



Syllabus

- 1 **Introduction to Software Engineering** ✓
- 2 **Software Life Cycle Model** ✓
- 3 **Software Planning** ✓
- 4 **Requirement Analysis and Specification** ✓ (SRS)
- 5 **Software Design & Implementation** ✓
- 6 **Software Testing** ✓



Chapter-1 : Introduction to Software Engineering

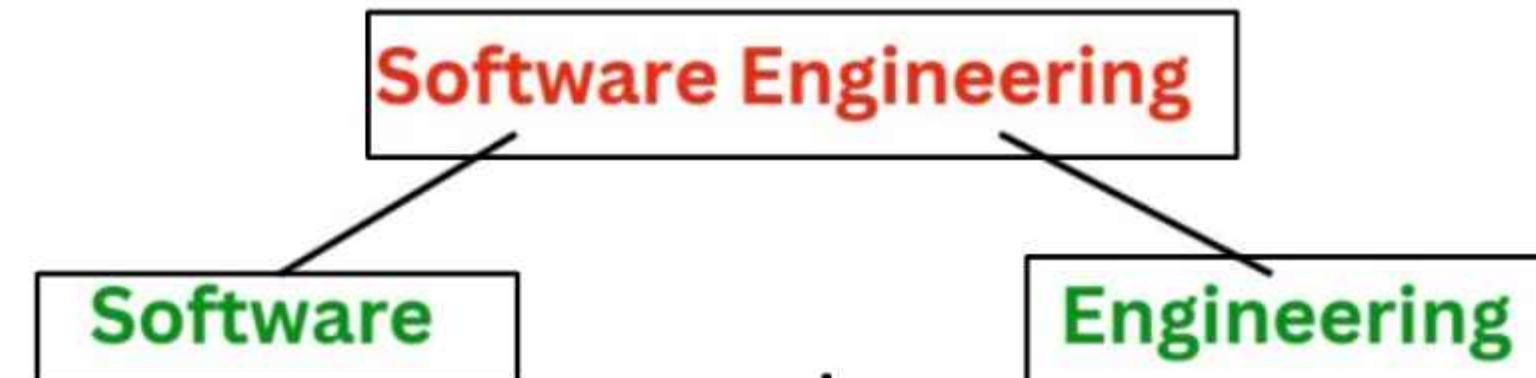
Syllabus :

1. Introduction to Software Engineering (10 periods)

System Concepts: **Types of systems : (open, closed, static and dynamic systems).**

Introduction, **Programmes v/s Software Products**

Emergence of Software Engineering- Early Computer Programming, High-level Language Programming, **Control flow based Design,** Data Structure Oriented Design, **Object Oriented Design**

**Software :**

- सॉफ्टवेयर instructions, प्रोग्रामों और डेटा का एक collection है जो कंप्यूटर को बताता है कि उसे कैसे काम करना है और कार्य कैसे Operate करना है।
- **Software is a set of instructions, programs, and data that tells a computer how to operate and perform tasks.**

Engineering :

- इंजीनियरिंग का मतलब है Well-defined structure , scientific principles और methods का उपयोग करके products का विकास करना।
- **Engineering is all about developing products, using well-defined structure, scientific principles and methods.**



Software Engineering:

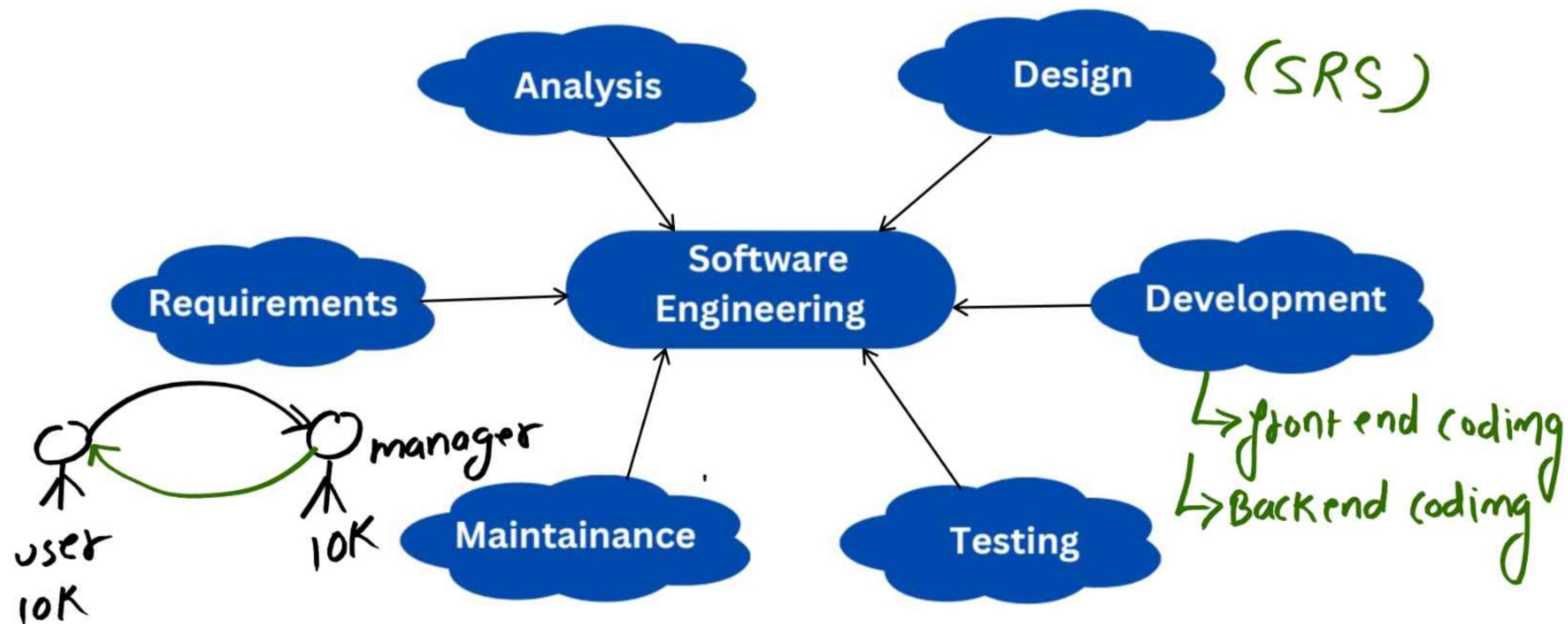
- सॉफ्टवेयर इंजीनियरिंग सॉफ्टवेयर application के डिजाइन, develop , test और maintaining की प्रक्रिया है।
- Software engineering is the process of designing, developing, testing, and maintaining software applications.

Need Of Software Engineering:

- गुणवत्ता युक्त सॉफ्टवेयर सुनिश्चित करता है। (Ensures quality software.)
- अपडेट करना आसान बनाता है। (Makes updates easier.)
- टीमवर्क का समर्थन करता है। (Supports teamwork.)
- समय और पैसा बचाता है। (Saves time and money.)
- नियमों और मानकों का पालन करता है। (Follows rules and standards.)
- जोखिम कम करता है। (Reduces risk.)



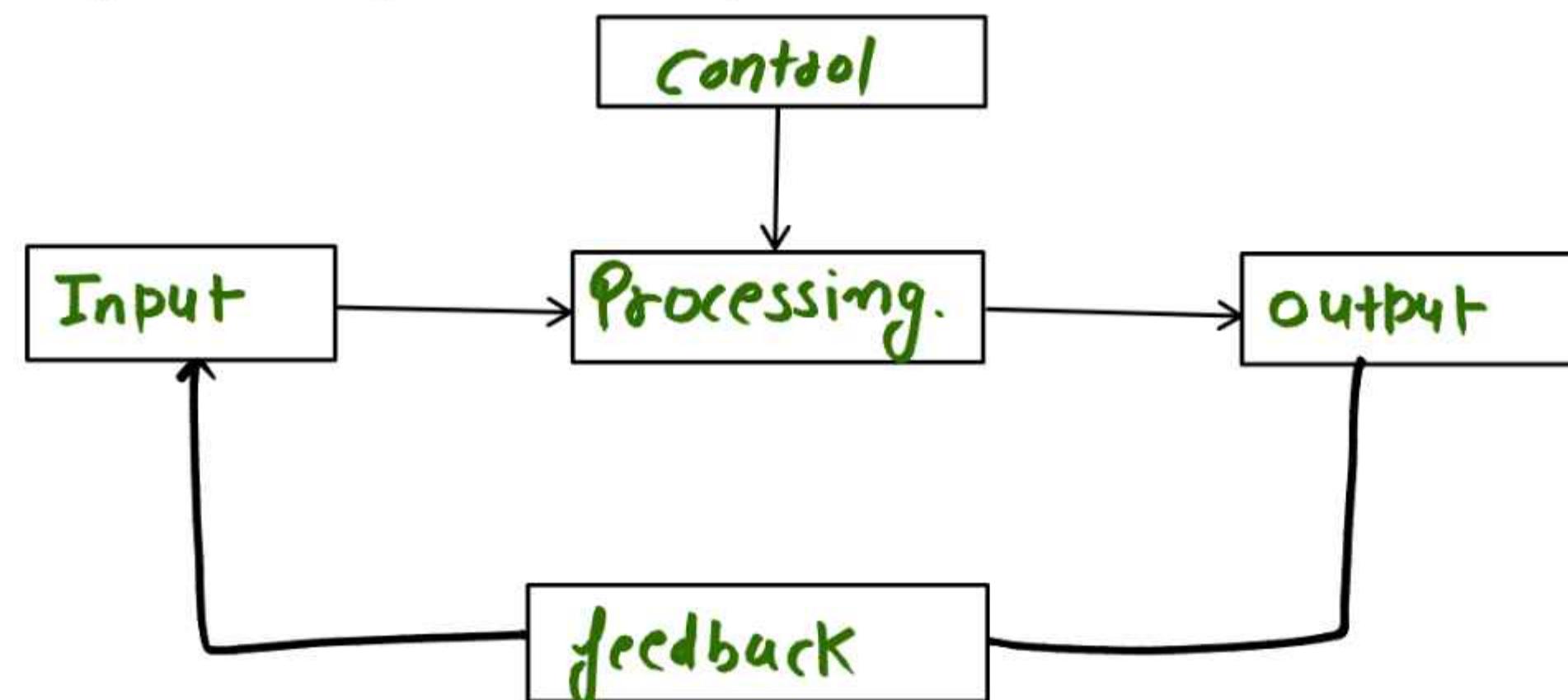
Components Of Software Engineering:





System:

- "सिस्टम एक ऐसा समूह होता है जिसमें कई सॉफ्टवेयर घटक (components) और हार्डवेयर संसाधन (resources) आपस में जुड़े होते हैं और मिलकर किसी विशेष कार्य को पूरा करते हैं।"
- "A system is a group of software components and hardware resources that work together to perform a specific task."





Based on Environment Interaction:

(a) Open System:

- An open system is one that interacts with its environment.
- It receive input from the outside and deliver output to outside.
- एक open system वह होती है जो अपने environment के साथ अंतःक्रिया करती है।
- यह बाहर से इनपुट प्राप्त करती है और बाहर ही आउटपुट देती है।

Example: ATM Machine.

- A living organism like a plant. It takes in sunlight, water, and nutrients, and it releases oxygen and waste.
- एक जीवित जीव जैसे कि पौधा। यह सूर्य का प्रकाश, पानी और पोषक तत्व ग्रहण करता है, तथा ऑक्सीजन और अपशिष्ट पदार्थ छोड़ता है।



Based on Environment Interaction:

(b) Closed System:

- A closed system is one that is isolated from its environment and does not exchange information, or materials with the outside world.
- बंद प्रणाली वह होती है जो अपने पर्यावरण से अलग होती है तथा बाहरी दुनिया के साथ सूचना या सामग्री का आदान-प्रदान नहीं करती।

Example:

- A sealed jar of gas. The gas inside does not interact with anything outside the jar.
- गैस से भरा एक sealed जार। अंदर की गैस जार के बाहर किसी भी चीज़ से प्रतिक्रिया नहीं करती।



Based on Time Change:

(a) Static System:

- Static systems are those system whose output depends upon only present value of input. They are physically reliable system. In computer terminology we can say static means fix.
- स्थिर प्रणाली वे प्रणाली हैं जिनका आउटपुट केवल इनपुट के वर्तमान मूल्य पर निर्भर करता है। वे physically रूप से reliable प्रणाली हैं। कंप्यूटर terminology में हम कह सकते हैं कि static का अर्थ है स्थिर।

Example: Html website

↳ Portfolio



Based on Time Change:

Example:

- A simple calculator app that performs basic arithmetic functions (addition, subtraction, multiplication, division) with a fixed user interface. The app's features and layout remain the same every time you use it.
- एक साधारण कैलकुलेटर ऐप जो बेसिक गणितीय क्रियाएँ (जोड़, घटाना, गुणा, भाग) करती है और इसका यूजर इंटरफ़ेस स्थिर रहता है। इस ऐप की विशेषताएँ और लेआउट हर बार उपयोग करने पर वही रहते हैं।

$$\begin{array}{r} 213 \Rightarrow 5 \\ 2+3 \Rightarrow 6 \\ \quad \quad \quad \downarrow \\ \quad \quad \quad 5 \end{array}$$



Based on Time Change:

(b) Dynamic System:

- Dynamic means capable of changing. There output may be variable output depends upon present as well as past value of input. A dynamic system is not physically .
- Dynamic का अर्थ है परिवर्तन करने में सक्षम। आउटपुट परिवर्तनशील हो सकता है आउटपुट इनपुट के वर्तमान और पिछले मूल्य पर निर्भर करता है। एक गतिशील प्रणाली भौतिक रूप से विश्वसनीय नहीं है

Example:

- A social media platform where the content and user interface change based on user preferences, interactions, and external trends.
- एक सोशल मीडिया प्लेटफॉर्म जहाँ content और user इंटरफ़ेस उपयोगकर्ता की preferences, बातचीत और बाहरी trends के आधार पर बदलते हैं।



Program:

- प्रोग्राम एक इंस्ट्रक्शन सेट होता है जो किसी स्पेसिफिक टास्क को पूरा करने के लिए लिखा जाता है।
- A program is an instruction set written to complete a specific task.
- प्रोग्राम का साइज छोटा होता है और यह एक सिंगल एप्लिकेशन या टास्क को पूरा करने के लिए यूज़ किया जाता है। $(KB \rightarrow MB)$
- The size of a program is small and it is used to complete a single application or task.
- जैसे, अगर आपको किसी कैलकुलेशन का काम करना है, तो उसके लिए आप एक सिंपल प्रोग्राम लिख सकते हैं।
- For example, if you have to do some calculation work, then you can write a simple program for that.

ex → Compiler (depend)
→ Single developer.



