Software Testing

Lecture 12

Test Cases

Test Case

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. OR

A test case is a document, which has a set of test data, preconditions, expected results and postconditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test Case acts as the starting point for the test execution, and after applying a set of input values, the application has a definitive outcome and leaves the system at some end point or also known as execution postcondition.

Why we write test cases?

The basic objective of writing test cases is to validate the testing coverage of the application.

Test Coverage : Test coverage measures the amount of testing performed by a set of test. It will include gathering information about which parts of a program are actually executed.

Test coverage is a measure of testing and to have better test coverage it is required to write all the tested cases for identified scenario. Ideally the following process will be in action.

Customer will give requirement to developer.

Test engineer has to spend enough time to understand the requirement

Test engineer has to identify all possible scenarios

Test engineer has to write test cases for all identified scenarios.

Test Case Format

Test case ID:	The Unique_ID of the test case
Test case description:	The objective or summary of the test case
Preconditions:	Any preconditions which need to be executed before starting the test
Test steps:	Procedure to execute the test.
Test data:	Data required while executing the test
Expected Result:	The expected output of the test
Actual Result:	The actual output of the test
Test Case Status:	Pass, Fail, 'Not executed' when test case is not executed and 'Blocked' when high severity bug is found
Created By:	Name of the person who wrote the test case
Date of creation:	The date on which the test case was authored
Executed By:	Name of the person who ran or executed the test case
Date of execution:	The date on which the test case was executed

Test Case Example's

Test Case ID	TC001
Test Case Name	To Verify the "Login" of Gmail account
Preconditions	1.User is authorized. 2. Has an account in Gmail
Test Steps	1.Enter Valid username 2.Enter valid password 3.Click on "Login" button
Test Data	xyz123@gmail.com
Expected Result	1.User should be able to login his Gmail account with his valid credentials 2. "Invalid username or password" should get displayed if the username and password are not valid
Actual Result	1.If the valid credentials are entered then the user will be able to login his/her account 2.If the invalid credentials are entered then nothing happens (the expected message not displayed)
Status	Fail
Created By	John
Date of Creation	01/01/2018
Executed By	Jane
Date of Execution	02/10/2018
Test Environment	•OS: Windows Y •Browser: Chrome N

Test Case Example's

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/ Fail
TC01	Check Customer Login with valid Data	1.Go to site http://faculty.comsats.edu.pk 2.Enter UserId 3.Enter Password 4.Click Submit	Userid = test123 Password = pass123	User should Login into application	As Expected	Pass
TC02	Check Customer Login with invalid Data	1.Go to site http://faculty.comsats.edu.pk 2.Enter UserId 3.Enter Password 4.Click Submit	Userid = test123 Password = pass123	User should not Login into application	As Expected	Pass

How to write good Test Cases?

Customer Requirement: Test cases should be created by keeping the customer requirements in mind.

Straightforward, Simple and Clear: Test cases should be very clear. It should be very straightforward. The number of times it gets executed no matter by whom, it should give the same output. Preferably use simple and clear language while writing the test cases like enter username, click on login button, navigate to home page, etc.

Do NOT presume: Do NOT make any guesses of any functionality or feature of your application. Always author the test cases as per the requirement specification document.

How to write good test Cases?

Unique test cases: Each of the test cases should have a unique name this helps in classifying the test cases while bug tracking or reviewing any requirement at later stage.

Should NOT be repeated: The test cases authored should not be repeated. If any of the test case requires to execute the same steps of other test case then instead of writing it again it's always good to call that test case by its ID in the prerequisites column.

Assure 100% test coverage: Ensure that all the customer requirements are met while authoring the test case. As per the customer specification all the conditions are covered.

How to write good test Cases?

Implementation of Testing Techniques: While writing any test case it is not possible to cover all the conditions of your software application. With the help of testing techniques we can find few test cases where the chances of finding bugs are more.

- **Boundary Value Analysis (BVA):** As the name suggests it's the technique that defines the testing of boundaries for specified range of values.

- **Equivalence Partition (EP):** This technique partitions the range into equal parts/groups that tend to have the same behavior.

- **State Transition Technique**: This method is used when software behavior changes from one state to another following particular action.

- **Error Guessing Technique:** This is guessing/anticipating the error that may arise while testing. This is not a formal method and takes advantages of a tester's experience with the application

Peer Review of Test Cases: Test cases should always be reviewed by peers. If any precondition or any condition is missed while authoring the test case then it can be covered as per the peer's feedback.

Test Cases Advantages

1. **Better Consistency :** When test cases are written for a test execution, the test engineer work will be organized better and simplified. Let's consider a scenario as an example**.**

-> When a release1.0 is being tested using existing test cases, the same test cases can be quickly executed for the next release 1.0.1, having test cases helps to identify issues which impacting the old release, this way we can achieve better consistency in test execution.

2. **Avoid training :** To avoid training of every new test engineer on the product or requirement. Consider the following situation.

Test Engineer write test cases.

Test engineer quits the job

New test engineer joins

It takes lot of effort to explain every module and it takes lot of time for new tester to understand the complete requirement.

But if we write test cases in the beginning, then the newly joined test engineer can test application by looking in to test cases on his own.

Here its not necessary to train him on requirement because, if he execute test cases for couple of release on test cycles he can understand the product.

Test Cases Advantages

3. No dependency : To depend on process rather than person. Every company will set their own process.

When we get the requirement test engineer should do follow the process.

Understand requirement

identify all scenarios and document.

