Course Title : **SOFTWARE ENGINEERING**

Course Code : **315323**

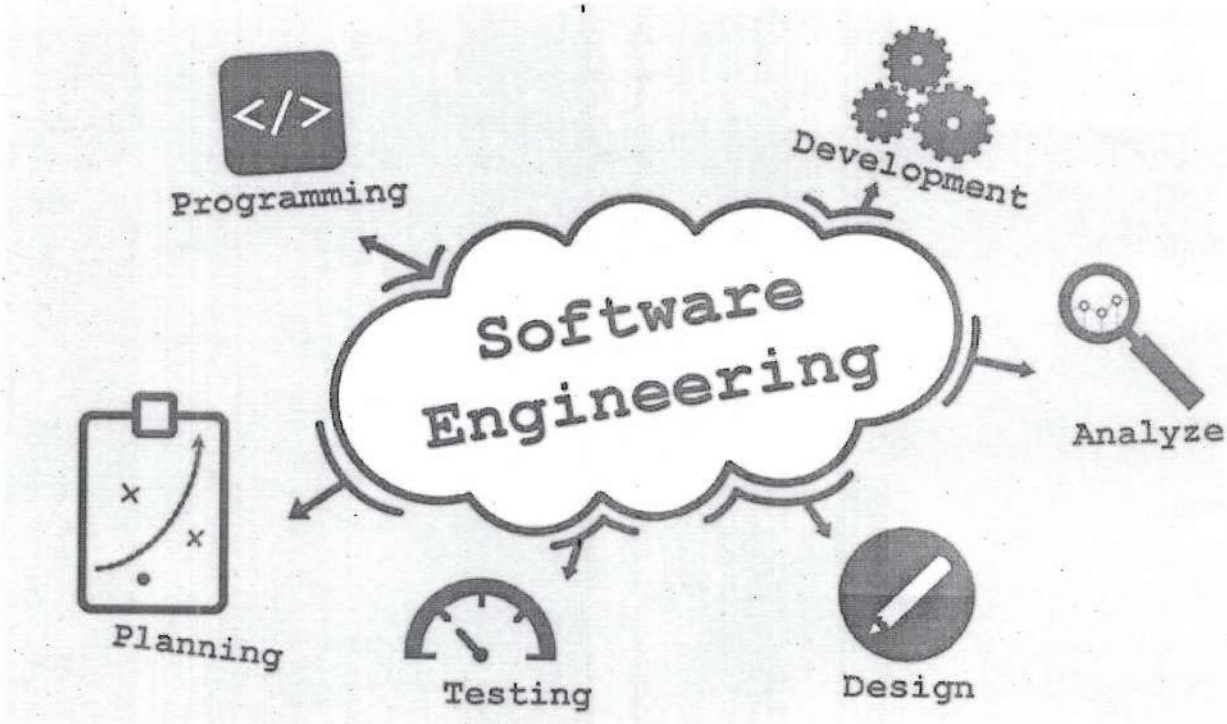
Programme Name : **Computer Engineering**

Semester : **Fifth**

Unit - III Software Modelling and Design

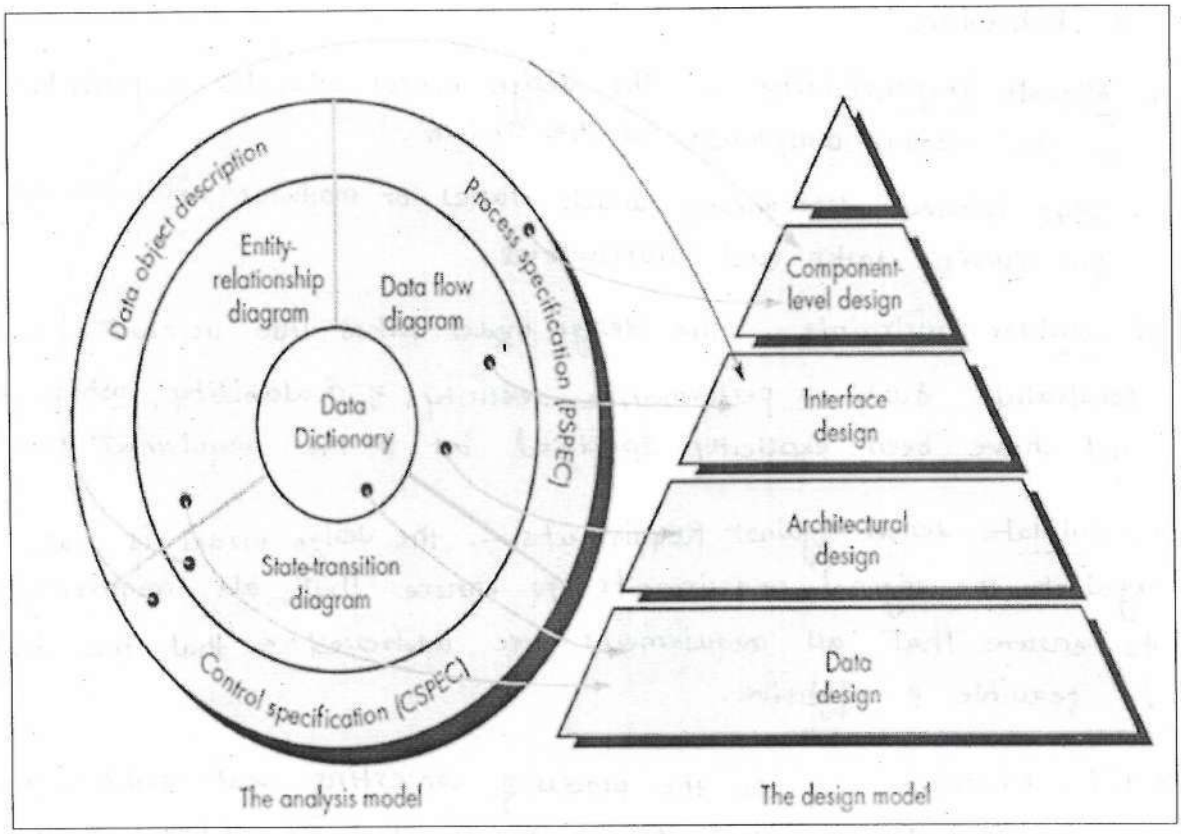
Total Marks: 16

Prepared By: **Mr.Yogeshwar L Patil**



UNIT-3 Software Modelling and Design marks-16

★ ★ Translating Requirment model into design model: Data modelling.



- Translating requirment model into a design model, specifically for data modelling involves transforming functional & non-functional requirments into detailed blueprint for the systems data structures which then guides implementation.
- This process typically involves several steps -

1. Identify Design Elements – First step is to identify the design elements that will fulfill the requirments.
   - This includes identifying the classes, interfaces, modules and other components that will make up the system.
2. Refine Requirments – the high-level requirments from the requirment model are refined & expanded upon to create more detailed design requirments - this may involves breaking down complex requirments into smaller, more manageable components.