Software product:

- सॉफ्टवेयर प्रोडक्ट्स एक बड़ा पैकेज होता है, जिसमें कई प्रोग्राम्स और अन्य टूल्स शामिल होते हैं जो किसी खास जरूरत को पूरा करने के लिए डिज़ाइन किए गए होते हैं।
- Software products are large packages that contain several programs and other tools designed to meet a specific need.
- इन्हें बड़े स्केल पर यूज़ किया जाता है, जैसे विंडोज़ ऑपरेटिंग सिस्टम या माइक्रोसॉफ्ट वर्ड।
- They are used on a large scale, such as the Windows operating system or Microsoft Word.

Range → mB → GB

- ① operating system.
- ② group of people.



Difference Between Program and Software Product:

Sr.No	Program	Software Products
1	They are usually small in size	They are large in size
2	Can developed by a single developer	Team of developers required
3	Size varies from kilobytes (Kb) to Mega bytes (Mb).	Size varies from megabytes (Mb) to gigabytes (Gb)
4	It takes less time to develop.	It takes more time to develop



Difference Between Program and Software Product:

Sr.No	Program	Software Products
1	वे आम तौर पर आकार में छोटे होते हैं	वे आकार में बड़े हैं
2	एक ही डेवलपर द्वारा develop किया जा सकता है	डेवलपर्स की टीम आवश्यक है
3	size किलोबाइट्स (Kb) से मेगाबाइट्स (Mb) तक भिन्न (varies) होता है।	आकार किलोबाइट्स (Mb) से मेगाबाइट्स (Gb) तक भिन्न (varies) होता है।
4	इसे develop होने में कम समय लगता है।	इसे develop होने में अधिक समय लगता है



Difference Between Program and Software Product:

Sr.No	Program	Software Products
5	Functionality of a program depends on compiler.	Functionality of software depend on <u>operating system.</u>
6	Their are less no. of code line in a programme.	Their are very large number of code lines in software product.
7	There is no proper documentation for a programme.	For software programme there is proper documentation and also user manual is provided.
8	Functionality of a programme is upto a very limited extant.	It provides a large scale functionality.



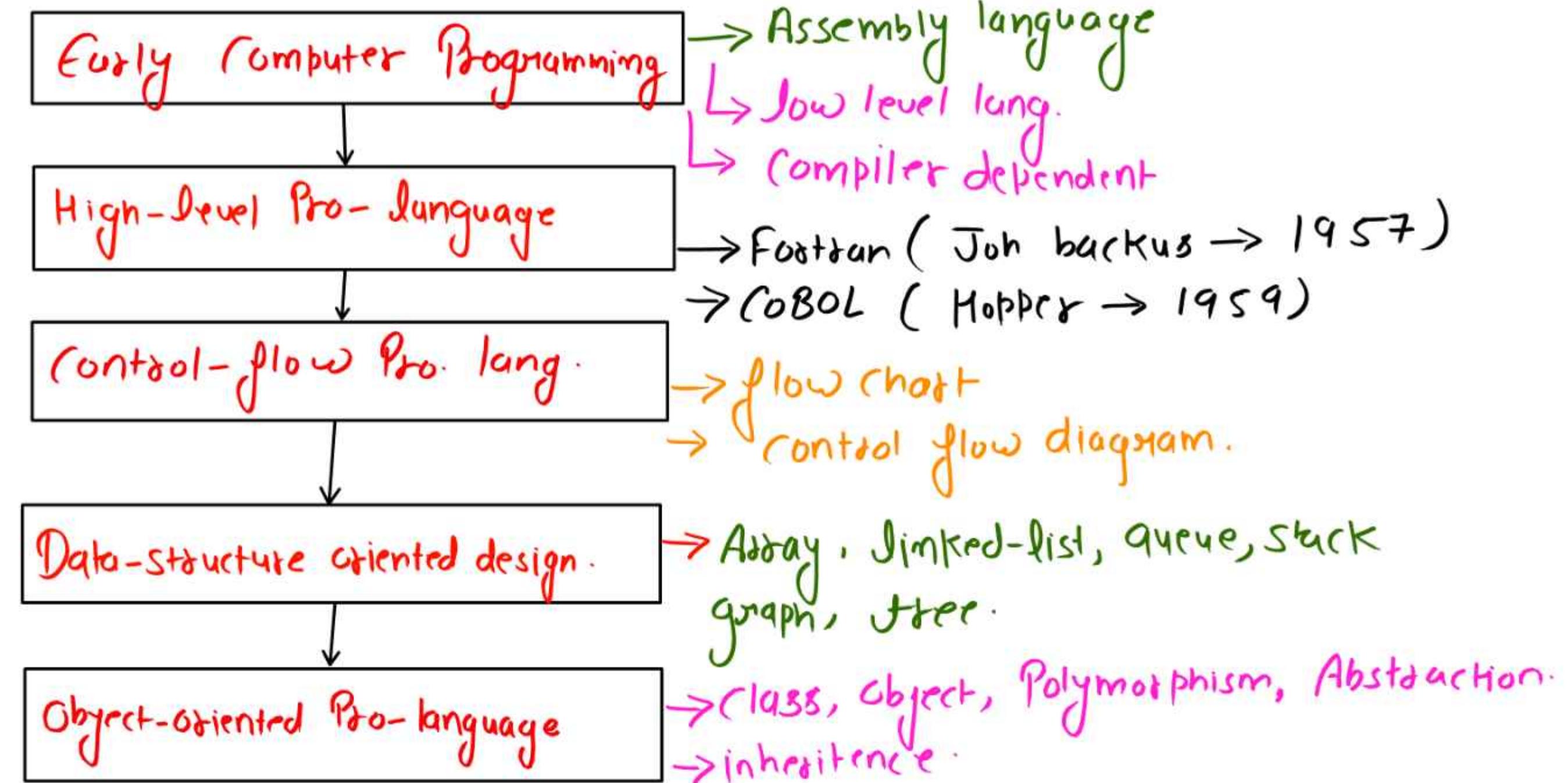
Difference Between Program and Software Product:

Sr.No	Program	Software Products
5	किसी प्रोग्राम की functionality कम्पाइलर पर निर्भर (depend) करती है।	किसी प्रोग्राम की functionality operating system पर निर्भर (depend) करती है।
6	किसी प्रोग्राम में कोड लाइन की संख्या कम होती है।	सॉफ्टवेयर products में कोड लाइनों की संख्या बहुत बड़ी होती है।
7	किसी भी programme के लिए कोई उचित documentation नहीं है।	सॉफ्टवेयर प्रोग्राम के लिए उचित documentation उपलब्ध हैं और user मैनुअल भी प्रदान किया गया है।
8	किसी programme की functionality बहुत सीमित सीमा तक होती है।	यह बड़े scale पर functionality प्रदान करता है।



EMERGENCE OF SOFTWARE ENGINEERING:

- सॉफ्टवेयर इंजीनियरिंग का emergence 1960 के दशक में हुआ, जब सॉफ्टवेयर development में कई समस्याएँ सामने आने लगीं।
- The emergence of software engineering occurred in the 1960s, when many problems in software development began to emerge.
- उस समय, सॉफ्टवेयर छोटे और simple होते थे, और उन्हें individual developer या छोटी टीमों द्वारा बनाया जाता था।
- At that time, software was small and simple, and was created by individual developers or small teams.





1: Early Computer Programming :

- उस समय मुख्य रूप से कंप्यूटर हार्डवेयर पर ध्यान focus किया जाता था। उस समय इस्तेमाल किए जाने वाले प्रोग्राम असेंबली (assembly) language में लिखे जाते थे।
- At that time mainly focus is on computer hardware. Programms used in that times are written in assembly language.
- लवलेस (Ada Lovelace) को पहली कंप्यूटर programmer के रूप में जाना जाता है।
- Ada lovelace is known as first computer programmer.
- 1940 में, IBM ने IBM 602 और IBM 604 जैसे पहले इलेक्ट्रॉनिक कंप्यूटर विकसित किए।
- In 1940, IBM has developed first electronic computer such as IBM 602 and IBM 604.

```
MOV A  
Add A,B  
SUB A,B
```



2: High-level Programming language:

- फोरट्रान और कोबोल जैसी early high level language के विकास के बाद सी++, पास्कल, विजुअल बेसिक, पायथन, जावा आदि जैसी कई भाषाएँ developed हुईं।
- After development of early high level language like Fortran and COBOL many language such as, C++, Pascal, Visual Basic, Python, Java etc.
- इन भाषाओं में सीधे रजिस्टर या मेमोरी एड्रेस जैसी चीज़ों से काम नहीं करना पड़ता, बल्कि वेरिएबल्स, एरेज़, बूलियन एक्सप्रेशन, ऑब्जेक्ट्स, फंक्शंस और लूप्स जैसी चीज़ों से काम किया जाता है।
- Instead of dealing with registers memory address etc., they deals with variable, arrays, boolean expression, objects, functions, loops etc.



Features Of high level Programming language:

- Human language के समान समझने में आसान।
- प्रोग्रामर के friendly , कोड करने में आसान
- रखरखाव में आसान।
- ये भाषाएँ compile और debug करने में आसान हैं।
- यह पोर्टेबल है, इसे संभालना आसान है।
- किसी भी प्लेटफ़ॉर्म पर चलाया जा सकता है
- Easy to understand similar to human language.
- ✓ Programmer friendly, easy to code
- ✓ Easy to maintain.
- ✓ These languages are carry to compile and debug.
- It is portable, easy to handle.
- ✓ Can run on any platform



4: Data Structure Oriented Design

- **Data Structure Oriented Design (DSOD)** एक ऐसी सॉफ्टवेयर डिज़ाइन तकनीक है जिसमें प्रोग्राम को इस आधार पर डिज़ाइन किया जाता है कि डेटा को कैसे स्टोर किया जाएगा और कैसे प्रोसेस किया जाएगा। इसमें सबसे पहले यह तय किया जाता है कि कौन-कौन से डेटा स्ट्रक्चर (जैसे कि Array, Stack, Queue, Linked List, Tree आदि) का इस्तेमाल करना है, फिर उसी के आधार पर प्रोग्राम की लॉजिक और फंक्शन बनाए जाते हैं।
- **Data Structure Oriented Design (DSOD)** is a software design technique in which a program is designed based on how the data will be stored and processed. In this approach, the first step is to decide which data structures (such as Array, Stack, Queue, Linked List, Tree, etc.) will be used. Based on these choices, the program's logic and functions are then developed accordingly.



5: Object Oriented Design:

- ऑब्जेक्ट-ओरिएंटेड डिज़ाइन (OOD) एक विधि है जिसमें सॉफ्टवेयर सिस्टम को डिज़ाइन करने के लिए उसे ऑब्जेक्ट्स में **break** किया जाता है, जो कि क्लासेज़ के उदाहरण (instances) होते हैं।
- Object-Oriented Design (OOD) is a method of designing a software system by breaking it down into objects, which are instances of classes.
- ये ऑब्जेक्ट्स आपस में **interact** करते हैं ताकि सिस्टम के आवश्यक ऑपरेशन्स को पूरा किया जा सके।
- These objects interact with each other to perform the desired operations of the system.