Prepare a test cases for each scenarios and document.

This process of documenting each and everything will remove the dependency on a person

4. Writing test cases will keep track of all the steps for a particular scenario, this is very important because complicated steps to test a scenario might be difficult to remember always.

Test Cases Advantages

5. Test cases will ensure that complete functional and non functional testing will be done and hence test cases will give complete list of all scenarios to meet the software product quality expectations.

6. The main objective of test case is to exercise every flow in application.

7. To ensure that it satisfies the BRS i.e. Business Requirement Specification and SRS i.e. System Requirement Specifications.

8. Test cases will ensure that every functionality working as expected in every possible scenario.

9. Test cases will ensure that all the requirements have been met as per the customer requirement document.

Test Case Disadvantages

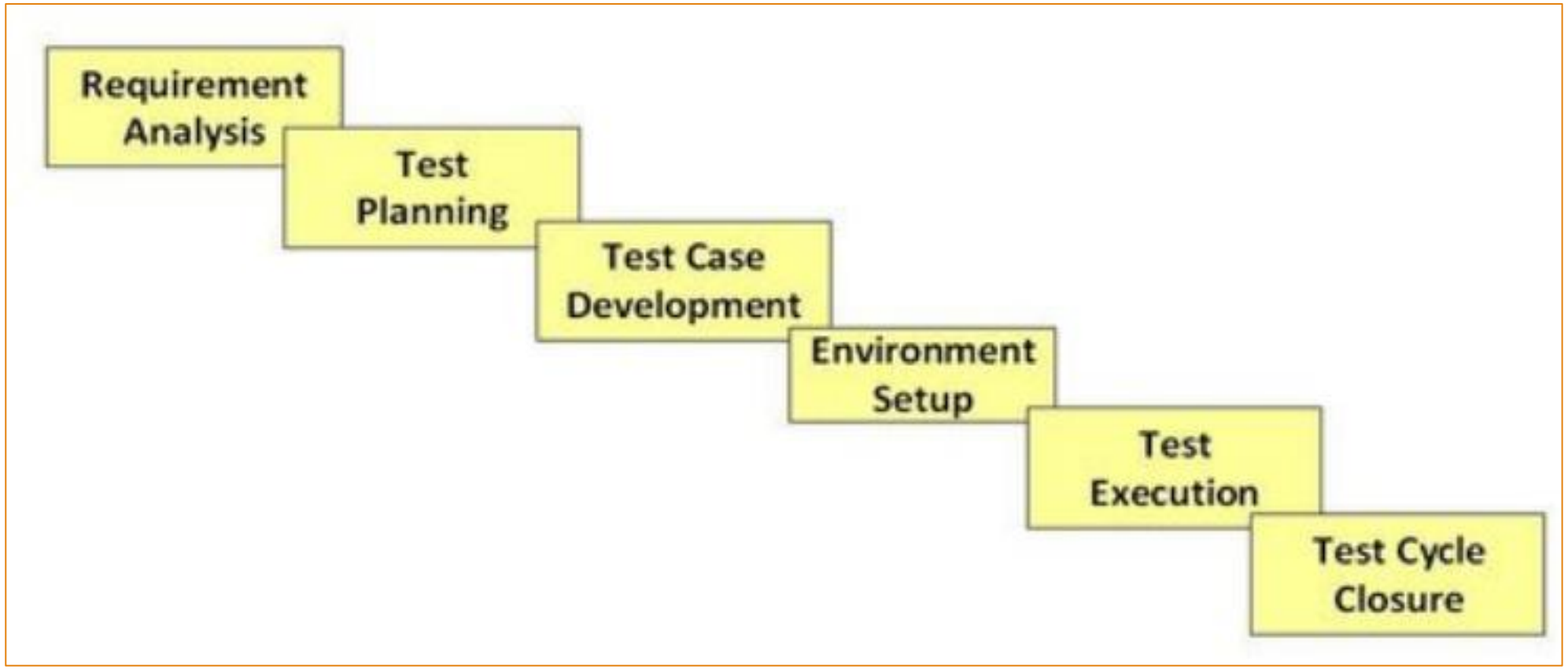
If any existing feature is changed then the related test Cases needs modification which is time consuming as one has to go through the entire list of test cases and find those test cases which requires modification.

If any feature becomes obsolete then the associated test cases should be cleaned.

Questions and Answers

Testing in Software Life Cycle

Software Testing Life Cycle(STLC)



Software Testing Life Cycle

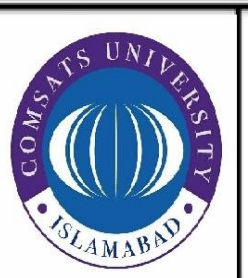
Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing.

It consists of series of activities carried out methodologically to help certify your software product.

Each of these stages have a definite Entry and Exit criteria; , Activities & Deliverables associated with it.

Entry Criteria: Entry Criteria gives the prerequisite items that must be completed before testing can begin.

