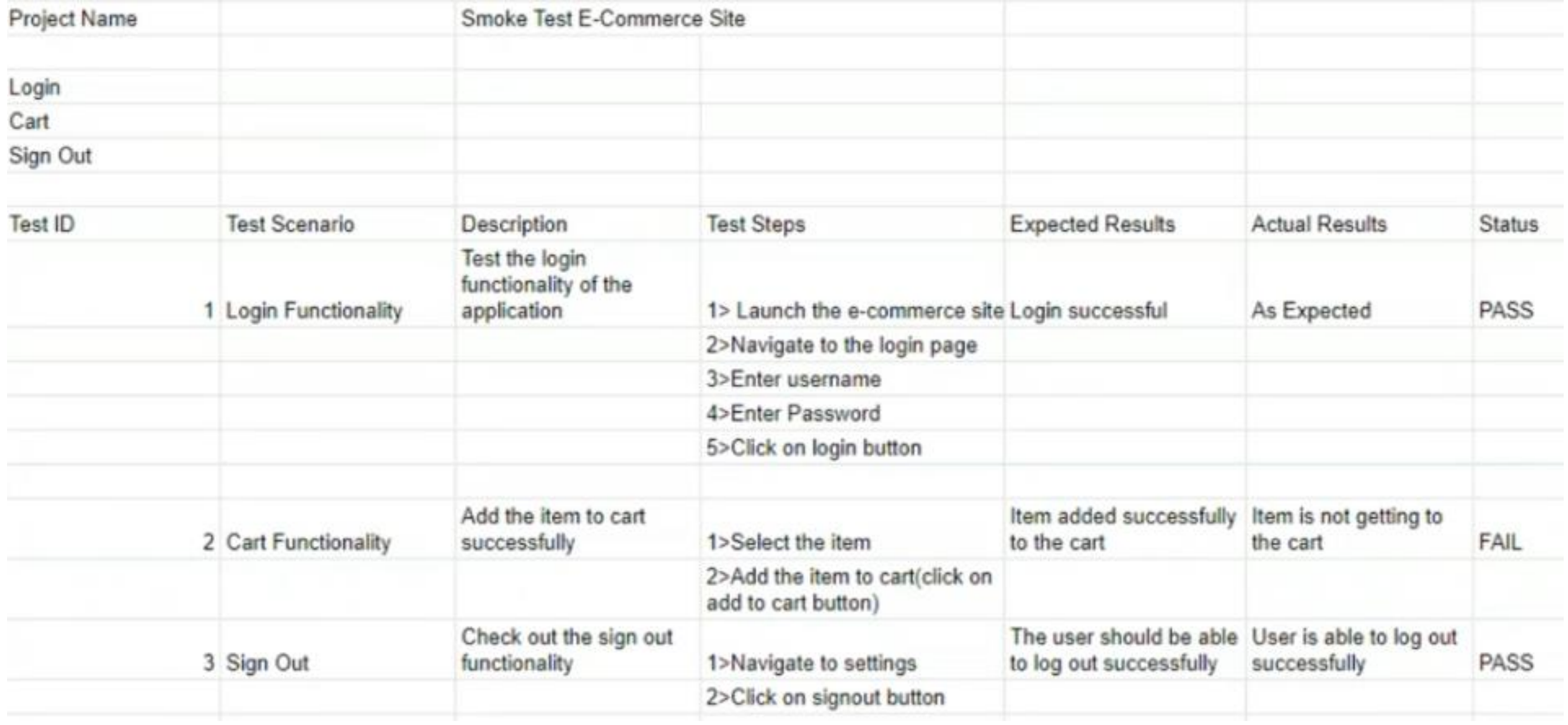
Recovery testing subjects a system to losses of resources in order to determine if it can recover properly from these losses.


Test Bed
Probes
Event logging, counting, sampling
Tools
Collect and process data from probes
Load generator
Software under test
Analyze data
Hardware environment
Evaluate results with respect to requirements

Examples of special resources needed for a performance test.