3: Control Flow Based Design:

- High level languages के introduction के साथ ही computer का usage तेजी से बढ़ गया। Programs और ज्यादा complex और बड़े होने लगे, जिससे इन्हें manage करना मुश्किल हो गया।
- With introduction of high level languages, the computer usage increase rapidly. Programs become more complex and large in size. So difficult to manage.
- Control Flow Based Design एक प्रोग्रामिंग approach है जिसमें कोड के execution की sequence को कंट्रोल स्ट्रक्चर्स जैसे कि कंडीशन्स (if-else), लूप्स (for, while), और switch के ज़रिए मैनेज किया जाता है।
- Control flow based design is a programming approach in which the sequence of code execution is managed through control structures like conditions (if-else), loops (for, while), and switches.

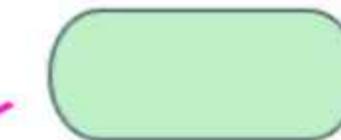
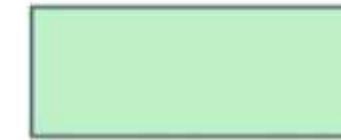
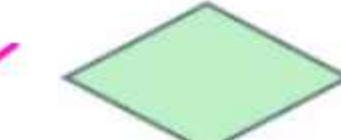


Flow Chart:

- फ्लो चार्ट किसी process का graphical या symbolic representation होता है। यह process flow के सभी detail को represent करता है।
- Flow chart are graphical or symbolic presentation of a process. It shows all the detail about the process flow.
- अलग-अलग processes को अलग-अलग symbols द्वारा represent किया जाता है। ये symbol arrow (arrow lines) से जुड़े होते हैं। फ्लो चार्ट प्रोग्राम के step to step process को दर्शाता है।
- So different process are denoted by different symbols. These symbols are connected by arrow lines. Flow chart represent step to step process of program.



Symbols of flow chart:

Symbol	Name	Function
	Oval	Represents the start or end of a process
	Rectangle	Denotes a process or operation step
	Arrow	Indicates the flow between steps
	Diamond	Signifies a point requiring a yes/no
	Parallelogram	Used for input or output operations



Example: Write an Algorithm to check whether a number is Positive or Negative

Algorithm

Step 1: Start ✓

Step 2: Input a number (say num)

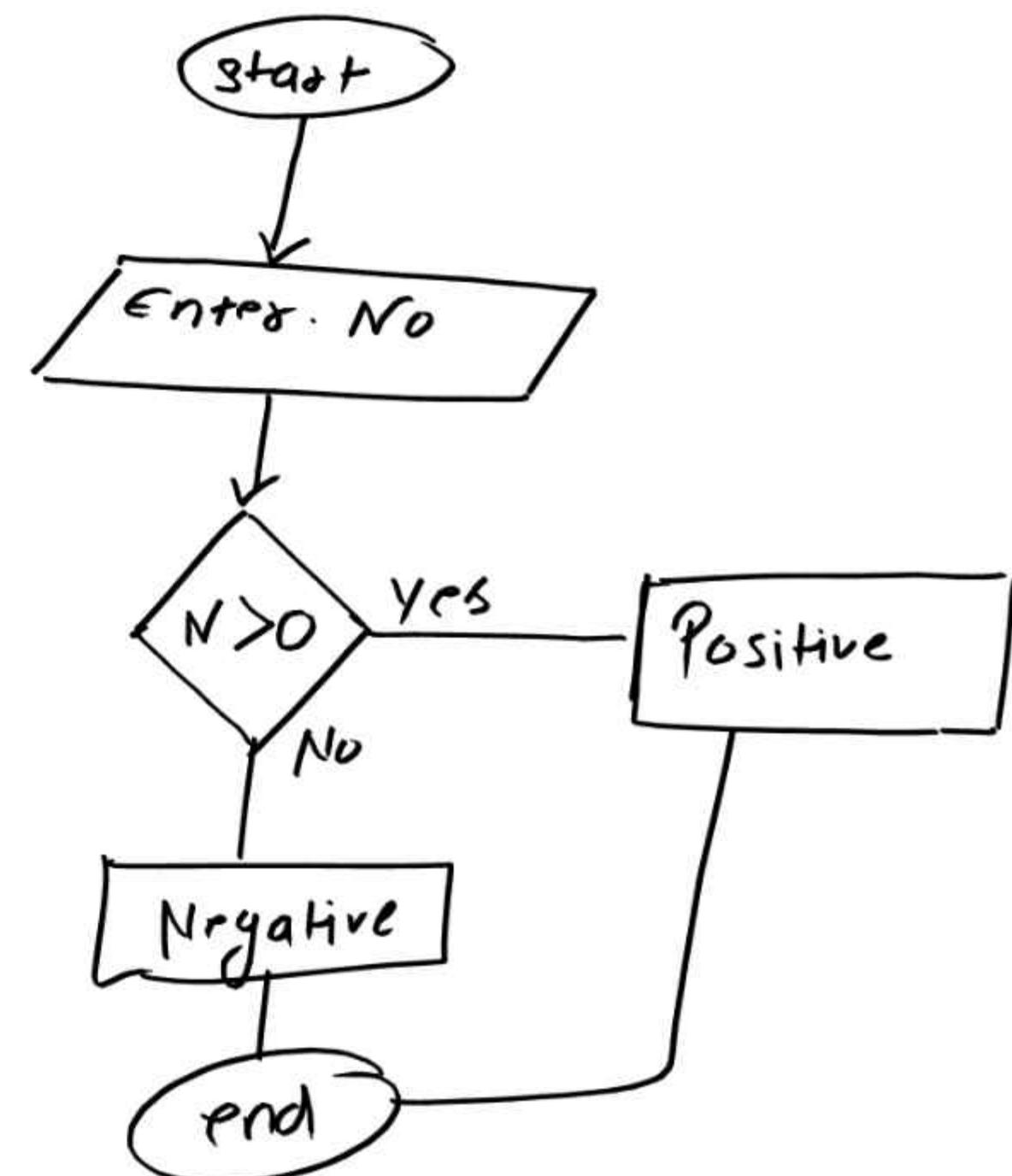
Step 3: If num > 0

→ Then print "Number is Positive"

→ Else

→ Print "Number is Negative".

Step 4: Stop ✓





Example: Write an Algorithm to check whether a number is positive, negative, or zero.

Algorithm

Step 1: Start

Step 2: Input the number

Step 3: Is the number > 0 ?

Step 4: If yes, print “Number is Positive”

Step 5: Else, is the number < 0 ?

Step 6: If yes, print “Number is Negative”

Step 7: Else, print “Number is Zero”

Step 8: End



→ Advantage of Flow Chart:

- फ्लो चार्ट बनाना बहुत आसान होता है।
 - यह logic को समझने में मदद करती है।
 - Non- technical लोग भी आसानी से गलतियों को पहचान सकते हैं।
 - ब्रांचिंग और लूपिंग को दिखाना भी आसान होता है।
- Easy to draw.
- Easy technique to understand logic.
- Even non-technical people can easily identify mistakes.
- Easy for branching and looping.



Disadvantage of Flow Chart:

- फ्लो चार्ट बनाने में समय लग सकता है।
 - फ्लो चार्ट को modification करना मुश्किल हो सकता है।
 - बड़े प्रोग्राम्स के लिए फ्लो चार्ट बनाना कठिन कार्य हो सकता है।
- Time consuming.
- Difficult for modification.
- Difficult task to draw flow charts for big programs.



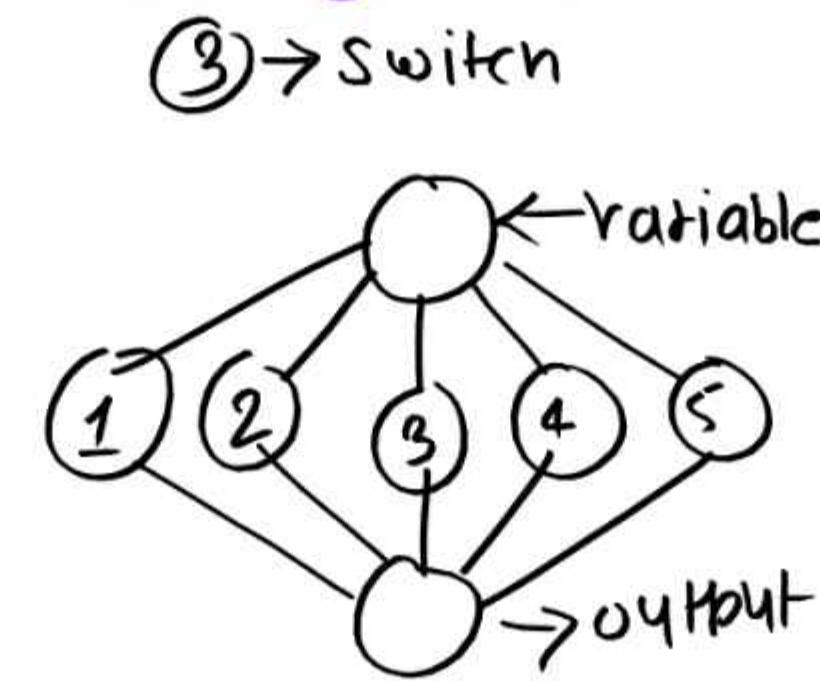
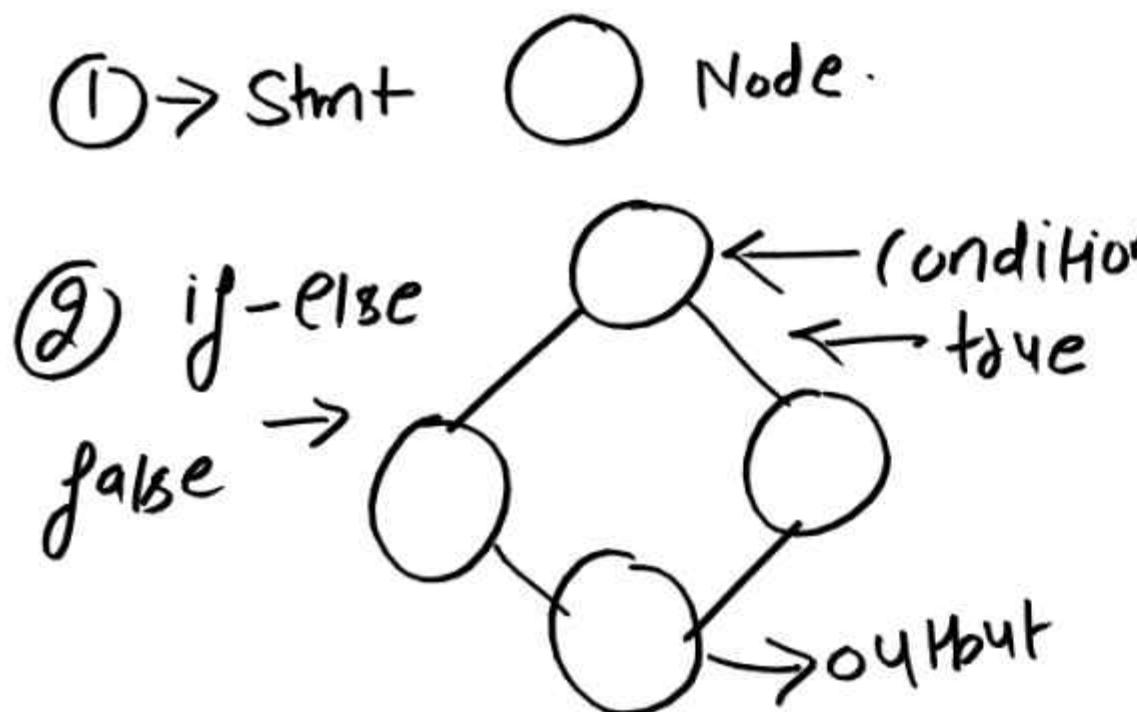
Control Flow Graph:

- कंट्रोल फ्लो ग्राफ़ (CFG) एक ग्राफिकल representation है जो एक प्रोग्राम के execution के दौरान सभी paths or track को दिखाता है। कंट्रोल फ्लो ग्राफ में प्रत्येक नोड एक बेसिक ब्लॉक को दर्शाता है।
- Control flow graph (CFG) is a graphical representation of all paths of a program during its execution. Each node in a control flow graph represents a basic block.