3. create Design Diagrams — Design diagrams such as class diagrams sequence diagram & State diagrams & al are used to visually represent the design elements and their relationships
- this diagrams provide a blueprint for the systems architecture & behavior.

4. Allocate Responsibility — the design model allocates responsibilities to the various components of the system.
- this involves determining which classes or modules will be responsible for specific tasks and interactions.

5. consider constraints — the design model takes into account various constraints such as performance, security, & scalability, which may not have been explicitly specified int in the requirmet model

6. validate design Against Requirments — the design model is validated against the orginal requirments to ensure that all requirments to ensure that all requirments are addressed & that the design is feasible & effective.

* Data Modelling — It is the process of converting real-world requirments into structured format that can be stored in a database It is key step when translating requirmet model into a design model.
- It defines what data should be stored, How data is related, & how data is organized.
- It act as blueprint for designing a database.

*Types of Data modelling —
1. conceptual model — the main aim of this model is to establish the entities, their attributes & their relationships.
- this data model defines WHAT the system contains.

2. logical model — Defines the HOW the system should be implemented regardless of the DBMS.
- It defines the structure of data elements & set of relationship betn them

3. physical model — It describes HOW the system will be implemented wing specific DBMS system.
- It describes specific implementation of the data model.

* Analysis Modeling -

- Analysis model is technical representation of the system.
- It act as link between the system description and the design model
- In analysis modeling, information, behavior and functions of the system are defined and translated into the architecture, component and interface level design in design modeling.
- It is the technical representation of the systems requirments, translating customer needs into structured format of diagrams & text to depict the information, functions & behavior of the software to be built.

- Purpose & Goals -

- converts user requirments into a structured, understandable format.
- connects the system level functionality with detailed software design.
- provides clear, graphical view of the system for both technical & non-technical Stakeholders.
- Establishes a basic for validating the softwares' requirments once it is built.

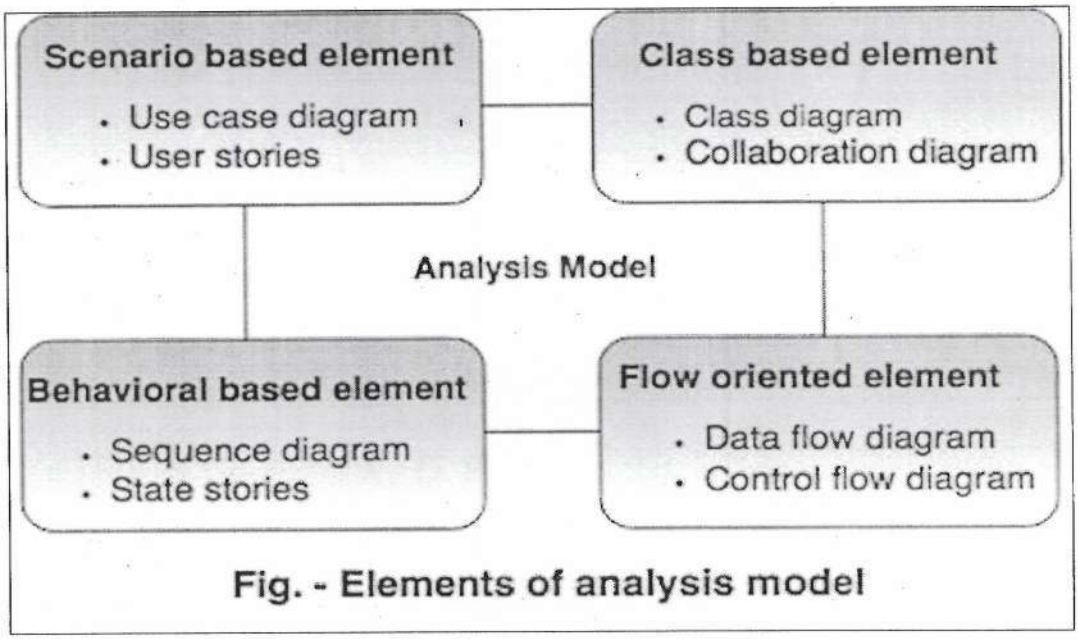


Fig. - Elements of analysis model

1. Scenario Based element –

- This type of elements represents the system user point of view
- scenario based elements are use case diagram, user stories.

2. Class Based element –

- the object of this type of element manipulated by the system.
- It defines the object, attributes and relationship
- the collaboration is occuring between classes.
- class based elements are the class diagrams, collaboration diagram

3. Behavioral element –

- Behavioral element represent state of the system & how it is changed by the external events.
- the behavioral elements are sequence diagram, State diagram

4. Flow - oriented element –

- An information flows through a computer based system it gets transformed.
- It shows how data object are transformed while they flow between the various system functions.
- the flow elements are data flow diagram, control flow diagram.

* Design Modelling —

- Design modelling in software Engineering is the creation of blueprint-like diagrams to represent a software system architecture, user interface & components details, serving as visual guide before coding begins.
- It act as detailed plan for software construction, illustrating how the system will be build.
- provides common visual understanding of the system for developers, designer & Stakeholders.
- Simplified complex software design into understandable diagrams, making them easier to grasp

* Fundamental Design concept —

* Abstraction — Hides complex, unnecessary details and presents a simpler, more understandable interface to the user or other developers.

* Information Hiding —

- Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.
- It means that modules should be specified & designed so that information contained within a module is inaccesible to other modules that have no need of for such information.

* Structure —

- The complete Structure of the software is known as software Architecture
- structure provides conceptual integrity for a system in number of ways
- the architecture is the Structure of program modules where they interact with each other in a specialized way and the components use the Structure of data.

* Modularity -

- Decomposes a system into smaller, independent and manageable components or modules.
- It software is seperately divided into name & addressable components. Sometimes they are called as modules which integrate to satisfy the problem requirments.

- modularity is the single attribute of a software that permits a program to be managed easily.

* concurrency -
- The technique and principles for designing system that execute multiple task or processes simultaneously or apparently simultaneously to improve efficiency, resource utilization & responsiveness.

* Verification -
- verification is the process of checking that software achieves its goal without any bugs.
- It is the process to ensure whether the product that is developed is right or not.
- It verifies whether the developed product fullfills the requirment that we have
- verification is Static Testing.

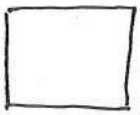
* Aesthetics -
- Aesthetic design in software engineering focuses on creating software interfaces and experiences that are visually pleasing, engaging & intuitive for users.
- the study of beauty & taste is known as Aesthetics
- It refers visual appeal & aspect of a design or an interface
- the study of Aesthetics looks at how people view both natural & artificial sources of experience.