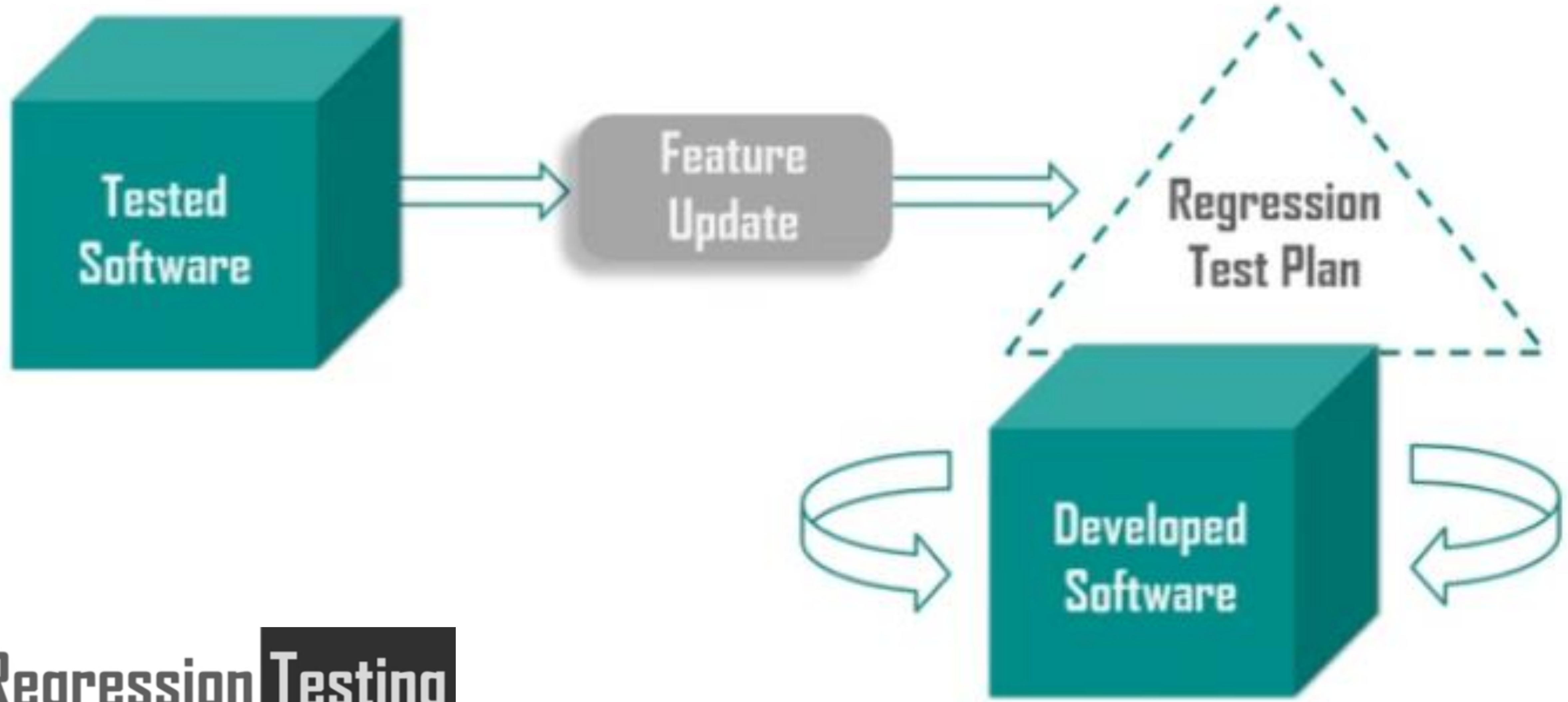
How is Smoke Testing Different from Sanity Testing?


Build 1, Build 2, Build 3, ...
Initial Unstable Build
Smoke Test
Verifying Critical Functionality
No
TEST
Yes
System or Regression Testing
Moderately stable builds after frequent regression testing
Verifying New Functionality
Build m, Build n, Build o, ...
Sanity Test