Exit Criteria: Exit Criteria defines the items that must be completed before testing can be concluded



Software Testing

Lecture 13

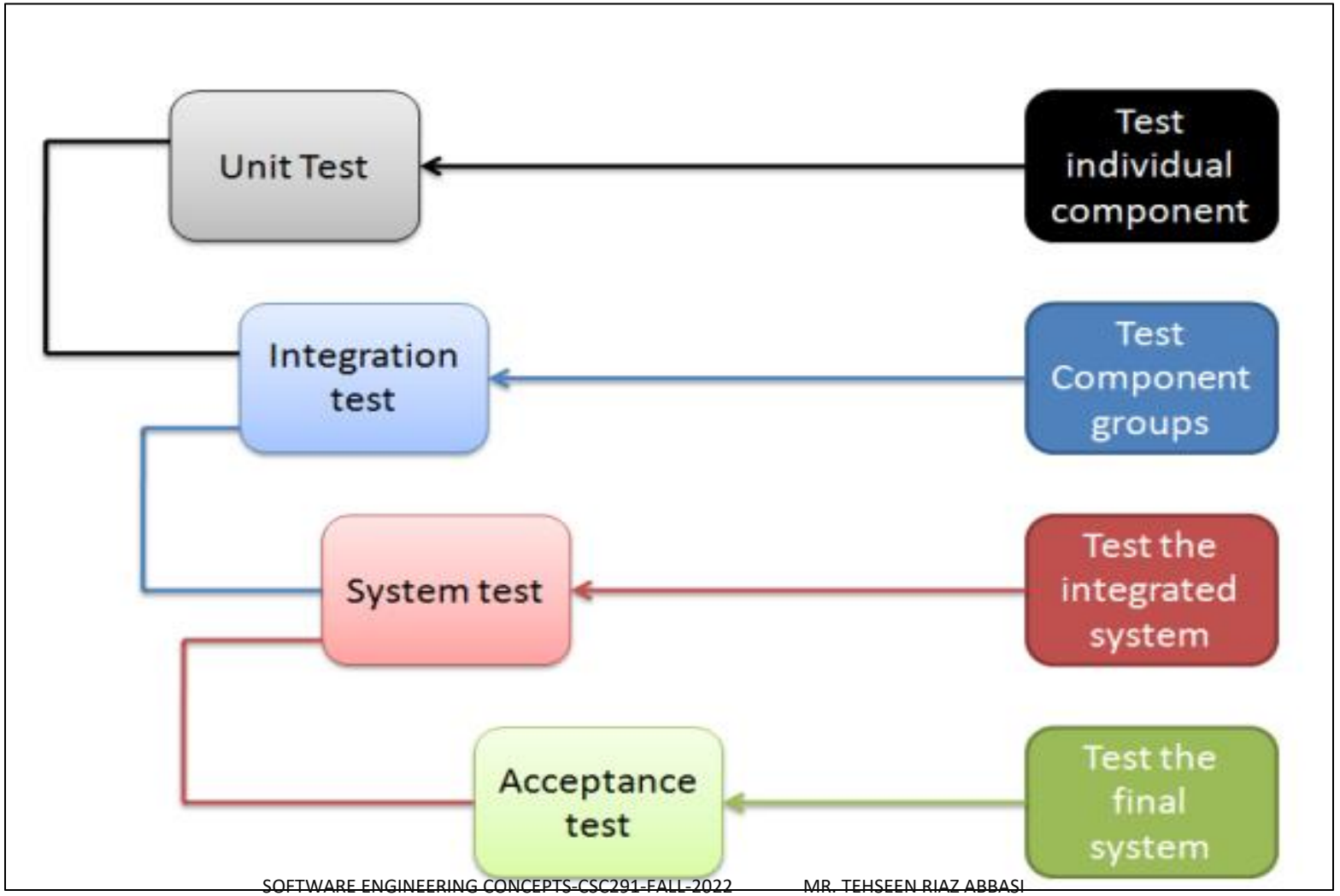
General Levels of Testing

Chapter 3: Testing in Software Life Cycle

Agenda for today

- Levels of Testing
- Component Testing
- Stub's and Drivers
- Unit Testing
- Points for Unit Testing
- Tools for Unit Testing
- Integration Testing
- Integration Testing Approaches
- Bottom Up and Top Down integration Testing
- BigBang and Adhoc integration Testing

Levels of Testing



Component Testing

- **Testing of separate software component** is known as component testing
- **Objective of component** testing is to verify the input/output behaviour of the test object
- **A group of components is known as module testing.**
- The main characteristic is that the software components are tested individually and isolated from all other software components of the system.
- The isolation is necessary to prevent external influences on components.
- Most often **stubs** and **drivers** are used to replace the missing software and simulate the interface between the software components

Test Objects, Stubs and Driver

- **Test objects** are program modules/units or classes, (database) scripts, and other software components.
- **Drivers** are the dummy programs which are used to call the functions of the lowest module in case the calling function does not exist.

OR

→ A driver calls the component to be tested.

- **Stubs** can be referred to as code a snippet which accepts the inputs/requests from the top module and returns the results/ response.

OR

-> A **stub** is called from the software component to be tested.