* Design Notations -

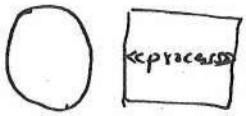
* Data Flow Diagram -(DFD)
- A Data flow diagram shows the way information flows through a process or system.
- It include data inputs & outputs, data Stores & the various subprocesses the data moves through
- It shows how data enters and leaves the system, what changes the information & where the data is Stored

- DFD's are built using Standardized Symbols and Notations to describe various entities and their relationships.

1. External entity :- external entity that stands outside of the system & communicates with the system. It can be organization like bank group of people like customers



2. process - process that changes data, producing an output. It might perform computations. or sort data based on logic
- A short label is used to describe process



3. Data store. - Data is files or repositories that hold information for later use such as database table or membership form

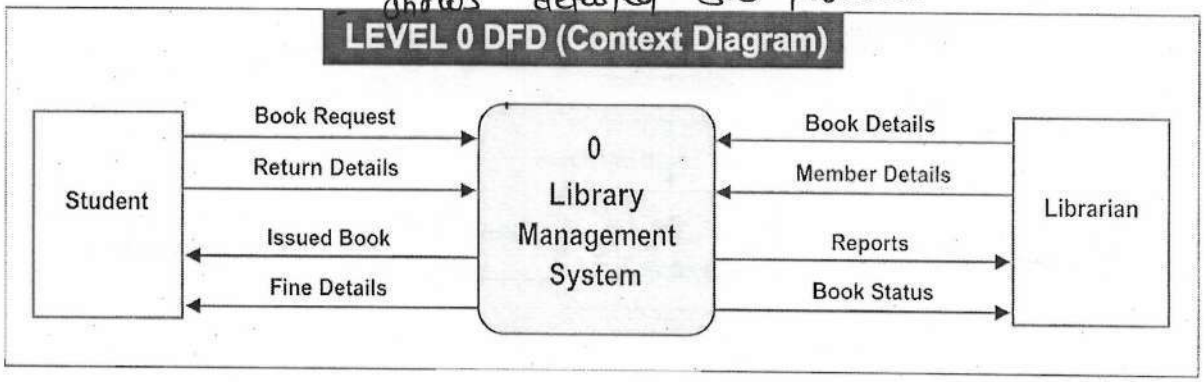
4. Data flow - Data flow describes the information transferring between different parts of the system. the arrow symbol is the symbol of data flow.

* levels of DFD -

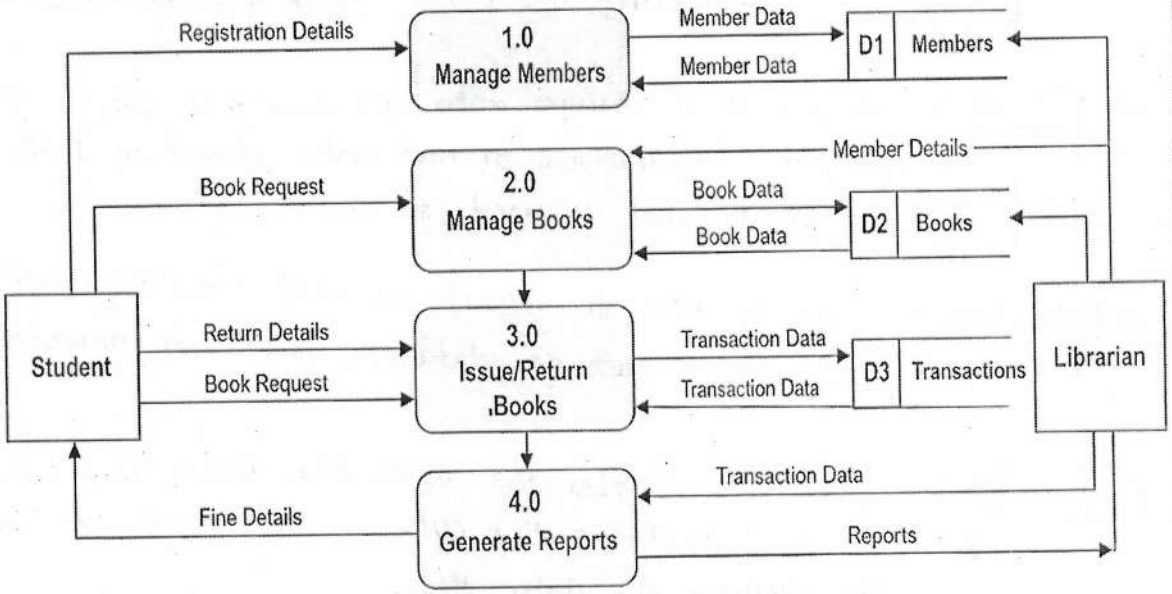
1. context level (level 0 DFD) - Shows entire system as single process & focus on External entities and Input output data.

2. level 1 DFD - It breaks level 0 into main processes
- It shows multiple processes, Data stores & Data flow

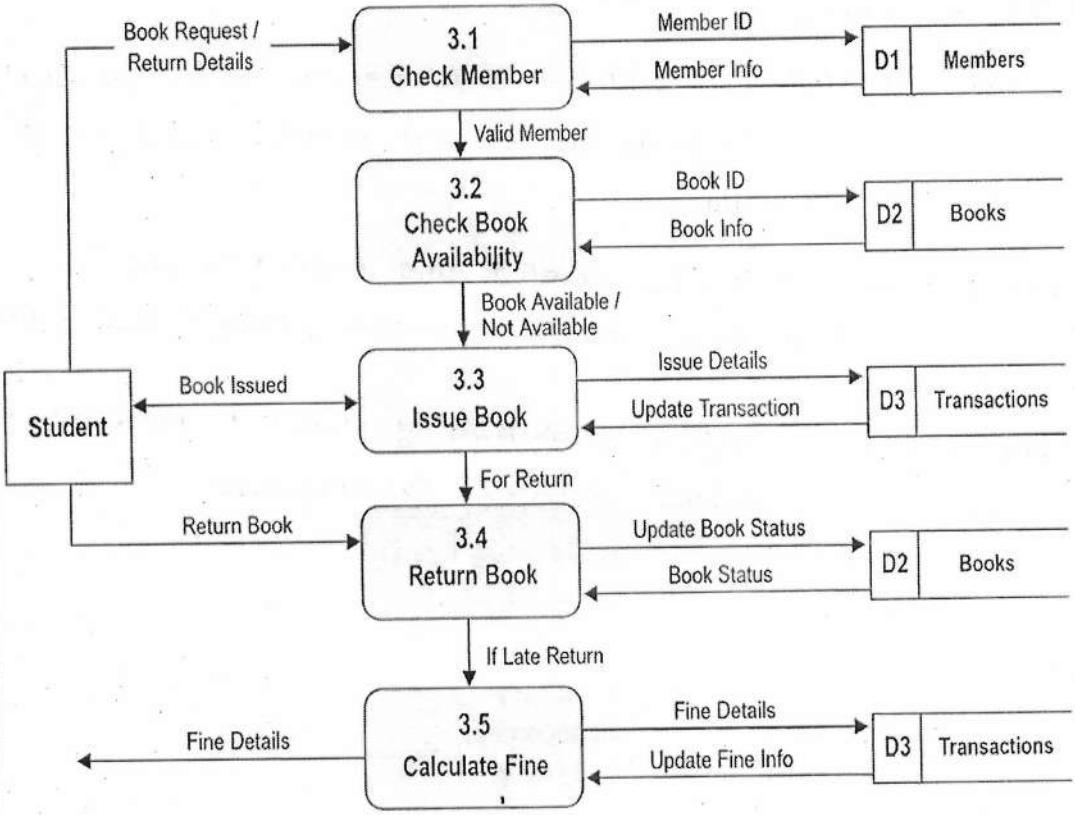
3. level 2 DFD - further breakdown of level 1 processes
shows detailed Sub-processes