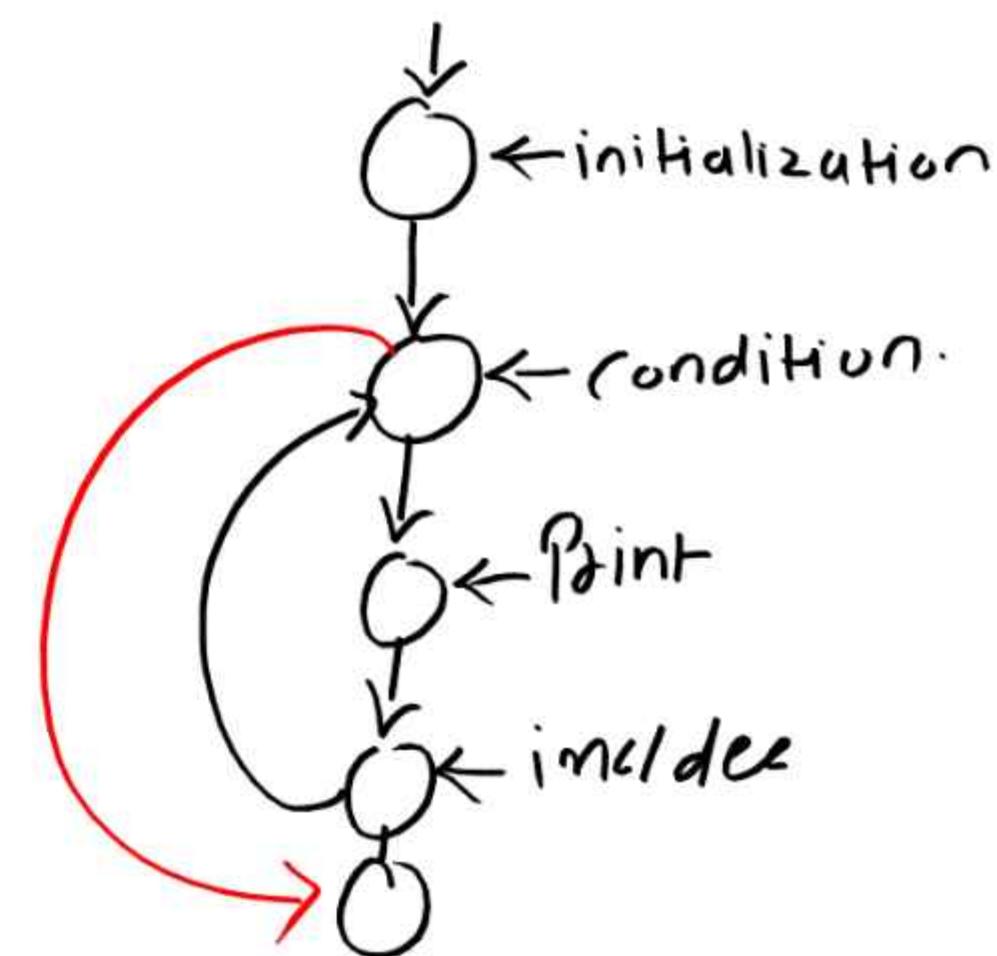
There are only two symbols in CFG:

1. Node: Represent each block.

2. Arrow: start from source and end at target block.



for(int i=0; i<5; i++)





Example: Write an Algorithm to find the greater of two numbers.

Algorithm

Step 1: Start

Step 2: Input two numbers: A and B

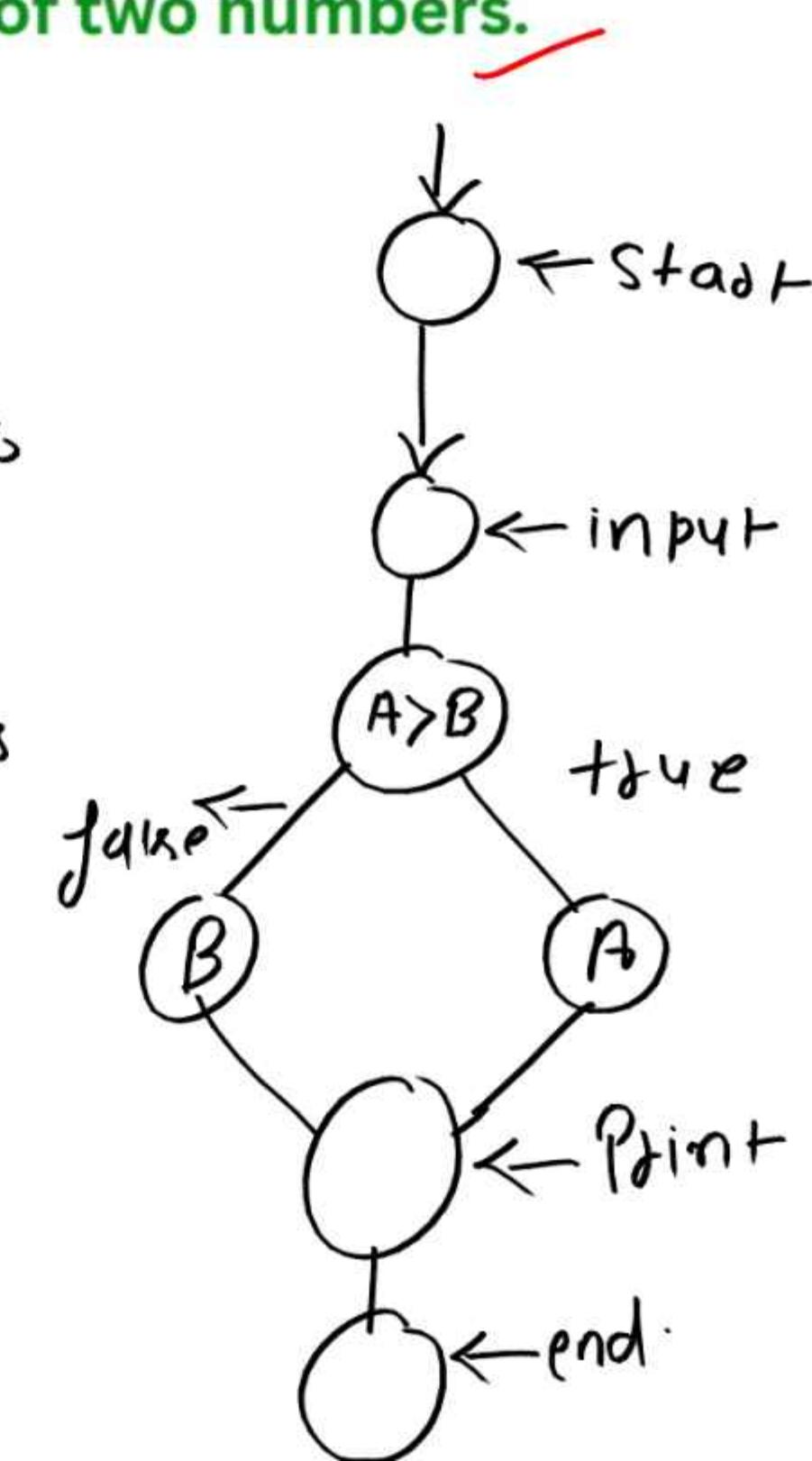
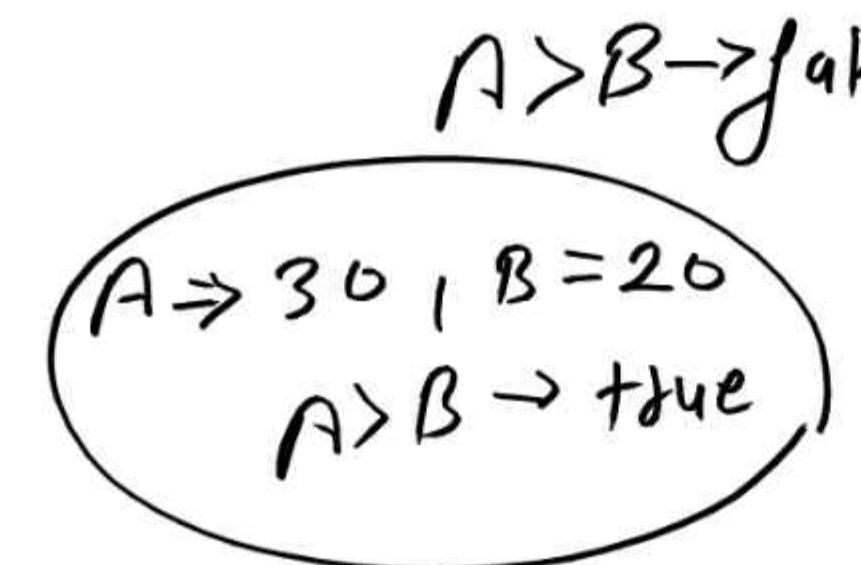
Step 3: Is $A > B$?

Step 4: If yes, print "A is greater"

Step 5: Else, print "B is greater"

Step 6: End

20 36
A B





Example: Write an algorithm to check a given number is odd or even

Algorithm

Step 1: Input the Number

Step 2: Is no is divisible by 2?

Step 3: If yes, print number is even

Step 4: If no, print number is odd

$N \rightarrow 10$
 ↓
 even

$N = 3$
 ↓
 odd.



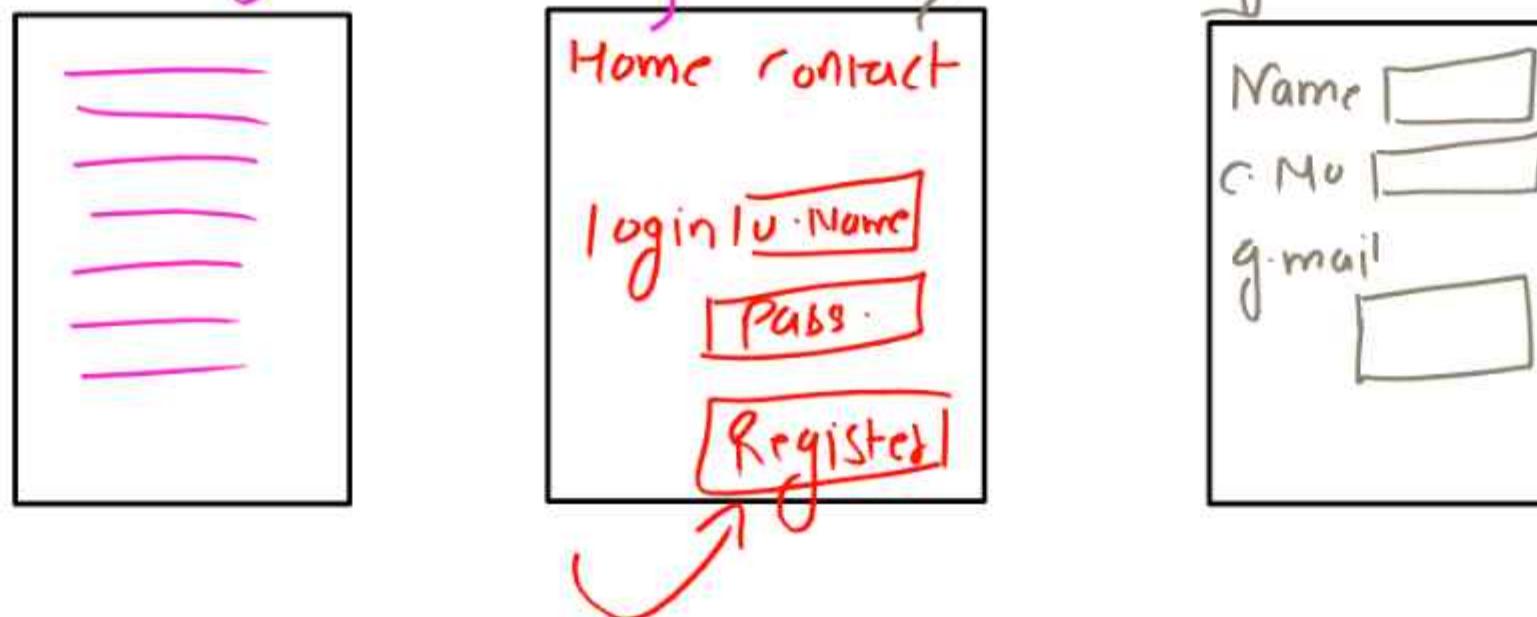
Features of a Control Flow Graph:

- यह एक प्रोग्राम का ग्राफिकल नोटेशन दिखाता है।
- यह प्रत्येक बेसिक सेक्शन की जानकारी प्रदर्शित करता है।
- यह उन कोड को खोजने में मदद करता है जो उपयोग में नहीं हैं।
- यह लूप्स और अन्य प्रोग्राम संरचनाओं को दिखाता है।
- It shows a graphical notation of a program.
- It displays information about each basic section.
- It helps find code that isn't used.
- It shows loops and other program structures.



Data Flow Oriented Design:

- Data Flow Oriented Design एक ऐसा सॉफ्टवेयर डिज़ाइन करने का तरीका है जिसमें इस बात पर ध्यान दिया जाता है कि डेटा सिस्टम के माध्यम से कैसे flow होता है।
- Data Flow Oriented Design is a way of designing software by focusing on how data moves through the system.
- इसके बजाय कि चीजें कैसे organized हैं (जैसे कि object-oriented design में), यहाँ हमारा मुख्य ध्यान इस पर होता है कि डेटा कैसे सिस्टम में enters करता है, कैसे flow होता है, और last में कैसे बाहर निकलता है।
- Instead of worrying about how things are organized (like in object-oriented design), we care more about how data enters, moves through, and leaves the system.





→ Data Flow Diagrams (DFDs):

- Data flow oriented design समझने के लिए हम Data Flow Diagrams (DFDs) का उपयोग करते हैं, जो map की तरह होते हैं और यह दिखाते हैं कि डेटा एक step से दूसरे step तक कैसे flow होता है।
- To visualize this, we use Data Flow Diagrams (DFDs), which are like maps showing how data flows from one step to another.
- प्रत्येक circle या bubble एक प्रोसेस को दर्शाता है, और तीर (arrows) यह दिखाते हैं कि इन step के बीच डेटा कैसे चलता है।
- Each circle or bubble shows a process, and arrows show the movement of data between these steps.