Component Testing Objectives

- Reducing risk
- Verifying whether functional and non-functional behaviours of the component are as expected
- Building confidence in the component's quality
- Finding defects in the component
- Preventing defects from escaping to higher test levels

Component Testing: Example

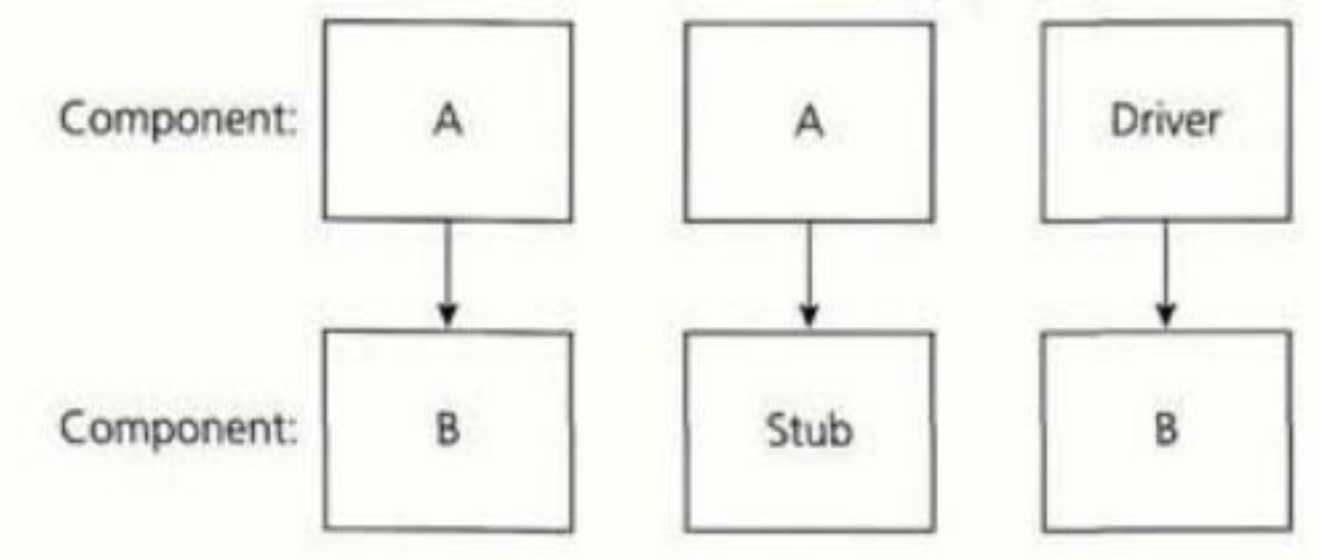
- **Scenario:** There are two web pages. In one of the web pages there are a many certain fields like username, address, mobile no. etc in which data has to be entered. In the other (second) web page also there are certain fields which carry forward the data from the first page. Testing the functionality of these individual pages is called Component Testing.
- **Components are tested as soon as they are created,**
- There can be results retrieved from a component under test, are dependent on other components which in turn are not developed
- So in order to test that component, we use **Stubs** and **Drivers**

Stubs and Driver's Difference

	Stub	Driver
Type	Dummy codes	Dummy codes
Description	A piece of code that emulates the called function.	A piece of code that emulates a calling function.
Used in	Top Down Integration	Bottom Up Integration
Purpose	To allow testing of the upper levels of the code, when the lower levels of the code are not yet developed.	To allow testing of the lower levels of the code, when the upper levels of the code are not yet developed.

Stub and Driver Example 1:

- **Suppose Functio**n A that calculates the total marks obtained by a student in a particular academic year.
- **Suppose this function derives** its values from another function (Function b) which calculates the marks obtained in a particular subject.



Example 2

- **We have 3 modules login, home, and user module in website**.
- Login module is ready and need to test it, but we call functions from home and user (which is not ready)
- **What we will do?**
- **Stub**
- For the same example if we have Home and User modules get ready and Login module is not ready.
- **Driver**

Unit Testing

- Check individual parts functionality
- Aim is to isolate each unit of the system to identify, analyze and fix the defects
- Done by developers
- Reduces cost of testing
- Developers can also re-use code
- Facilitates changes
- Example: Testing a function: whether the loop or program is working properly or not

Unit Testing Example

Almost every web application requires its users/customers to log in. For that, every application has to have a "Login" page which has these elements:

- **Account/Username**
- **Password**
- **Login/Sign in Button**

- For unit testing what which test cases will be needed?

Points for Unit Testing

- **Functional :**

  - Does code functionally perform the task?

- **Boundaries :**

  - What are the minimum, maximum values for the function?

- **Termination:**

  - What happens in the normal termination and abnormal termination?

- **Outputs:**

  - What are the expected outputs of the function?

- **Inputs:**

  - What are the expected inputs to the function?

- **Interaction:** - What other modules/functions does this interact with?

Unit Testing Tools

- Junit
- Nunit
- JMocKit
- EMMA
- PHPUnit

Integration Testing

- **Integration testing** is carried out when integrating (i.e., combining):
  - Units or modules to form a component
  - Components to form a product
  - Products to form a system
- Multiple modules & these are developed by different developers.
- Find interface defects between the modules/functions
- Checks connectivity or data transfers
- Developer's or tester's performs this type of testing.
- **Example:** Battery and sim card are integrated i.e. assembled in order to start the mobile phone.
- **e.g.** Computer and keyboard

Integration Testing Example

- Online shopping website
- One developer was assigned to develop each of the modules below.
  - User registration and Authentication/Login
  - Product Catalogue
  - Shopping Cart
  - Billing
  - Payment gateway integration
  - Shipping and Package Tracking
- The QA Manager suggested that integration testing should be performed.
- Which types of bugs found during integration Testing?