LEVEL 1 DFD

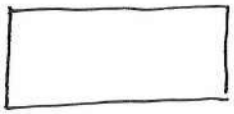
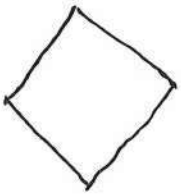
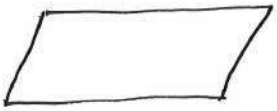


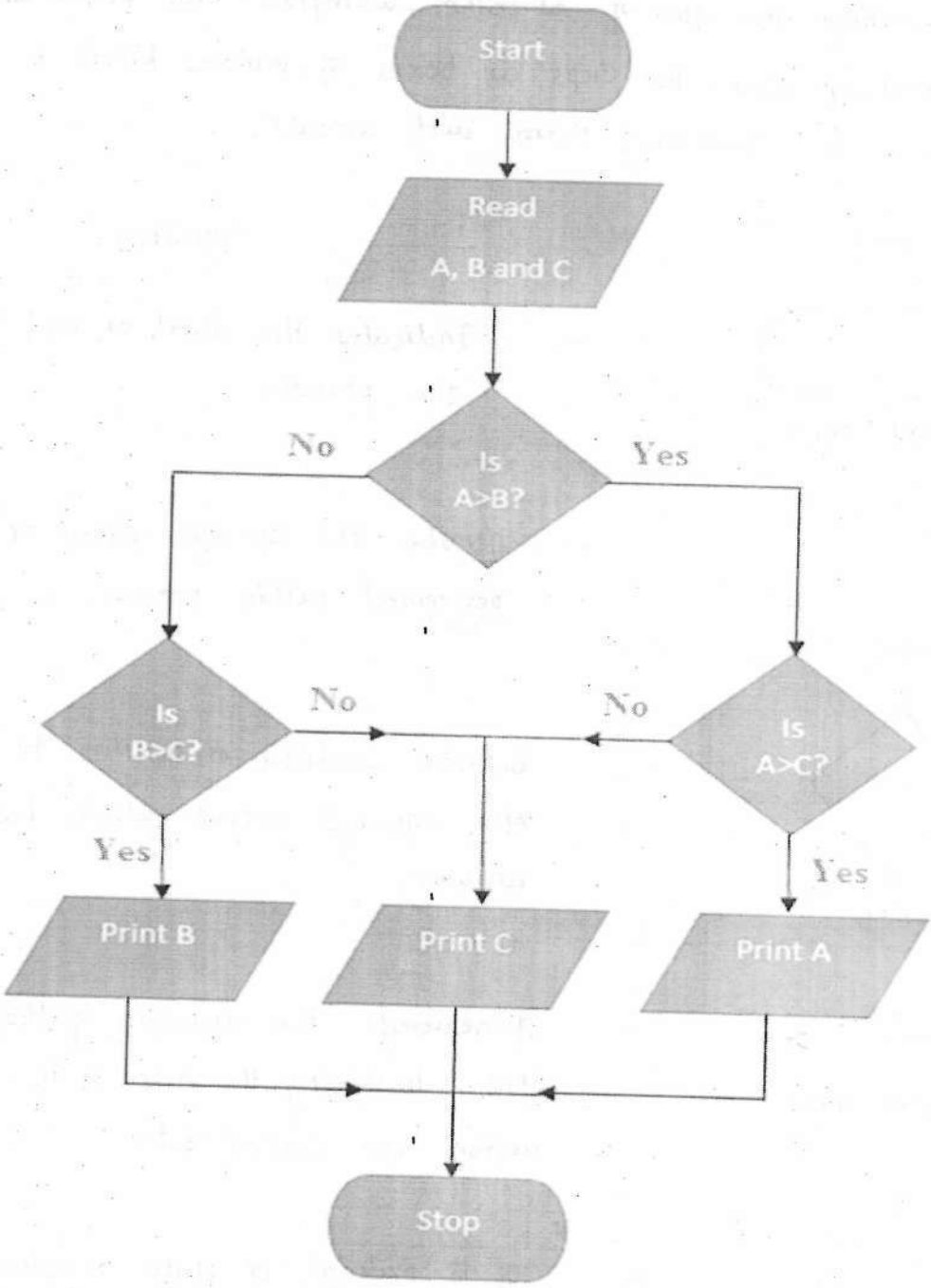
LEVEL 2 DFD – Decomposition of Process 3.0 (Issue/Return Books)



✱ Structured Flow chart —

- A structured flowchart is simply a graphical representation of steps.
- It shows steps in sequential order and is widely used in presenting the flow of algorithm, workflow and processes.
- flowchart shows the steps as boxes of various kinds & their order by connecting them with arrows.

Symbol	function.
 Start / End	- Indicates the start or end point of the process.
 process	- Describe the specific action or task to be performed within process.
 Decision	- Describe condition or question to be evaluated Also different output options based on the answer.
Flow arrow	- It represents the direction of the process flow, indicating the order of in which action are carried out.
 Input / output	- It is material or data entering or leaving the system.


Start
Read
A, B and C
Is
A>B?
No
Yes
Is
B>C?
No
No
Is
A>C?
Yes
Yes
Print B
Print C
Print A
Stop

* Decision Table - It is tabular method used to represent complex business rules and logic by mapping various input combinations to their corresponding output.