Component of Data Flow Diagrams (DFDs):

- (A) Entity.** **(B) Process.** **(C) Data Flow**
(D) Wavehouse/ Database. **(E) Terminator.**

(a) Entities:

- एंटिटी information या डेटा के source और destination होते हैं, या हम कह सकते हैं डेटा के इनपुट या आउटपुट होते हैं। इन्हें आयत (rectangles) द्वारा दर्शाया जाता है।
 - Entities are source and destination of information or data, or we can say input or output of data. They are represented by rectangles.

(b) Process:

- Process इनपुट को आउटपुट में transform करती है। यह डेटा पर की गई operation होती है। इसे circle या गोल कोनों वाले आयतों द्वारा दर्शाया जाता है।
 - Process transform input into output. It is operation done on data. It is shown by circle or round cornered rectangles.





(c) Data flow:

- यह system के एक part/step से दूसरे part/step तक information /डेटा/सामग्री का transfer को show करता है। इसे तीर (arrow) द्वारा represent किया जाता है। तीर flow की दिशा को दिखाता है।
- It shows transfer of information/data/material from one part/stage of system to another part/stage. It is represented by arrow. The arrow show the direction of flow

(d) Wavehouse/database:

- Process डेटाबेस का उपयोग डेटा को फ़ाइलों या अन्य रूपों में future में उपयोग के लिए store करने के लिए किया जाता है।
- Database is used to store data in form of files or other form for later use. We can use this data later on.
- हम इस डेटा का बाद में उपयोग कर सकते हैं। इसे दो समांतर रेखाओं द्वारा दर्शाया जाता है, और इन रेखाओं के बीच में स्टोरेज डिवाइस का नाम लिखा जा सकता है।
- It is represented by two parallel lines and name of storage device can be written in between these lines



(e) Terminator:

- यह एक बाहरी एंटिटी होती है जो सिस्टम के साथ **communicates** रती है, लेकिन सिस्टम का हिस्सा नहीं होती। यह system का बाहरी element होता है।
- It is an external entity which communicate with system but not a part of system. It is outside element of a system

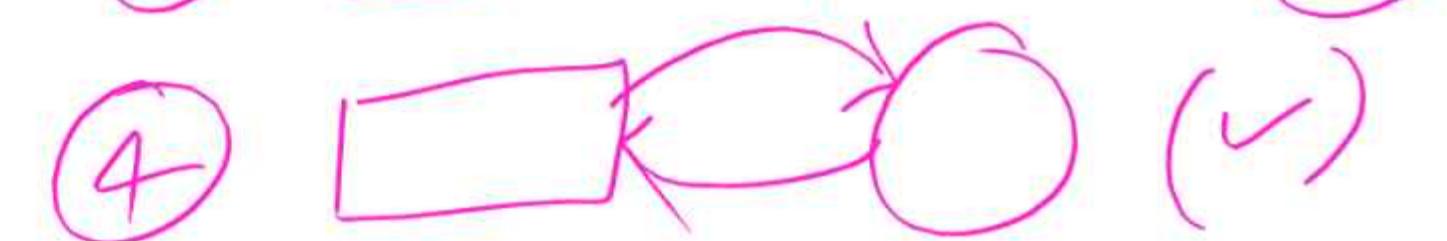
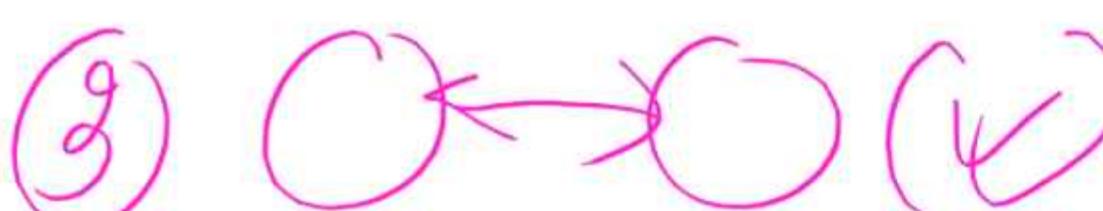
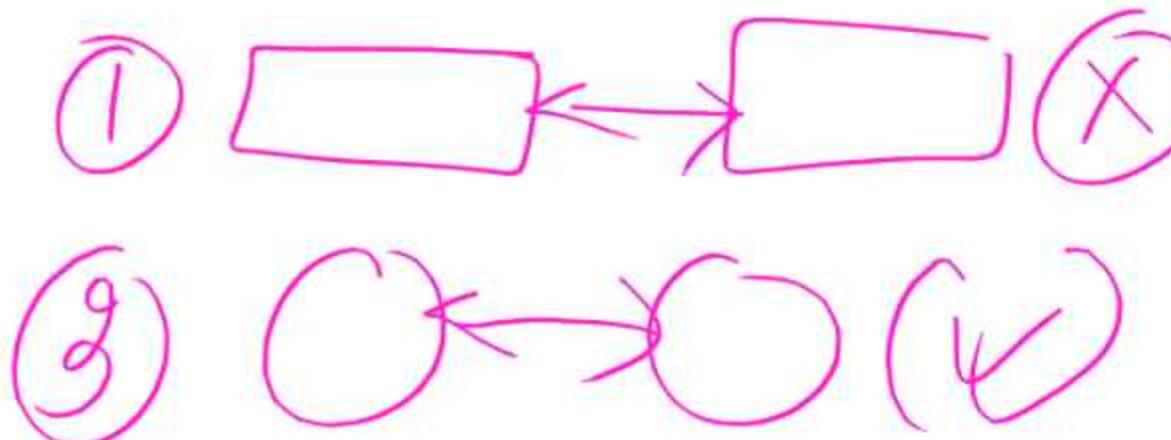
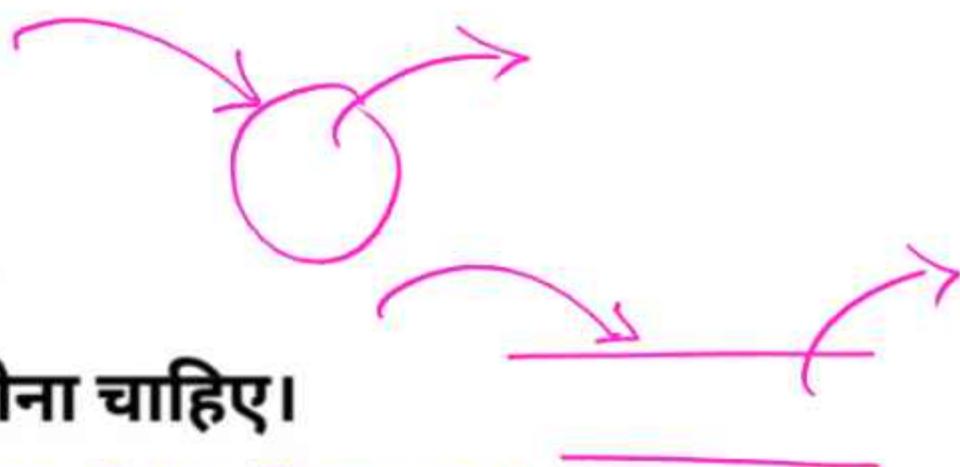
Level in a Data Flow Diagram (DFD):

- (i) Level 0 DFD.
- (ii) Level 1 DFD.
- (iii) Level 2 DFD.



Rules for DFD:

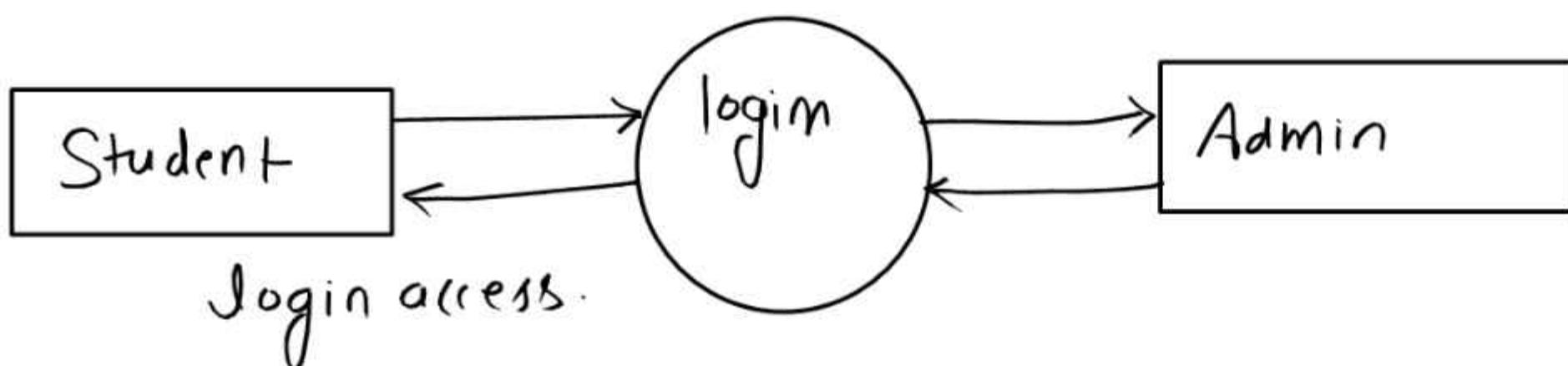
- प्रत्येक process में कम से कम एक इनपुट और एक आउटपुट होना चाहिए।
- Each process should have at least one input and an output.
- प्रत्येक डेटा स्टोर में कम से कम एक डेटा फ्लो अंदर और एक डेटा फ्लो बाहर होना चाहिए।
- Each data store should have at least one data flow in and one data flow out.
- किसी भी सिस्टम में store डेटा को एक process से होकर गुजरना चाहिए।
- Data stored in a system must go through a process.
- DFD (डेटा फ्लो डायग्राम) में सभी processes या तो दूसरी process या डेटा store की ओर जाती हैं।
- All processes in a DFD go to another process or a data store.





(i) Level 0 DFD:

- इसे संदर्भ आरेख (Context Diagram) भी कहा जाता है। इसमें एक ही process होती है और इनपुट और आउटपुट केवल इसी एकल process से जुड़े होते हैं।
- It is also called context diagram. Here there is a single process and input and output are linked with this single process only.

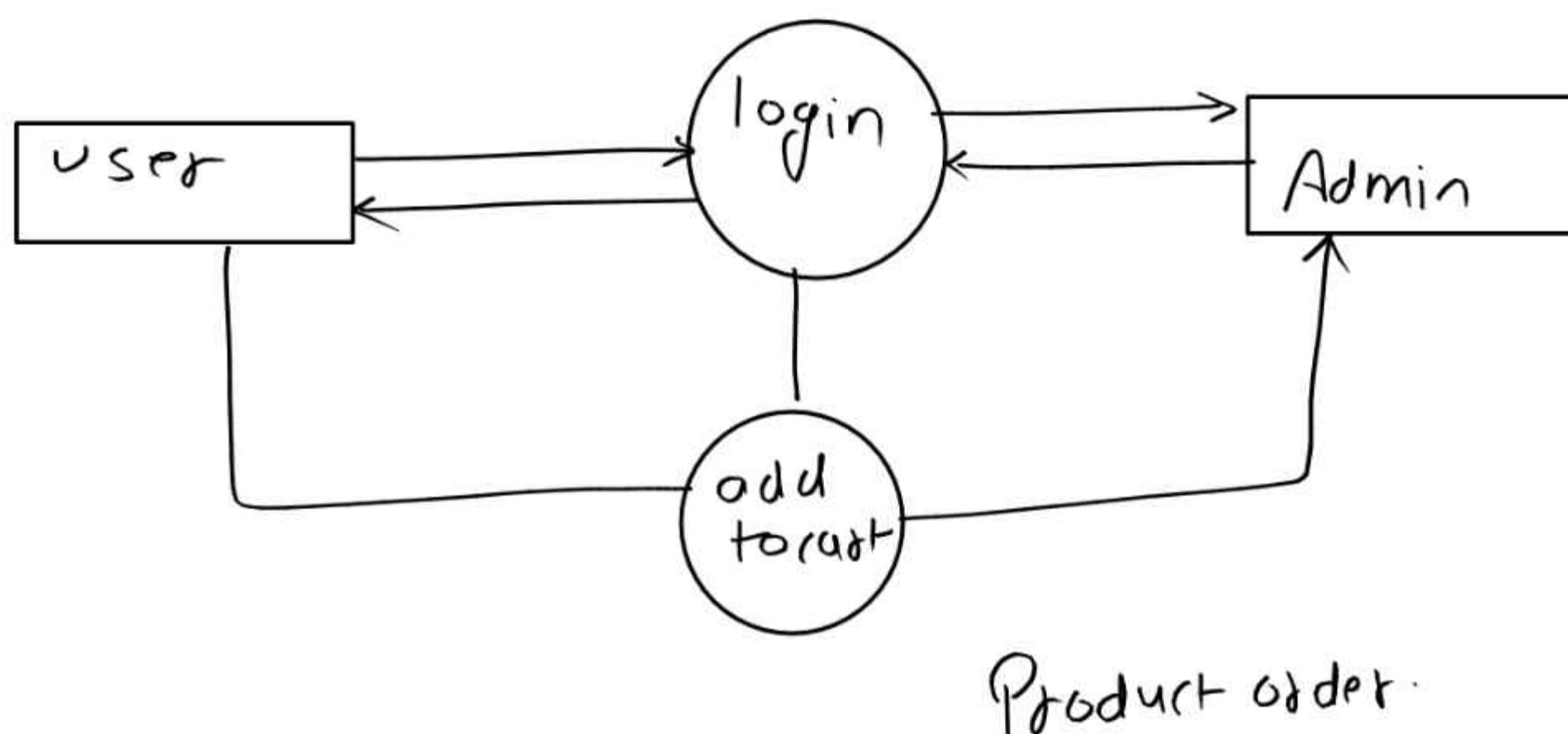


Student Login.