Project Name		Smoke Test E-Commerce Site
Login		
Cart		
Sign Out		

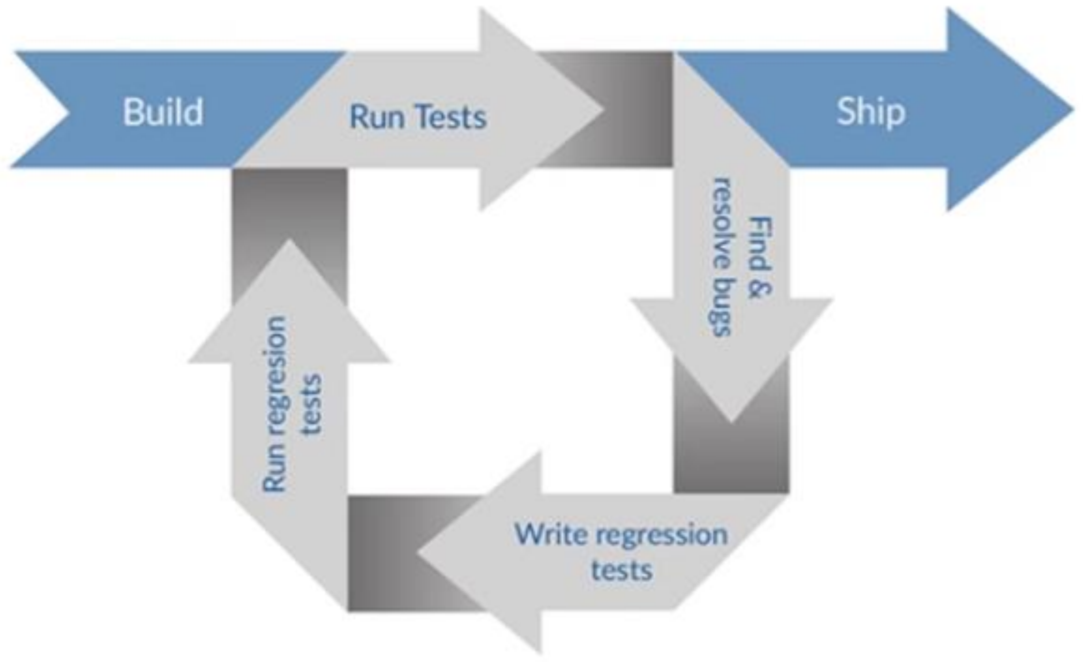
Test ID	Test Scenario	Description	Test Steps	Expected Results	Actual Results	Status
1	Login Functionality	Test the login functionality of the application	1> Launch the e-commerce site	Login successful	As Expected	PASS
			2>Navigate to the login page			
			3>Enter username			
			4>Enter Password			
			5>Click on login button			
2	Cart Functionality	Add the item to cart successfully	1>Select the item	Item added successfully to the cart	Item is not getting to the cart	FAIL
			2>Add the item to cart(click on add to cart button)			
3	Sign Out	Check out the sign out functionality	1>Navigate to settings	The user should be able to log out successfully	User is able to log out successfully	PASS
			2>Click on signout button			