- It is a software testing technique which is used for testing the system behavior for different input combinations.
- this table helps you deal with different combination inputs with their associated outputs.
- A decision table is a structured grid that shows -
  - * conditions - the various inputs or criteria that the system must evaluate.
  - * Actions - the result or outputs that occur based on condition met.
  - * condition ~~entries~~ entries - the combinations of conditions (often represented as True/false) for each rule.
  - * Action entries - the specific action taken for each combination of condition.

eg. how to create decision table for login screen

- the condition states that if the user provides the correct username & password the user will be redirected to the home page.
- If any of input is wrong an error message will be displayed

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
username	F	T	F	T
Password	F	F	T	T
output	E	E	E	H

- In above excample,
  - T - correct username / password
  - F - wrong username / password
  - E - error message displayed
  - H - Home screen is displayed.

- Now let's understand and interpretation of the above cases.

- Case 1 - username and password both were wrong. the user is shown error message.

case 2 - username was correct, but the password was wrong the user is shown as error message.

case 3 - username was wrong. but the password was correct the user is shown as error message.

case 4 - username & password both are correct & the user is navigated to homepage.

➢ Example of transferring money online to an account which is already added and approved.
Here the conditions to transfer money are ACCOUNT ALREADY APPROVED, OTP (One Time Password) MATCHED, SUFFICIENT MONEY IN THE ACCOUNT
And the actions performed are TRANSFER MONEY, SHOW A MESSAGE AS INSUFFICIENT AMOUNT, BLOCK THE TRANSACTION INCASE OF SUSPICIOUS TRANSACTION
Here we decide under what condition the action be performed Now let's see the tabular column below.

ID	CONDITIONS/ACTIONS	TEST CASE 1	TEST CASE 2	TEST CASE 3	TEST CASE 4	TEST CASE 5
Condition 1	Account Already Approved	T	T	T	T	F
Condition 2	OTP (One Time Password) Matched	T	T	F	F	X
Condition 3	Sufficient Money in the Account	T	F	T	F	X
Action 1	Transfer Money	Execute				
Action 2	Show a Message as 'Insufficient Amount'		Execute			
Action 3	Block The Transaction Incase of Suspicious Transaction			Execute	Execute	X

In the first column I took all the conditions and actions related to the requirement
All the other columns represent Test Cases.

T = True, F = False, X = Not possible

From the case 3 and case 4, we could identify that if condition 2 failed then system will execute Action 3. So we could take either of case 3 or case 4

So finally concluding with the below tabular column

ID	CONDITIONS/ACTIONS	TEST CASE 1	TEST CASE 2	TEST CASE 3	TEST CASE 4
Condition 1	Account Already Approved	T	T	T	F
Condition 2	OTP (One Time Password) Matched	T	T	F	X
Condition 3	Sufficient Money in the Account	T	F	T	X
Action 1	Transfer Money	Execute			
Action 2	Show a Message as 'Insufficient Amount'		Execute		
Action 3	Block The Transaction Incase of Suspicious Transaction			Execute	X

We write 4 test cases for this requirement.

★ UML Modelling –

- unified modelling language.
- It is standard language for specifying, visualizing, constructing, and documenting the artifacts of software system.
- We use UML diagrams to show the behaviour & structure of system.
- UML helps software engineers, bussinessman & system architects with modelling, design & analysis.

1. Use case diagram –

- use case diagram represents the interaction between users and a system.
- It captures the functional requirments of a system, showing how different users engage with various use cases or specific functionalities within the system.

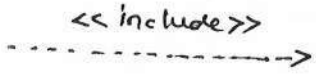
Symbols –

– Actor

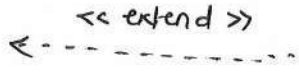
– use case

– Generalization symbol between actors & between use cases.

– Association between actor & use-case.



– Include relationship between use case



– Extend relationship between use case

* Actor – Actors are external entities that interact with the system this can include users or other system or hardware devices.
- actors initiate use cases & receives the outcome.

* Use case – use cases are like scenes in the play, they represent specific things your system can do

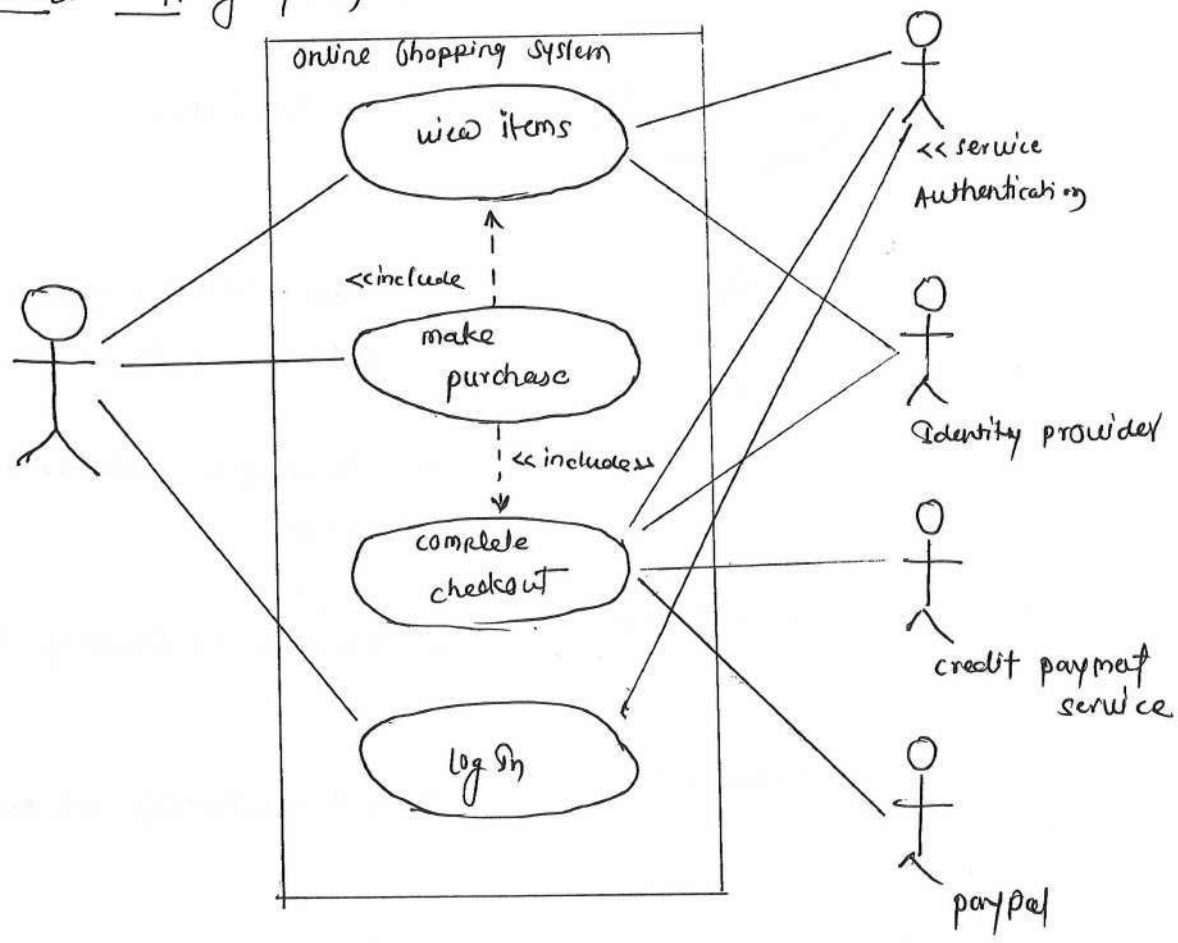
* Association – represents communication or interaction between an actor and use case.
- It is depicted by a line connecting the actor to the use case.