(ii) Level 1 DFD:

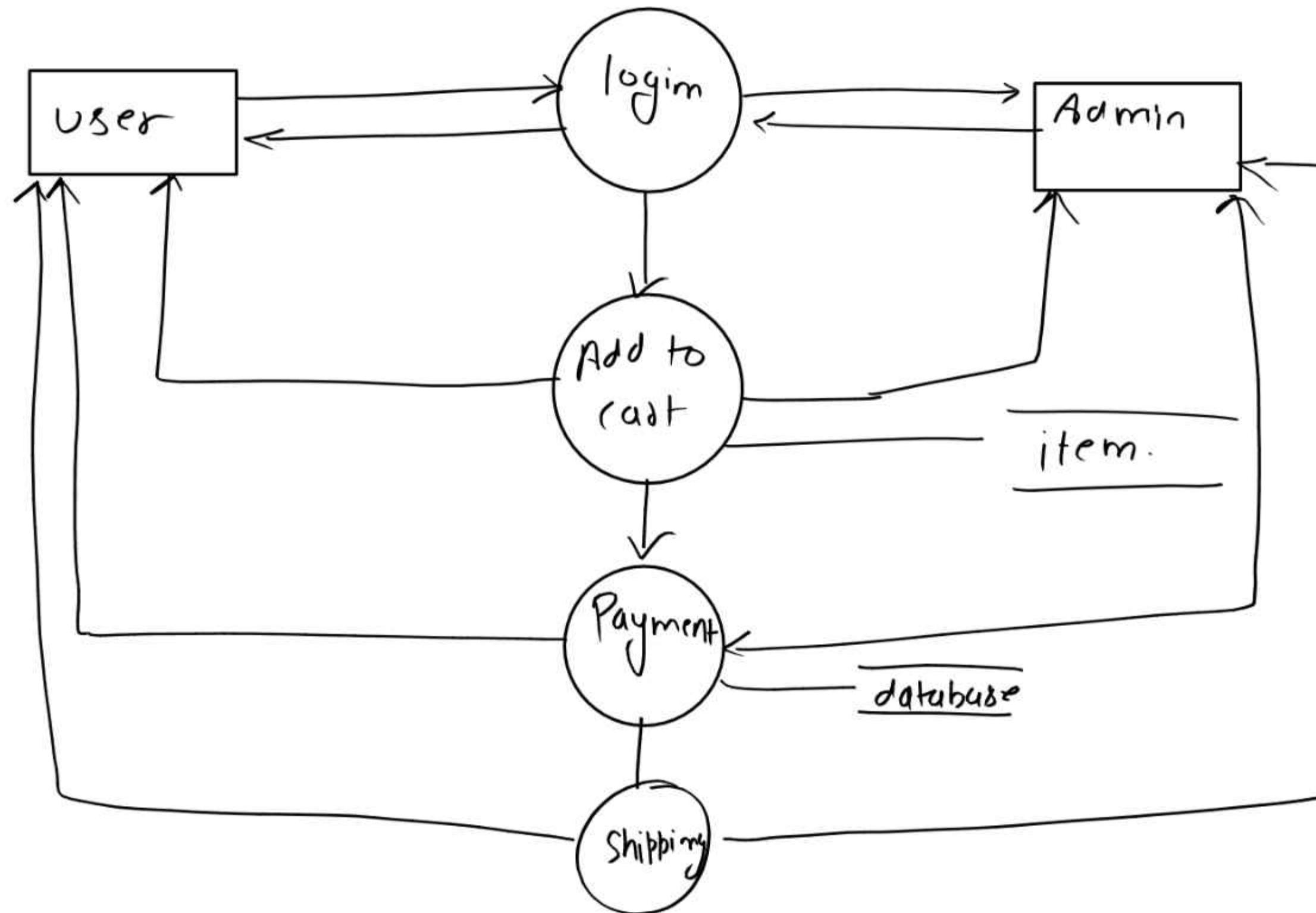
- इसमें multiple processes होती हैं। पूरी process को कई sub-process में विभाजित किया जाता है।
- It has multiple processes. The whole process to divided into many sub-process.





→ (iii) Level 2 DFD:

- यह process की अधिक जानकारी प्रदान करता है। यह process के अधिक detail को represents करता है। इसका उपयोग system की specific structure को रिकॉर्ड करने या plan बनाने के लिए किया जा सकता है।
- It goes into more details of process. It presents more detail of the process. It can be used to record or plan specific makeup of a system



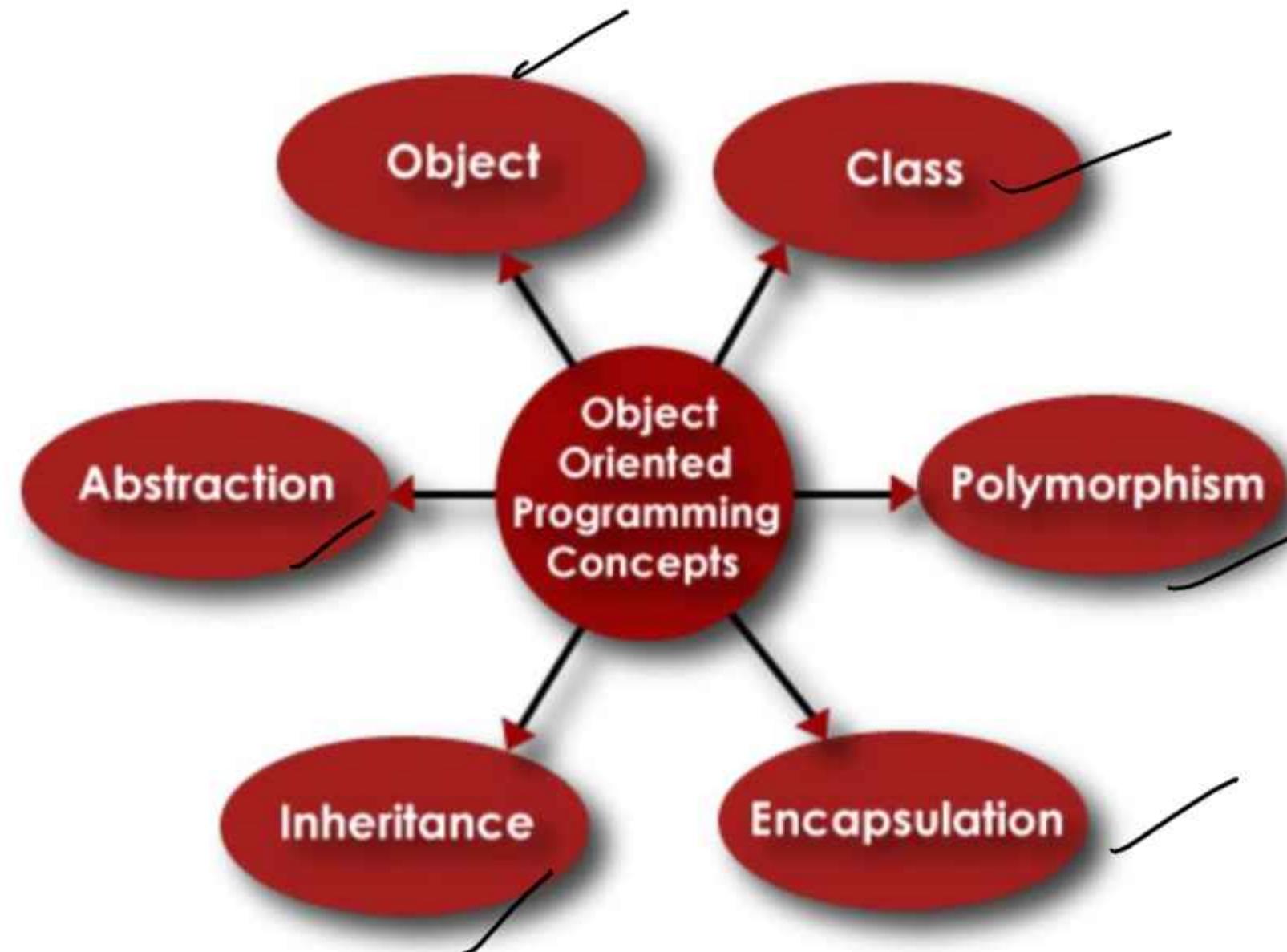
4: Data Structure Oriented Design

- Data Structure Oriented Design (DSOD) एक ऐसी सॉफ्टवेयर डिज़ाइन तकनीक है जिसमें प्रोग्राम को इस आधार पर डिज़ाइन किया जाता है कि डेटा को कैसे स्टोर किया जाएगा और कैसे प्रोसेस किया जाएगा। इसमें सबसे पहले यह तय किया जाता है कि कौन-कौन से डेटा स्ट्रक्चर (जैसे कि Array, Stack, Queue, Linked List, Tree आदि) का इस्तेमाल करना है, फिर उसी के आधार पर प्रोग्राम की लॉजिक और फंक्शन बनाए जाते हैं।
- Data Structure Oriented Design (DSOD) is a software design technique in which a program is designed based on how the data will be stored and processed. In this approach, the first step is to decide which data structures (such as Array, Stack, Queue, Linked List, Tree, etc.) will be used. Based on these choices, the program's logic and functions are then developed accordingly.



5: Object Oriented Design:

- ऑब्जेक्ट-ओरिएंटेड डिज़ाइन (OOD) एक विधि है जिसमें सॉफ्टवेयर सिस्टम को डिज़ाइन करने के लिए उसे ऑब्जेक्ट्स में break किया जाता है, जो कि क्लासेज़ के उदाहरण (instances) होते हैं।
- Object-Oriented Design (OOD) is a method of designing a software system by breaking it down into objects, which are instances of classes.
- ये ऑब्जेक्ट्स आपस में इंटरैक्ट करते हैं ताकि सिस्टम के आवश्यक ऑपरेशन्स को पूरा किया जा सके।
- These objects interact with each other to perform the desired operations of the system.



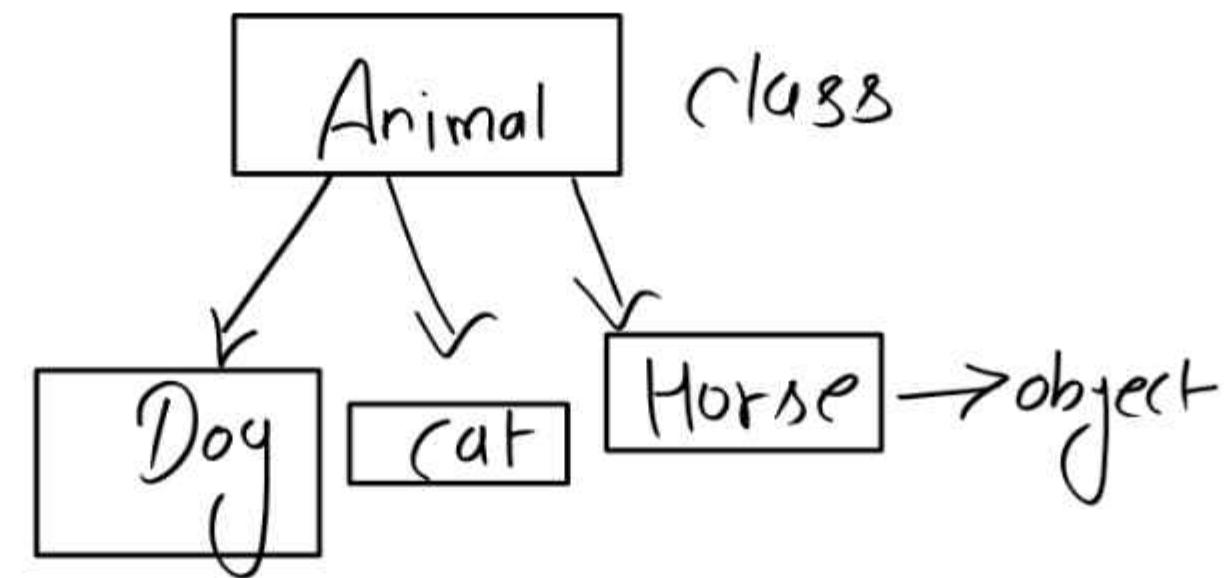


Object:

- डिज़ाइन या प्रोग्राम में शामिल सभी entities को ऑब्जेक्ट कहा जाता है।
- उदाहरण के लिए: कर्मचारी, कंपनी, उपयोगकर्ता, बैंक, व्यक्ति आदि ऑब्जेक्ट्स होते हैं।
- All entities, involved in a design or program are known as objects.
- For example: Employers, company, users, banks, person etc. are objects.