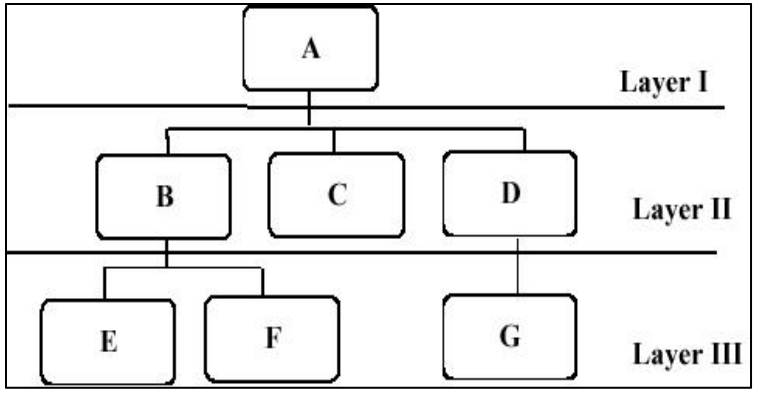
Integration Testing Approaches

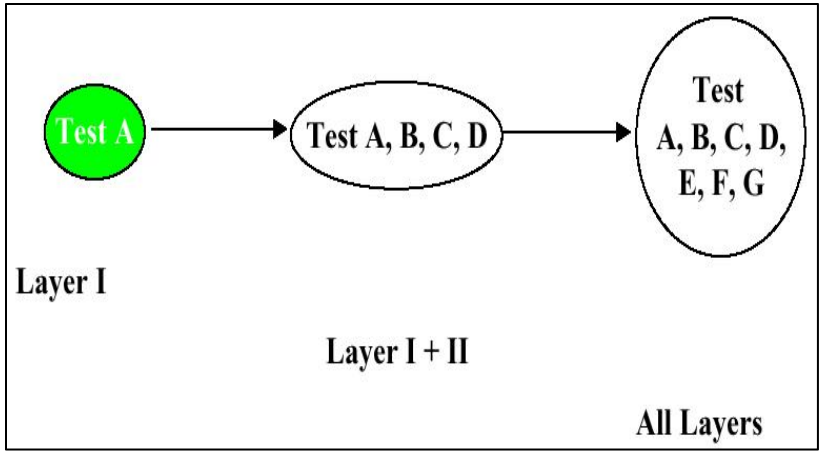
Mainly four approaches:

- Top Down approach
- Bottom up approach
- BigBang approach
- Adhoc approach

Top-down Integration Testing

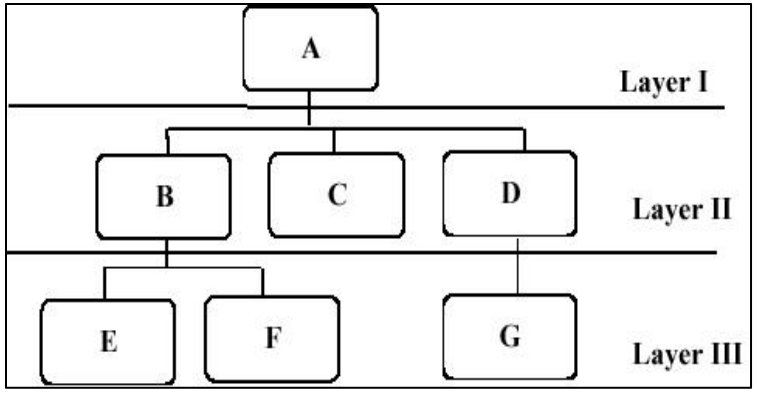
- Testing the top-most modules
- Gradually moving down to the lowest set of modules one-by-one
- **Stubs** used when lower modules are not ready
- Critical Modules are tested on priority

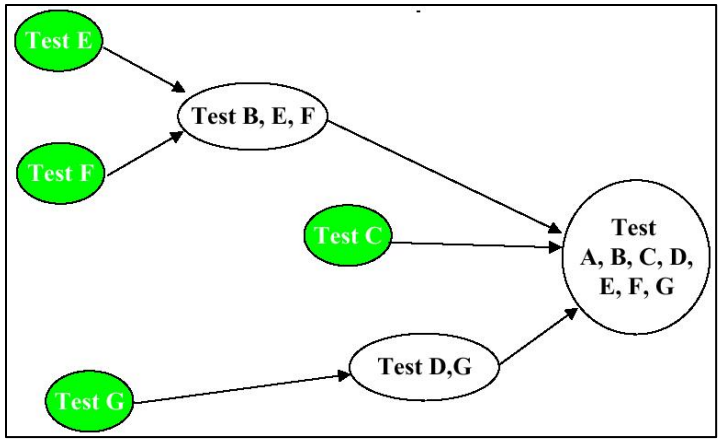




Bottom up Integration Testing

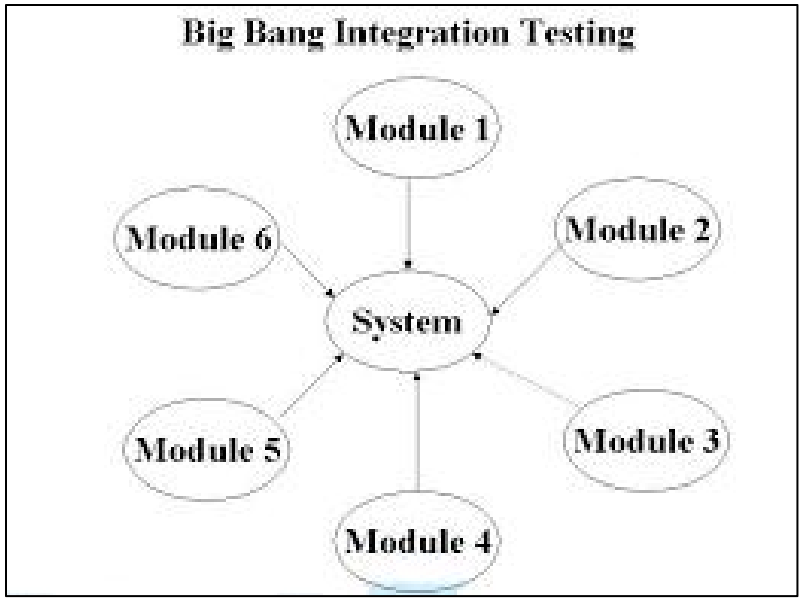
- Starts with testing the lowest units of the application
- Gradually moving up one-by-one.
- **Drivers** will be used to simulate the functionality of the missing modules
- Development and testing can be done together