* Include relationship – It indicates that a use case includes the functionality of another use case.

* Extend Relationship – It illustrates that a use case can be extended by another use case under specific conditions.

* Generalization Relationship – It establishes an "Is-a" connection between two use cases. Indicates that one use case is a specialized version of another.

e.g. Online Shopping system –



2. Class Diagrams —

- It is static structure diagram that graphically represents a system's classes, their attributes, operations & the relationship between them.

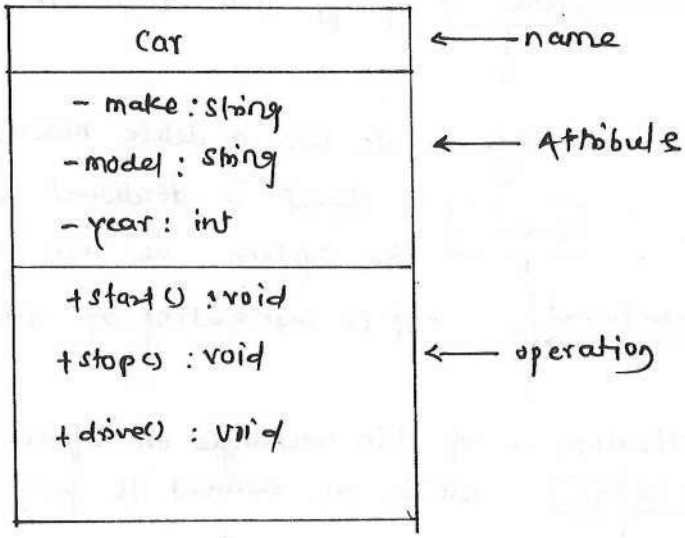
* **Classes** – Rectangle representing the fundamental building block of an object oriented system, showing the system's structure.

* **Attributes** – the data or variables that class holds, listed within the class box.

* **Operations** – the function or actions a class can perform, also listed within the class box.

* **Relationships** – Different types of arrows & lines connecting classes to show how they interact

1. Association (—) It is bi-directional relationship beth two classes
2. Directed Association (⟶) - represents relationship beth two classes wher the association has direction
3. Aggregation (⟷◇) - represents a "whole-part" relationship where one class contains or is composed another class
4. Composition (⟷◆) - Indicating a more significant ownership or dependency relationship.
5. Generalization (Inheritance) - represents an "is-a" relationship between classes.



- +/- visibility Notation — + for public (visible to all classes
  - for private (visible only within class)

3. Sequence Diagram – It shows how objects interact with each other over time in system.

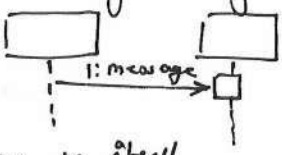
- It helps visualize the order of messages & operations between objects.
- Shows the sequence of operations over time.

Sequence Diagrams Notations —

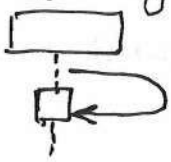
* Actors – represents a type of role where it interact with the system & its object. actor is always outside the scope of the system.

* Lifelines – A lifeline is named element which depict an individual participant in a sequence diagram. each instance is represented by lifeline. lifeline elements are located at top

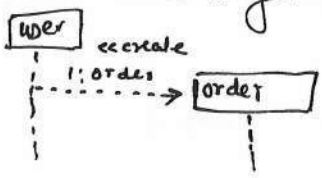
* Message – Communication between objects is depicted using messages
- message appears in sequencial order on the lifeline.



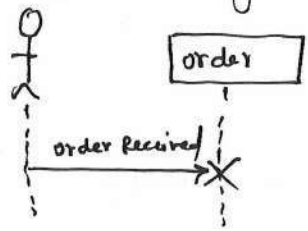
* Self message – Sometimes object needs to send message to itself Such message called as self message.



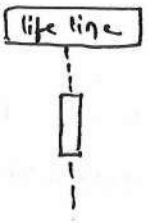
* Create message – to instantiate new object. It is represented with a dotted arrow & create word labelled on it specify that it is the create message symbol



* Delete Message – We use a delete message to delete an object when a object is deallocated memory or is destroyed within the system we use delete message symbol it is representing by arrow terminating with X



* Activation – A thin rectangle on lifeline represents the period during which an element is performing an operation.



* Testing -

- Software Engineering testing is a crucial software development lifecycle (SDLC) process that involves verifying and validating a software application to ensure it meets requirments, functions correctly & is free of bugs.
- software testing is the process of evaluating software to find bugs, verify requirments are met and ensure the product is high-quality, secure, efficient & user-friendly before release
- Its about verifying that the software meet its requirments and validating that its satisfies the user's actual needs.

* purpose of software Testing —

1. find and fix defects — Identify bugs, errors, gaps or missing functionalities in the software allowing them to be fixed before the product reaches the end users.

2. Ensure product Quality - Guarantee that the software is high Quality, reliable secure & perform efficiently under various condition

3. Improve user satisfaction - verify that the software is user friendly & meets user expectations.

4. Reduce cost - fixing defects early in development process is significantly less expensive than fixing them after the software has been release.

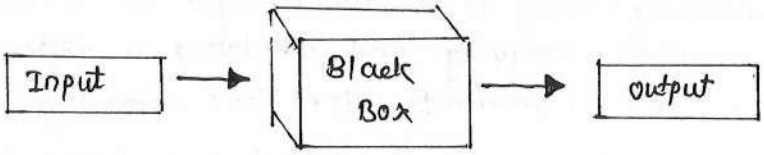
5. Risk mitigation - Reduce project risk related to software failure, security breaches and poor performance by identifying potential issues in advance.

* Testing Methods -

1. Black Box Testing -

- It is way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.
- It is also known as data-driven, box testing, data & functional testing.
- this type of testing is ideal for higher levels of testing like system testing, Acceptance testing.
- It is mostly done by software testers.
- No knowledge of implementation is needed.
- It is functional test of the software.