Class:

- क्लास एक blueprint या टेम्प्लेट है, जिससे ऑब्जेक्ट्स बनाए जाते हैं। यह उन attributes और method को परिभाषित करती है जो ऑब्जेक्ट्स में होंगी।
- A class is a blueprint or template from which objects are created. It defines the attributes and methods that the objects created from the class will have.



Attribute: ⇒ colour, leg, ear

function ⇒ sound(), speed()



Abstraction:

- ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग में एब्स्ट्रैक्शन का अर्थ है केवल **essential** या **relevant information** को दिखाना और बाकी **detail** को छिपाना।
- Abstraction in object orient programming means displaying only essential or relevant information and hiding the details

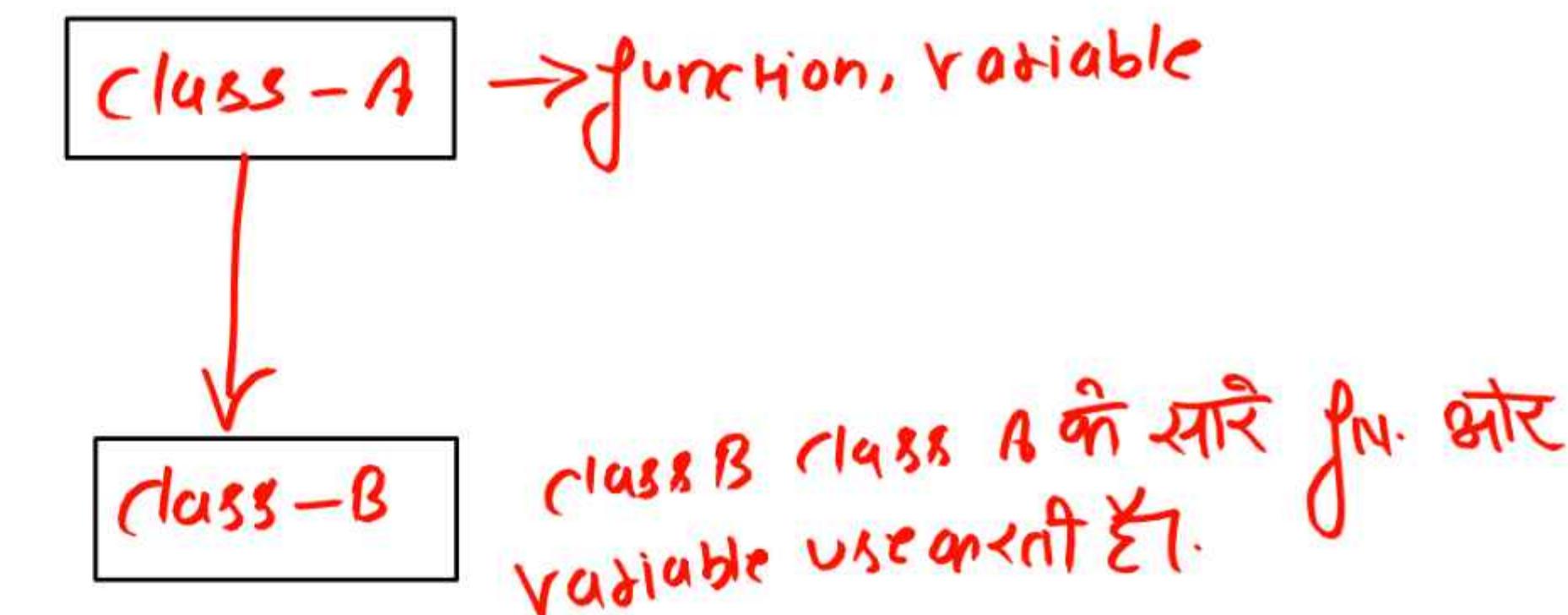
Encapsulation:

- एन्कैप्सुलेशन का मतलब है डेटा (attributes) और उस पर काम करने वाले मेथड्स (functions) को एक ही unit या क्लास में **bundling** करता है
- Encapsulation means bundling the data (attributes) and methods (functions) that operate on the data into a single unit or class.



inheritance:

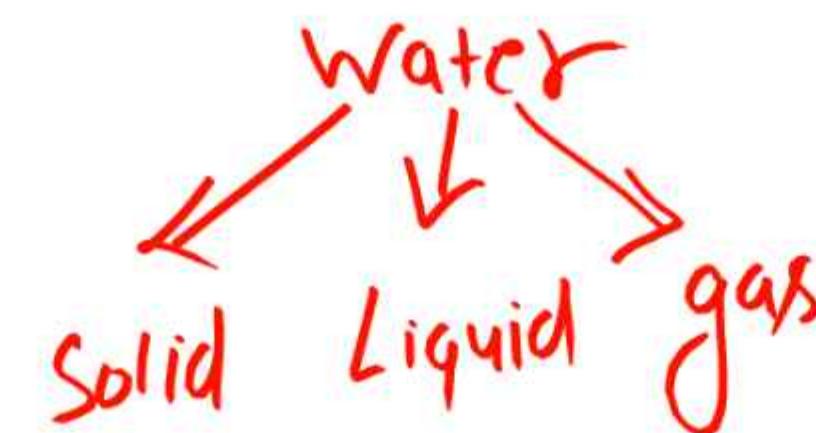
- इस mechanism में एक क्लास दूसरी क्लास की property को प्राप्त कर सकती है। उदाहरण के लिए, जैसे एक बच्चा अपने माता-पिता की property को विरासत में प्राप्त करता है।
- In this mechanism one class can acquire property of another class. For example child inherits properties of his/her parent.





Polymorphism:

- "पॉली" शब्द का अर्थ है many और "मॉफर्स" का अर्थ है form, इसलिए इसका अर्थ है Many Form रूप।
- The word “poly” means many and “morphs” means forms, So it means many forms.
- हम Polymorphism को एक message को एक से अधिक form में represent करने की ability के रूप में define कर सकते हैं।
- we can define Polymorphism as the ability of a message to be displayed in more than one form.



Chapter No-1 (Software Testing)

Q 1: Define software engineering. What are the characteristics of the software?

Q 1: सॉफ्टवेयर इंजीनियरिंग को परिभाषित करें। सॉफ्टवेयर की विशेषताएं क्या हैं?

Q 2: What are the two types of software product? Explain.

Q 2: सॉफ्टवेयर उत्पाद के दो प्रकार क्या हैं? समझाइए।

Q 3: Define High Level Language and its advantages in Software Engineering.

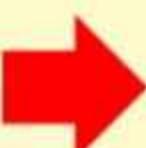
Q3: उच्च स्तरीय भाषा को परिभाषित करें और सॉफ्टवेयर इंजीनियरिंग में इसके लाभ बताएं।

Q 4: Define Object Oriented Programming. Define Encapsulation and Polymorphism.

Q 4: ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग को परिभाषित करें। एनकैप्सुलेशन और पॉलीमॉर्फिज्म को परिभाषित करें।



Chapter-2 : Software Life Cycle Models



Syllabus :

Requirement of Life Cycle Model, Classic Waterfall Model, Prototyping Model, Evolutionary Model, Spiral Model, introduction to agile methodology.

Comparison of different Life Cycle Models



→ Requirement of Life Cycle Model:

✓ Structured Development (संरचित विकास):

- यह सॉफ्टवेयर विकास को चरणों में बाँटता है – जैसे कि Planning, Analysis, Design, Coding, Testing, Deployment और Maintenance – जिससे पूरा काम योजना अनुसार होता है।
- It divides software development into phases – such as Planning, Analysis, Design, Coding, Testing, Deployment, and Maintenance – so that everything works according to plan.

✓ Better Project Management (बेहतर प्रोजेक्ट प्रबंधन):

- प्रत्येक चरण की स्पष्ट पहचान होती है, जिससे Project Manager को योजना बनाना, मॉनिटर करना और समय पर पूरा करना आसान होता है।
- Each phase is clearly identified, making it easier for the Project Manager to plan, monitor, and complete it on time.



Improved Quality (बेहतर गुणवत्ता):

- Testing और Verification के चरणों से सॉफ्टवेयर की Quality सुनिश्चित होती है।
- The quality of the software is ensured by the testing and verification steps.

Customer Satisfaction (ग्राहक संतुष्टि):

- Clear Requirement Gathering और Feedback Loops से ग्राहक की ज़रूरतों को बेहतर तरीके से पूरा किया जा सकता है।
- Clear Requirement Gathering and Feedback Loops allow for better meeting of customer needs.

Risk Management (जोखिम प्रबंधन):

- हर चरण में संभावित जोखिमों की पहचान कर उनका समाधान किया जा सकता है।
- At every stage, potential risks can be identified and addressed



Efficient Resource Utilization (संसाधनों का कुशल उपयोग):

- सही योजना और टाइमलाइन से manpower, tools और budget का सही उपयोग होता है।
- Proper planning and timeline ensures proper utilization of manpower, tools and budget.

Maintenance और Upgradation आसान होता है:

- Life cycle के अंतिम चरणों में maintenance और future upgrades को आसानी से manage किया जा सकता है।
- Maintenance and future upgrades can be easily managed in the later stages of the life cycle.



→ Software Development Life Cycle Model(SDLC):

- सॉफ्टवेयर डेवलपमेंट लाइफ साइकिल (SDLC) एक प्रक्रिया है जिसका उपयोग सॉफ्टवेयर इंडस्ट्री में सॉफ्टवेयर एप्लिकेशन के डिज़ाइन(Design), विकास(Develop) और परीक्षण(Test) के लिए किया जाता है। इसमें सॉफ्टवेयर के विकास की विभिन्न अवस्थाएँ (Stage) शामिल होती हैं।
- The Software Development Life Cycle (SDLC) is a process used in the software industry to design, develop, and test software applications. It includes various stages of software development.



Software Development Life Cycle Model(SDLC):





Phases:

(1:) Requirement Gathering and Analysis:

- इस phase में, अंतिम उपयोगकर्ताओं(end user) या हितधारकों(stockholder) की आवश्यकताओं और अपेक्षाओं(expectation)को एकत्र किया जाता है और उनका documentation किया जाता है।
- In this phase, the needs and expectations of the end users or stakeholders are gathered and documented.

(2:) System Design:

- एक बार आवश्यकताएँ स्पष्ट हो जाने के बाद, सिस्टम की architecture और डिज़ाइन तैयार किया जाता है। इसमें डेटा मॉडल, इंटरफ़ेस और वर्कफ़्लो बनाना शामिल है ताकि यह निर्धारित किया जा सके कि सिस्टम कैसे काम करेगा।
- Once the requirements are clear, the system's architecture and design are prepared. This includes creating data models, interfaces, and workflows to establish how the system will function.



Phases:

(3:) Implementation (Coding/Development):

- यह वह phase है जहाँ वास्तविक कोडिंग होती है। डेवलपर्स सिस्टम डिज़ाइन document आधार पर कोड लिखते हैं।
- This phase is where actual coding happens. Developers write code based on the system design documents.

(4:) Testing:

- विकास(development) के बाद, सॉफ्टवेयर परीक्षण(testing) का उपयोग दोषों(defects), बगों और अन्य मुद्दों(issue) को खोजने के लिए किया जाता है। यह सुनिश्चित करता है कि सॉफ्टवेयर आवश्यकताओं को पूरा करता है और ठीक से काम करता है
- After development, the software testing is used to find defects, bugs, and other issues. This ensures the software meets the requirements and works as.



Phases:

(5:) Deployment:

- एक बार जब सॉफ्टवेयर का परीक्षण(testing) और अनुमोदन(approved) हो जाता है, तो इसे production वातावरण में तैनात(deployed) किया जाता है जहाँ उपयोगकर्ता इसे एक्सेस और उपयोग कर सकते हैं।
- Once the software has been tested and approved, it is deployed to the production environment where users can access and use it.