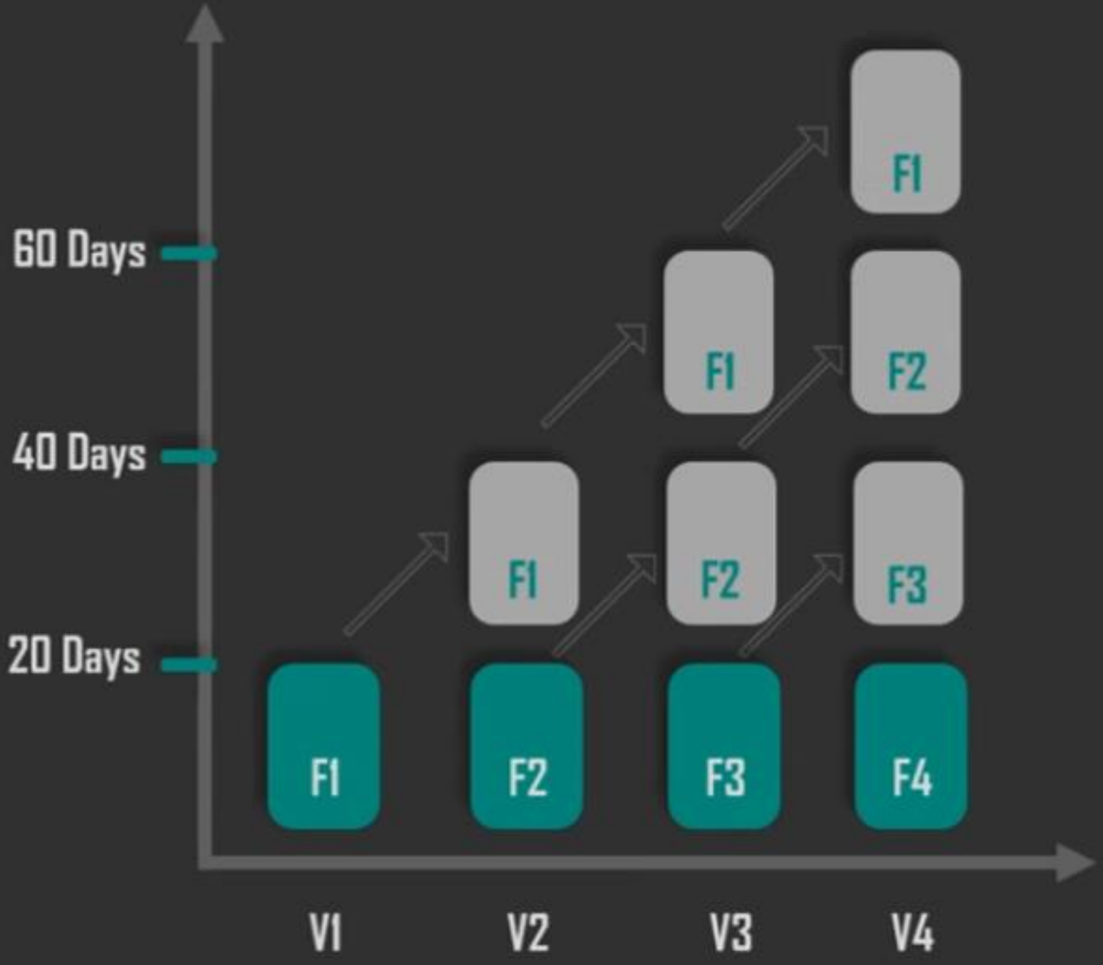
Regression Testing

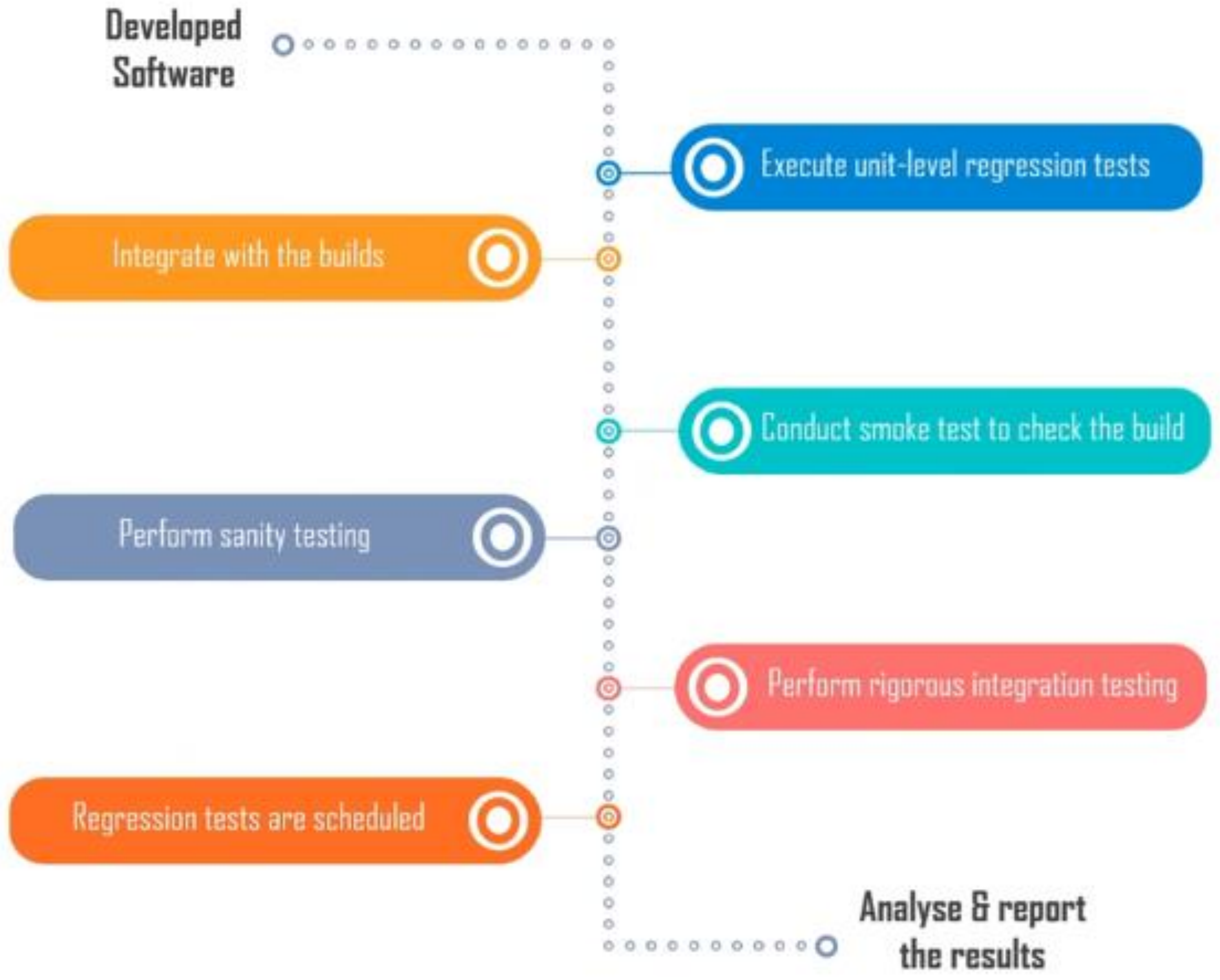
Testing of a previously tested program following modification, to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made