- Testing can start after preparing requirment specification documents



Technique used -

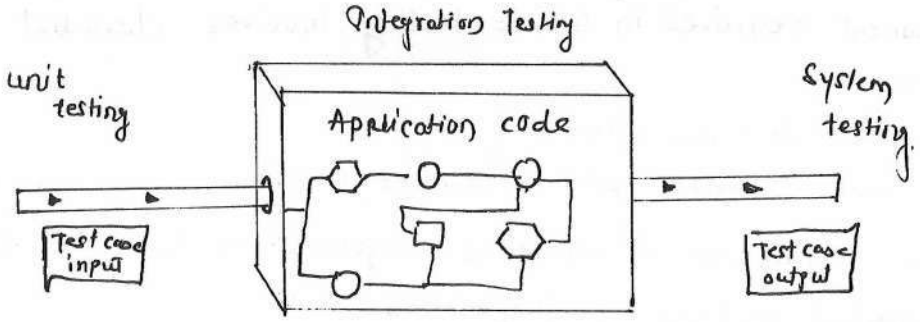
* equivalence partitioning - It divides input values into valid & invalid partitions & selecting corresponding values from each partition of the test data.
* Boundary value analysis - check boundaries for input values.

* Advantages -
  1. Efficient when used on large systems
  2. tester can be a non-technical
  3. There is no need for the tester to have detailed functional knowledge of system.
  4. Test will be done from an end users point of view.

* Disadvantages -
  1. Test cases are challenging to design without having clear functional specification.
  2. It is dificult to identify all possible inputs in limited testing time.
  3. There are chances of having unidentified paths during the testing process.

2. White Box Testing -

- It is way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
- It is also called structural testing, clear box testing, code-based testing. or glass box testing.
- Testing is best suited for lower level of testing like unit testing, Integration testing.
- It is mostly done by software developers
- knowledge of implementation is required
- Testing can start after preparing for Detailed design Documents.



* Techniques used -

1. Statement coverage - It validates whether every line of the code is executed at least once.
2. Branch coverage - It validates whether each branch is executed at least once.
3. path coverage - method tests all the paths of the program.

* Advantages -

1. code optimization by finding hidden errors
2. white box test cases can be easily automated
3. Testing is more through as all code paths are usually covered
4. Testing can start early in SDLC even if GUI is not available.

* Disadvantages -

1. white box testing can be quite complex & expensive
2. white box testing is time consuming, bigger programming applications take the time to test fully.
3. white box testing requires professional resources with a detailed understanding of programming & implementation.

3 Static Testing -

- Static testing is performed to check the defects in the software without actually executing code.
- the objective of Static testing is to prevent defects.
- It is performed at the early stage of software development
- In static testing, whole code is not executed.
- Static testing is performed before code deployment
- It is less costly.

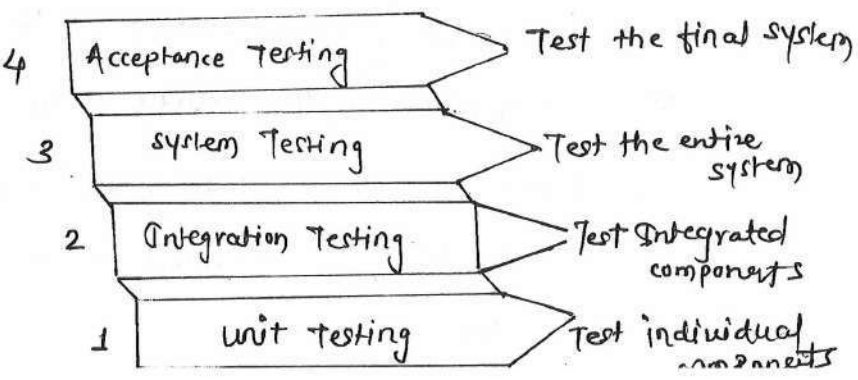
- document required in Static testing involves checklist for testing process.
- It can discover variety of bugs.
- It may complete 100% statement coverage incomparably in less time.
- the techniques it includes informal reviews, walkthrough, technical reviews & inspections.
- It is verification process.

4. Dynamic Testing –

- Dynamic Testing is performed to analyze the dynamic behaviour of the code.
- the objective of Dynamic Testing is to find & fix defects
- It is performed at the later stage of software development.
- In dynamic Testing, the whole code is executed.
- Dynamic Testing is performed after code deployment.
- It is highly costly.
- document required in dynamic Testing involves test cases for the testing process.
- It usually takes a longer time for the testing process
- It exposes the bugs that are explorable through execution hence discovering only a limited type of bugs.
- Dynamic Testing only achieves less than 50% coverage.
- It involves functional & non-functional testing.
- It is validation process.

* Level of Testing –

Fig. level of testing.



1. Unit Testing –

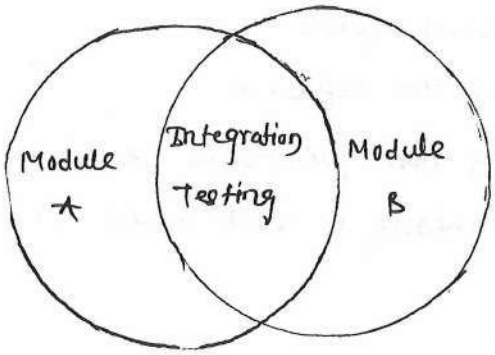
- Unit Testing is the first step in testing your software.
- It focuses on checking ~~the~~ individual components or functions of the application to make sure they work correctly on their own.
- The goal here is to catch any issues early before those small components are integrated with the rest of the system.
- Unit test helps developers spot bugs early in development process, making it easier & quicker to fix them.
- It's the first layer of defence to verify that each part of the application performs as expected.

features are –

1. Testing individual functions or components
2. Ensuring correct behaviour at the smallest level
3. catching errors early.

2. Integration Testing –

- After unit testing, Integration testing take over.
- This level checks how different modules or components of software work together.
- It is important because even if individual parts work perfectly, they might face issues when interacting with one another.
- Integration testing ensures that data flows correctly between modules and that they communicate seamlessly
- It helps catch problems that might arise when different parts of the system interact.



Types of Integration Testing: –

* Bottom-Up Integration Testing –

- Lower level modules are tested first, followed by gradual integration of higher level modules

- It uses test drivers to simulate higher-level modules that are not yet available
- focuses on validating internal module logic & data flow first
- Defects in core components are easier to detect & fix early.

* Top-Down Integration Testing -
- High-level modules are tested first & lower level modules are integrated step by step.
- uses stubs to simulate lower level modules during early testing.
- Helps validate system architecture & major workflows early in development.
- Ensures critical business logic is tested before full system integration.

* Mixed Integration Testing -
- combination of both top-down & bottom-up approaches executed in parallel.