Big Bang Integration Testing

- **All components tested atleast once**
- Individual modules of the programs are not integrated until every thing is ready.
- Convenient for small systems
- Saves time



Ad hoc Integration Testing

- **Testing performed** without proper planning and documentation
- **The components** are being integrated in the (casual) order in which they are finished.
- **Saves time** because every component is integrated as early as possible into its environment.
- **Used for limited time** to do exhaustive testing

Software Testing

Lecture 14

Requirement Traceability Matrix (RTM)

RTM captures all requirements proposed by the client or software development team and their traceability in a single document delivered at the conclusion of the life-cycle.

In other words, it is a document that maps and traces user requirement with test cases.

The main purpose of RTM is to see that all test cases are covered so that no functionality should miss while doing Software testing.

Why RTM is Important?

The main agenda of every tester should be to understand the client's requirement and make sure that the output product should be defect-free. To achieve this goal, every QA should understand the requirement thoroughly and create positive and negative test cases.

RTM Example

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

Other parameters in RTM

Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

RTM Can

Show the requirement coverage in the number of test cases

Design status as well as execution status for the specific test case

If there is any User Acceptance Test to be done by the users, then UAT status can also be captured in the same matrix.

The related defects and the current state can also be mentioned in the same matrix.

STLC Phases – 1- Requirement Analysis

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Requirement Analysis	Requirements Document available (both functional and non functional) Acceptance criteria defined. Application architectural document available.	Analyze business functionality to know the business modules and module specific functionalities. Identify all transactions in the modules. Identify all the user profiles. Gather user interface/ authentication, geographic spread requirements. Identify types of tests to be performed. Gather details about testing priorities and focus. Prepare Requirement Traceability Matrix (RTM). Identify test environment details where testing is supposed to be carried out. Automation feasibility analysis (if required).	Signed off RTM Test automation feasibility report signed off by the client	RTM Automation feasibility report (if applicable

STLC Phases – 2- Test Case Planning

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Test Planning	Requirements Documents Requirement Traceability matrix. Test automation feasibility document.	Analyze various testing approaches available Finalize on the best-suited approach Preparation of test plan/strategy document for various types of testing Test tool selection Test effort estimation Resource planning and determining roles and responsibilities.	Approved test plan/strategy document. Effort estimation document signed off.	Test plan/strategy document. Effort estimation document.

STLC Phases – 3- Test case development

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Test case development	Requirements Documents RTM and test plan Automation analysis report	Create test cases, test design, automation scripts (where applicable) Review and baseline test cases and scripts Create test data	Reviewed and signed test Cases/scripts Reviewed and signed test data	Test cases/scripts Test data

STLC Phases – 4- Test Environment setup

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Test Environment setup	System Design and architecture documents are available Environment set-up plan is available	Understand the required architecture, environment set-up Prepare hardware and software development requirement list Finalize connectivity requirements Prepare environment setup checklist Setup test Environment and test data Perform smoke test on the build Accept/reject the build depending on smoke test result	Environment setup is working as per the plan and checklist Test data setup is complete Smoke test is successful	Environment ready with test data set up Smoke Test Results.