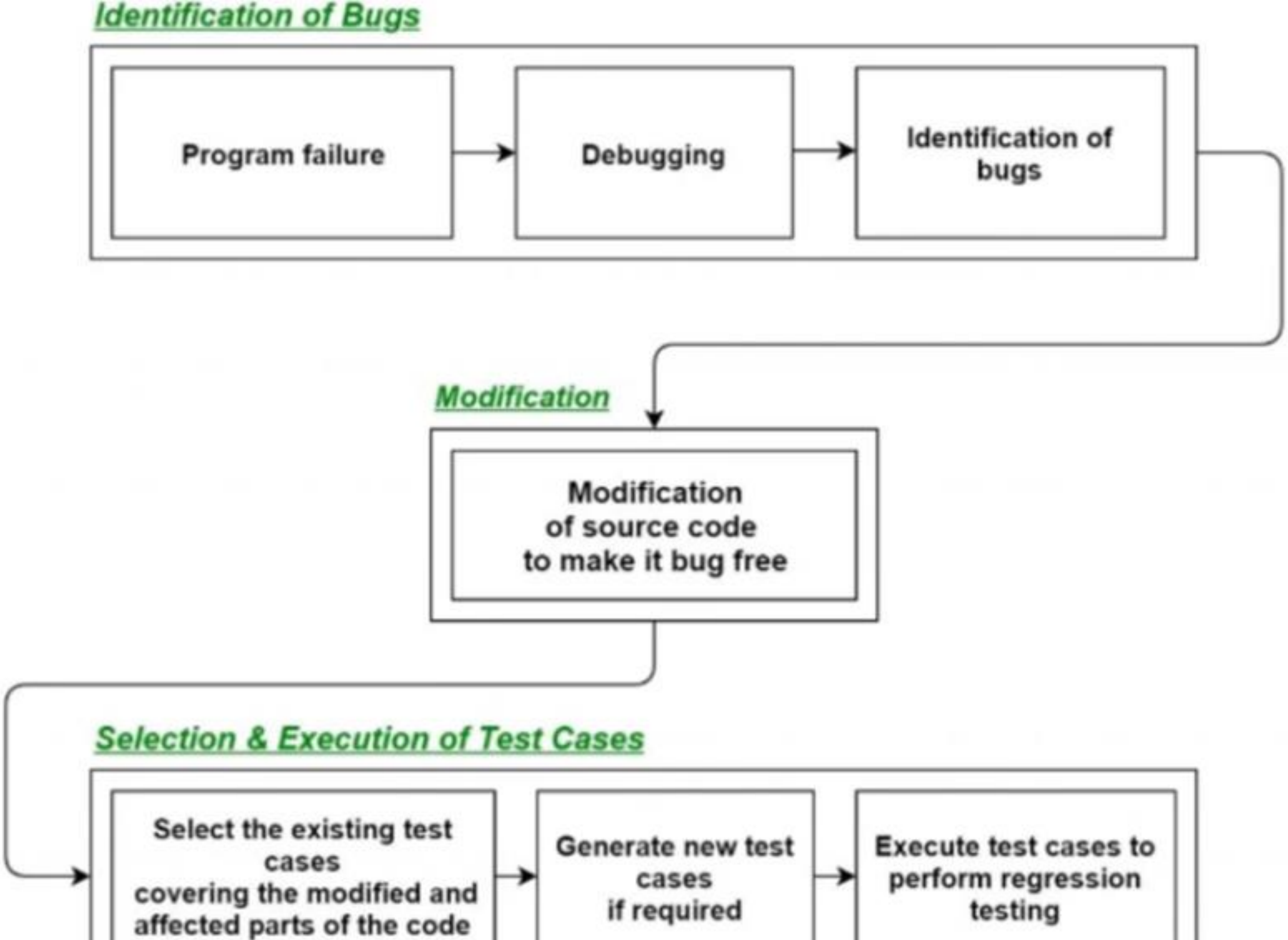
Regression Testing



Regression Testing Cycle




Developed Software
Execute unit-level regression tests
Integrate with the builds
Conduct smoke test to check the build
Perform sanity testing
Perform rigorous integration testing
Regression tests are scheduled
Analyse & report the results

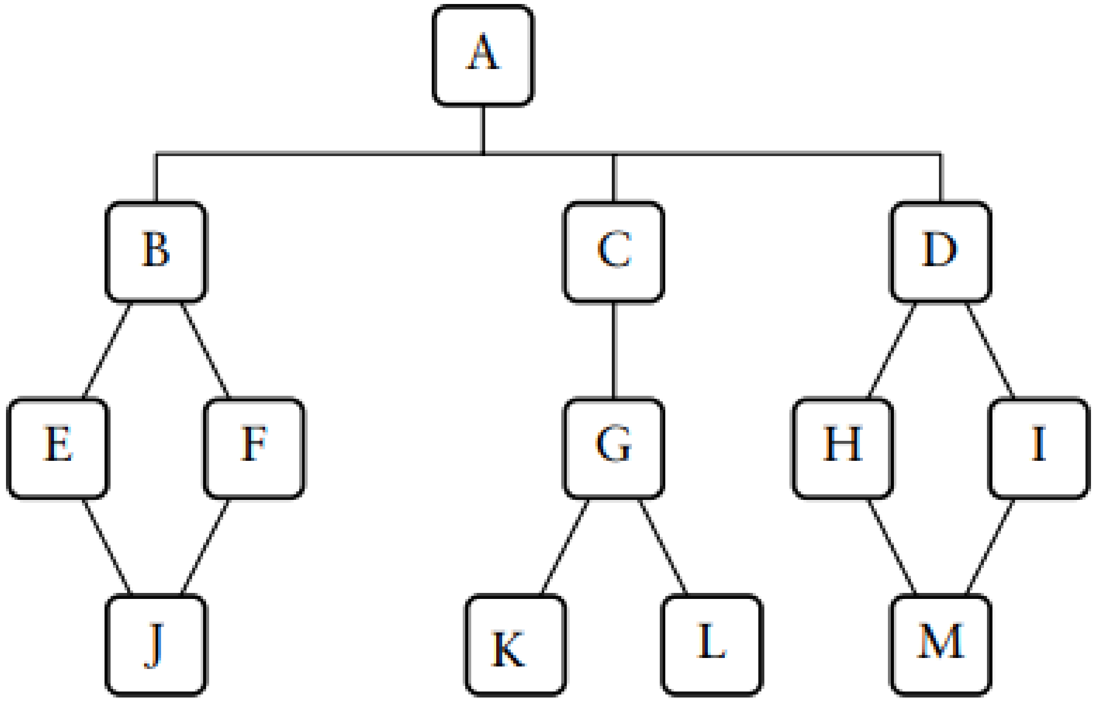

Identification of Bugs
Program failure
Debugging
Identification of bugs
Modification
Modification of source code to make it bug free
Selection & Execution of Test Cases
Select the existing test cases covering the modified and affected parts of the code
Generate new test cases if required
Execute test cases to perform regression testing

Practice Problems

From your experience with online and/or catalog shopping, develop a use case to describe a user purchase of a television set with a credit card from a online vendor using web-based software. With the aid of your use case, design a set of tests you could use during system test to evaluate the software.

Suppose you were developing a stub that emulates a module that passes back a hash value when passed a name. What are the levels of functionality you could implement for this stub? What factors could influence your choice of levels?

Using the structure chart shown below, show the order of module integration for the top-down (depth and breadth first), and bottom-up integration approaches. Estimate the number of drivers and stubs needed for each approach. Specify integration testing activities that can be done in parallel, assuming you have a maximum of three testers. Based on resource needs and the ability to carry out parallel testing activities, which approach would you select for this system and why?


A
B
C
D
E
F
G
H
I
J
K
L
M