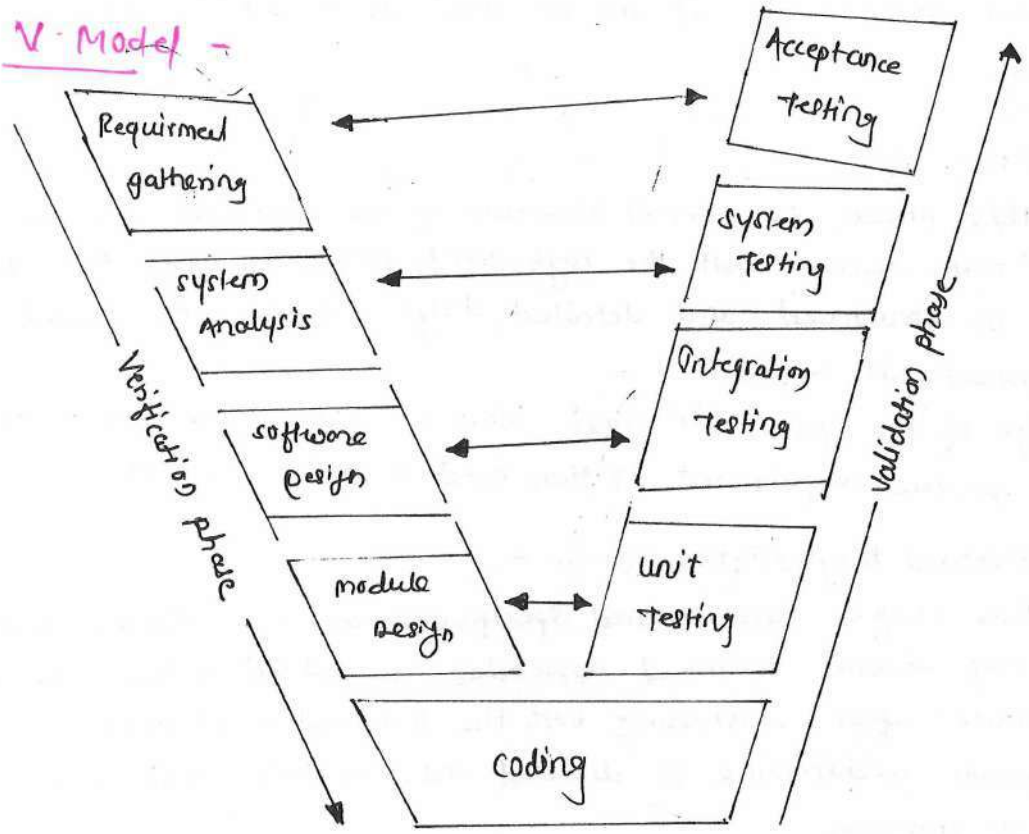
* Big-Bang Integration Testing -
- all modules are integrated at once & tested together as complete system.

3. System Testing -
- system testing is when you test the software as a system.
- this stage checks whether the entire system functions as expected in real-world environment.
- It include both functional & non-functional tests to ensure that the software meets customer needs
- full end to end testing of the software.
- verifying both functional & non-functional requirments
- Testing the software's behaviour in real-world conditions.

4. Acceptance Testing :-
- Acceptance Testing also known as User Acceptance Testing (UAT) is the final test before releasing the software to the end users
- In this phase, the customer or end-user verify if the software meets their needs and expectation.

- UAT is crucial because it verify the software is ready for production.
- It's the last chance to catch any overlooked issues before deployment
- If the software passes this stage, the customer gives the green light for release.
- It validating the software against business requirements
- Ensuring the software meets customer expectation
- Getting final approval from the customer.

★ V-Model -



★ V-model verification phases -

- This is where the process begins. The first step is to gather and understand the customers needs of the software.
- The goal is to define the scope of the project clearly to make sure everyone is on the same page.
- It involves a static analysis technique done without executing code.
- It is the process of evaluation of the product development phase to find whether specified requirments are met.
- There are several verification phases in the V-Model —

1. Business Requirment Analysis –

- This is the first step of the designation of the development cycle where product requirment needs to be cured from customers perspective.
- These phases include proper communication with the customer to understand their requirments.
- as most of time customers do not known exactly what they want & they are not sure about it at that time then we we acceptance test design planning which is done at the time of business requirment it will be used as an input for acceptance testing.

2. System Design –

- In this phase, the overall structure of the software is planned out
- the team develops both the high-level design i.e how the system will be structured and detailed design i.e how the individual component will work
- Design of the system will start when the overall we are clear with the product requirment. & then need to design the system completely.

3. Architecterral Design / Software Design –

- In this stage, architectural specification are comprehended & designed
- usually several technical approaches are put out & the ultimate choice is made after considering both the technical & financial viability
- A system architecture is divided into modules that each handle distinct function
- At this point the exchange of data and communication between the internal modules & external systems are well understood & defined
- During this phase integration tests can be created & documented using information provided.

4. Module Design –

- This phase known as low-level Design (LLD), Specifies comprehensive internal design for every system module.
- compatibility between the design & other external system as well as other modules in the system architecture is crucial,

- unit test are crucial component of any development process since they assist in identifying & eradicating the majority of mistakes and flaws at an early stage Based on the internal module design, this unit test may now be created.

5. Coding phase –

- This is where the software is actually built.
- Developers write the code based on the design created in the privious phase.
- The coding step involves writing the code for the system modules that were created during Design phase.
- The system & architectural requirments are used to determine which programming language is most appropriate.

Unit III –Software modelling and Design

Question Bank

1. Define Data Modelling in Software Engineering. **(S-26)**
2. Define the terms: (i) Abstraction (ii) Modularity. **(W-25)**
3. Explain the Elements of Analysis Model with neat diagram. **(W-25, S-26)**
4. Enlist/Describe the notations (symbols) used in Data Flow Diagram (DFD). **(W-24, W-25)**
5. Explain the V-Model of Software Testing. **(S-26)**
6. Sketch/Draw Use Case Diagram for a Management System (Hotel/Library). **(W-24, W-25, S-26)**
7. Draw and explain Sequence Diagram with suitable example. **(S-26)**
8. Compare/Differentiate Black Box Testing and White Box Testing. **(S-24, W-24, W-25)**
9. Differentiate between Static Testing and Dynamic Testing. **(S-26)**
10. Draw a Decision Table with suitable example. **(S-24, W-25)**
11. Draw Context/Level-0 and Level-1 DFD for a Management System (Library/Hotel/Railway/Book Publishing). **(S-24, W-24, S-25, W-25)**
12. Describe any six Fundamental Design Concepts with an example. **(S-26)**
13. Draw neat labelled diagram for translation of Requirement Model into Design Model and explain it. **(S-24, W-24, S-25)**