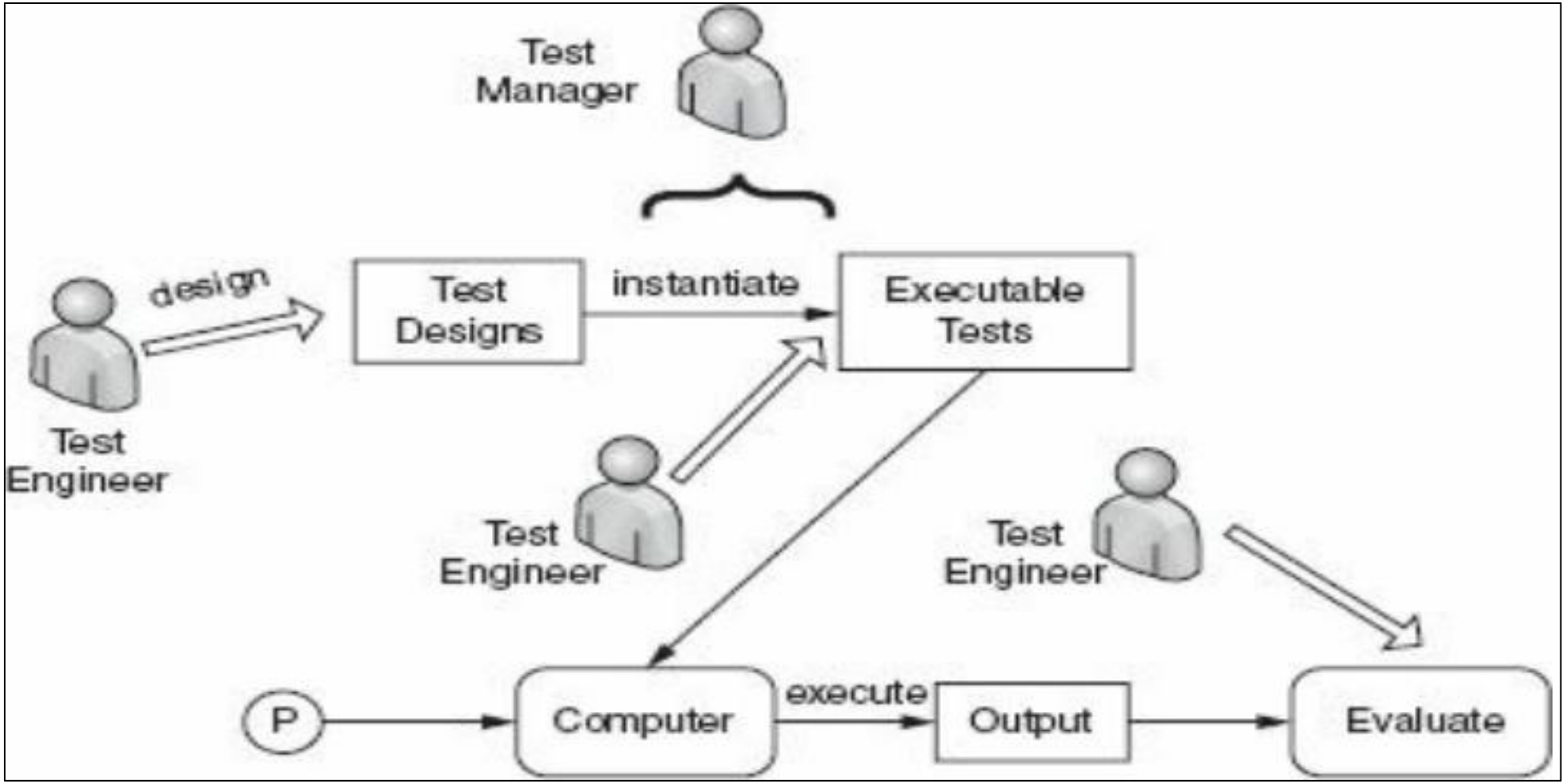
STLC Phases – 5- Test Execution

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Test Execution	Baselined RTM, Test Plan , Test case/scripts are available Test environment is ready Test data set up is done Unit/Integration test report for the build to be tested is available	Execute tests as per plan Document test results, and log defects for failed cases Update test plans/test cases, if necessary Map defects to test cases in RTM Retest the defect fixes Regression Testing of application Track the defects to closure	All tests planned are executed Defects logged and tracked to closure	Completed RTM with execution status Test cases updated with results Defect reports

STLC Phases – 6- Test Closure

STLC Stage	Entry Criteria	Activity	Exit Criteria	Deliverables
Test Cycle closure	Testing has been completed Test results are available Defect logs are available	Evaluate cycle completion criteria based on - Time, Test coverage, Cost, Software Quality, Critical Business Objectives Prepare test metrics based on the above parameters. Document the learning out of the project Prepare Test closure report Qualitative and quantitative reporting of quality of the work product to the customer. Test result analysis to find out the defect distribution by type and severity	Test Closure report signed off by client	Test Closure report Test metrics

Software Testing Activities



V Model

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape.

Each phase must be completed before the next phase begins.

Testing of the phase is planned in parallel with corresponding phase of development in V model.

V Model

The V-Model is a unique, linear development methodology used during a SDLC.

V model is also known as Verification and Validation model.

Testing phase goes in parallel with the development phase.

Testing phase starts right at the beginning of SDLC.

V-Model

Developers Life Cycle (Verification)

Tester's Life Cycle (Validation)

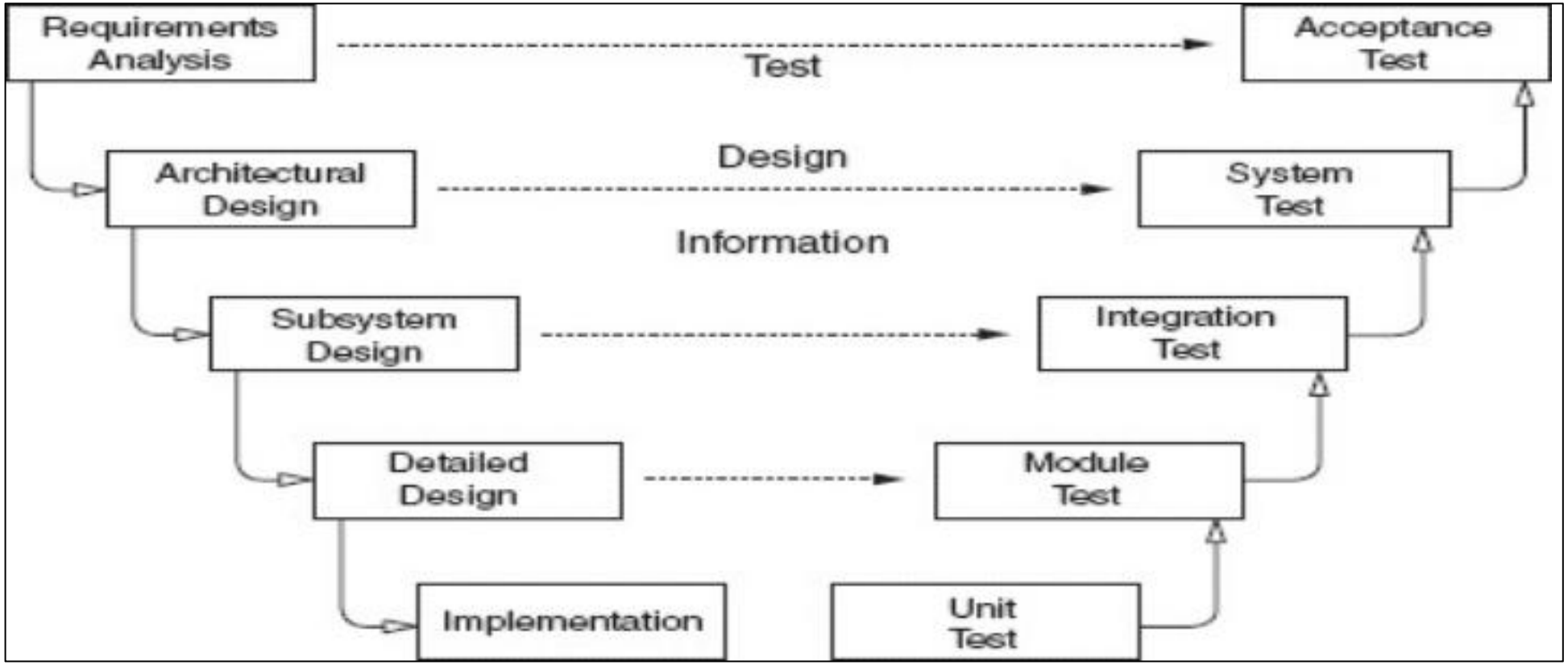


Fig. V-Model

Verification & Validation

Verification	Validation
Verification addresses the concern: "Are you building it right?“ or if we are in the right track of creating the final product.	Validation addresses the concern: "Are you building the right thing?“
Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the Proposed behavior.
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.
Done by developers.	Done by testers.
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.

V Model

Typical V-model shows Software Development activities on the Left hand side of model and the Right hand side of the model actual Testing Phases can be performed.

In this process “Do-Procedure” would be followed by the developer team and the “Check-Procedure” would be followed by the testing team to meets the mentioned requirements.

The V-model typically consist of the following phases:

1. **Acceptance Testing** : assess software with respect to requirements or users’ needs.
2. **System Testing** : assess software with respect to architectural design and overall behaviour.
3. **Integration Testing** : assess software with respect to subsystem design.
4. **Module Testing**: assess software with respect to detailed design.
5. **Unit Testing** : assess software with respect to implementation.

V-Model

Requirement Analysis:

The requirements analysis phase of software development captures the customer's needs.

Acceptance testing is designed to determine whether the completed

software in fact meets these needs.

Acceptance testing probes whether the software does what the users want.

Architectural Design

This step maps requirements onto functions and dialogues of the new system.

It's the phase where system engineers analyze and understand the business of the proposed system by studying the user requirements document.

They figure out possibilities and techniques by which the user requirements can be implemented. If any of the requirements are not feasible, the user is informed of the issue.

V-Model

Architectural Design:

System testing is designed to determine whether the assembled system meets its specifications.

It assumes that the pieces work individually and asks if the system works as a whole.

This level of testing usually looks for design and specification problems.

Subsystem Design:

The subsystem design phase of software development specifies the structure and behavior of subsystems, each of which is intended to satisfy some function in the overall architecture. Often, the subsystems are adaptations of previously developed software.

Integration testing is designed to assess whether the interfaces between modules (defined below) in a subsystem have consistent assumptions and communicate correctly.

Integration testing must assume that modules work correctly. Some testing literature uses the terms integration testing and system testing interchangeably.

V-Model

Detailed Design:

The detailed design phase of software development determines the structure and behavior of individual modules. A module is a collection of related units that are assembled in a file, package, or class.

Module testing is designed to assess individual modules in isolation, including how the component units interact with each other and their associated data structures.

Most software development organizations make module testing the responsibility of the programmer; hence the common term developer testing.

Implementation:

It's a phase of software development that actually produces code. A program unit, or procedure, is one or more contiguous program statements, with a name that other parts of the software use to call it.

Unit testing is designed to assess the units produced by the implementation phase and is the "lowest" level of testing.

V Model Basic Idea/Characteristics

Implementation and testing activities are separated but are equally important (left side / right side).

The V illustrates the testing aspects of verification and validation.

We distinguish between different test levels, where each test level is testing “against” its corresponding development level.

V-Model Advantages

Each phase of development is tested before moving to next phase, hence there is higher rate of success.

It avoids down word flow of defects because each phase is verified explicitly.

The model has clear and defined steps. So, it is easier to implement.

It is suitable for smaller projects where requirements are easily understood and fixed.

It saves a lot of time.

Disadvantages of V-Model

The testing team starts in parallel with development. Hence, the overall budget and resource usage increases.

Change in requirement are difficult to incorporate. Because if any changes happens, then the test requirement document must be updated.

Guessing the error in the beginning of the project could take more time.

The working model of the software is only available in the later phases of the development.

It is not suitable for complex and large applications because of its rigid process.

When to use the V-model?

The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.

V Model should be followed for the project where very less probability to make the changes in the middle of testing or development phase which are unplanned.

The V-Shaped model should be chosen when sufficient technical resources are available with needed technical expertise.

High confidence of customer is required for choosing the V-Shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations.


Thank
you