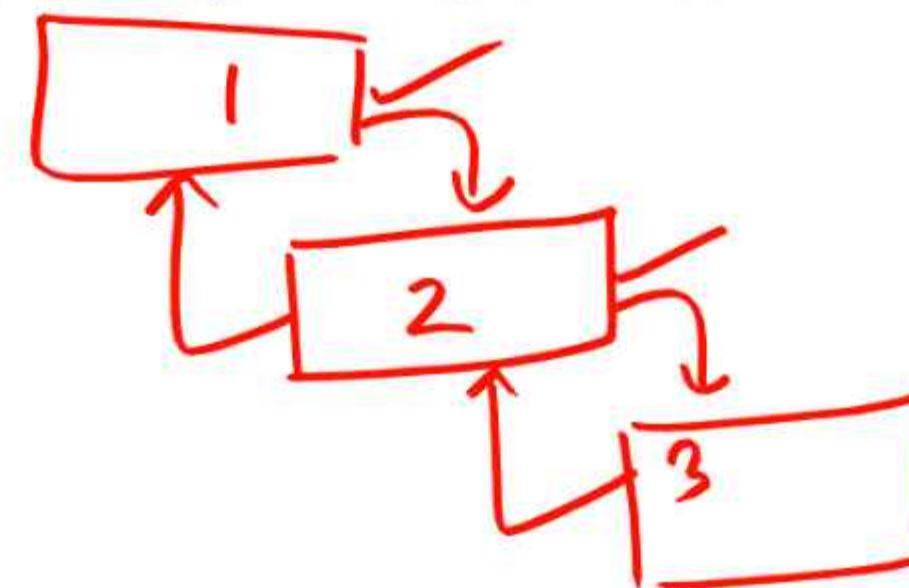
(6:) Maintenance:

- तैनाती(deployment) के बाद, सॉफ्टवेयर रखरखाव(maintenance) चरण में प्रवेश करता है। इस चरण में, लाइव वातावरण में पाए जाने वाले किसी भी दोष(defect) को ठीक किया जाता है, और आवश्यकतानुसार अपडेट या enhancements लागू किए जाते हैं।
- After deployment, the software enters the maintenance phase. In this phase, any defects found in the live environment are fixed, and updates or enhancements are applied as needed.



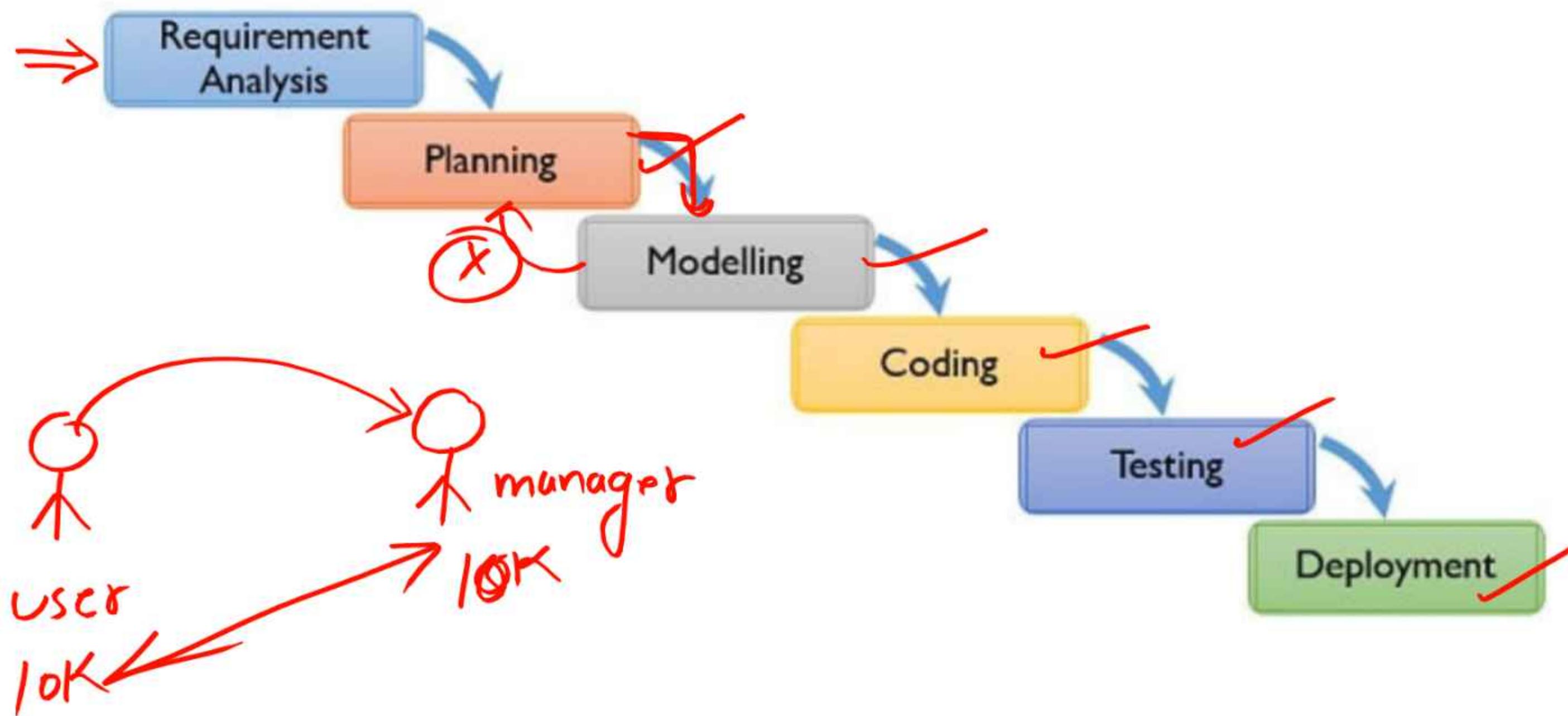
Waterfall Model:

- यह एक रेखीय(linear) और अनुक्रमिक(sequential) approach का पालन करता है, जहां project को अलग-अलग चरणों में विभाजित किया जाता है, और अगले चरण पर जाने से पहले प्रत्येक चरण को पूरा किया जाना चाहिए। चरणों के बीच कोई **ओवरलैप नहीं है**, जिससे इसे समझना और प्रबंधित करना आसान हो जाता है।
- It follows a linear and sequential approach, where the project is divided into distinct phases, and each phase must be completed before moving on to the next. There is no overlap between phases, which makes it easy to understand and manage.**





Waterfall Model:





→ Advantage Of Waterfall Model:

- Simple and easy to use.
- सरल और इस्तेमाल करने में आसान।
- Well-defined, fixed requirements minimize changes during development.
- अच्छी तरह से तय की गई आवश्यकताएँ विकास के दौरान बदलाव को कम करती हैं
- Effective for smaller projects with clear objectives.
- स्पष्ट लक्ष्यों(objective) वाले छोटे प्रोजेक्ट्स के लिए प्रभावी।
- Structured approach to quality control and testing.
- गुणवत्ता(quality) नियंत्रण(control) और परीक्षण(testing) के लिए व्यवस्थित तरीका।



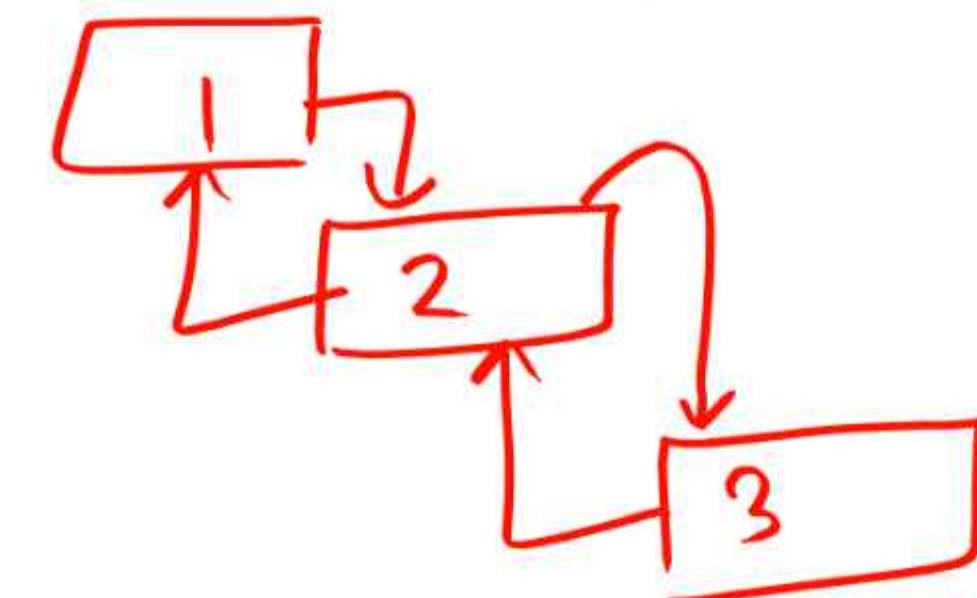
→ Disadvantage Of Waterfall Model:

- Difficult to make changes once development has started.
• एक बार काम शुरू हो जाने के बाद बदलाव करना कठिन हो जाता है। ✓
- Testing happens late, leading to delayed error detection.
• टेस्टिंग देर से होती है, जिससे गलतियों का पता लगाने में देरी होती है। ✓
- Not ideal for large, complex projects with evolving requirements.
• बड़े और मुश्किल प्रोजेक्ट्स के लिए यह तरीका सही नहीं है, खासकर जहाँ ज़रूरतें बदलती रहती हैं। ✓
- Limited customer feedback during the development process.
• प्रोजेक्ट के दौरान ग्राहक की राय कम मिलती है। ✓



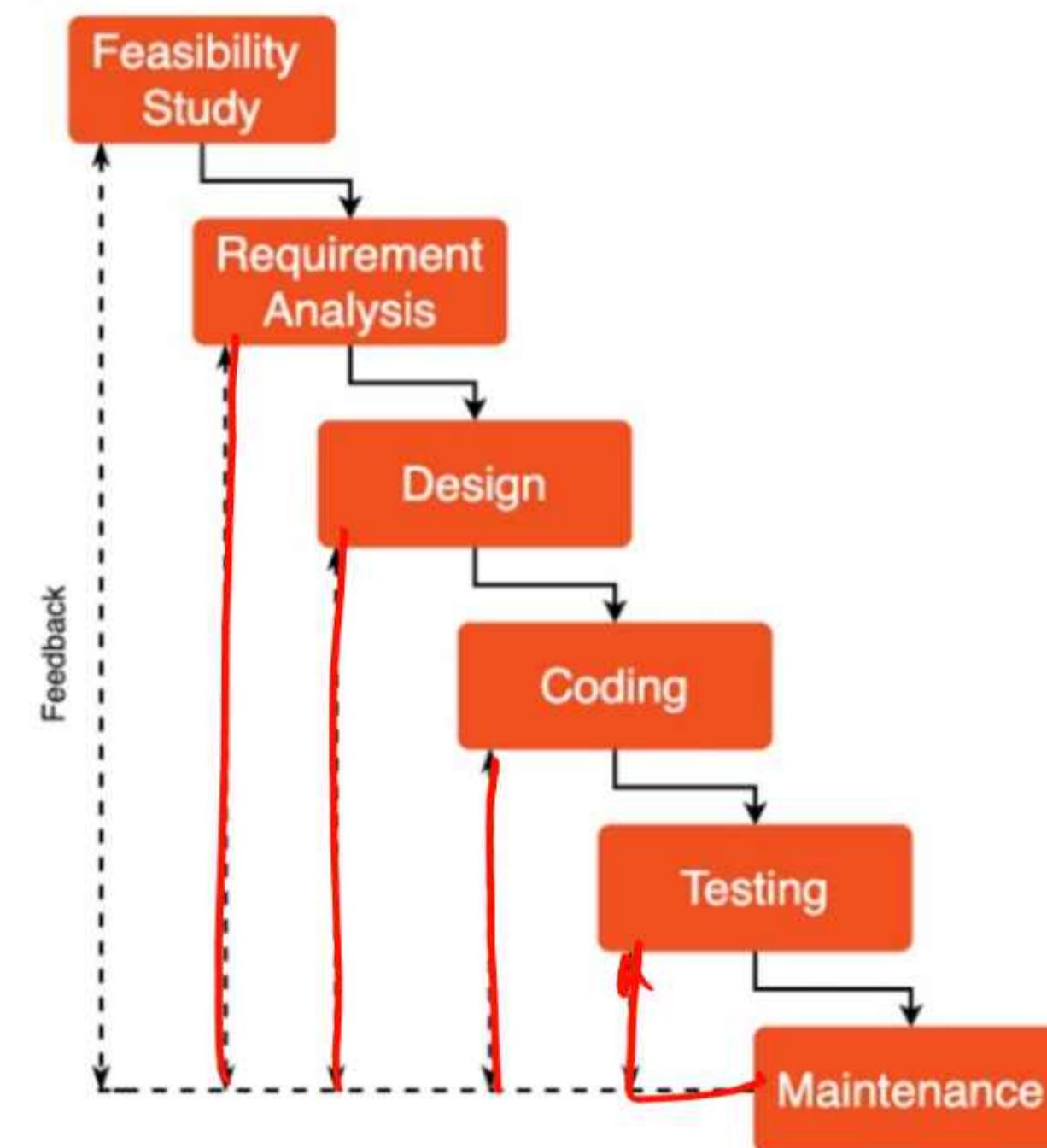
Iterative Waterfall Model:

- इस मॉडल में, चरण (phase) अभी भी एक क्रम(sequence) में होते हैं, लेकिन यदि आवश्यक हो तो पिछले चरण (phase) में वापस जाने में अधिक लचीलापन (flexible) होता है। उदाहरण के लिए, यदि परीक्षण (testing) में कोई समस्या सामने आती है, तो टीम आवश्यक समायोजन (adjustments) करने के लिए डिज़ाइन चरण (phase) में वापस आ सकती है।
- In this model, the phases still occur in a sequence, but there is more flexibility in moving back to a previous phase if needed. For example, if testing reveals a problem, the team can return to the design phase to make necessary adjustments.





Iterative Waterfall Model:





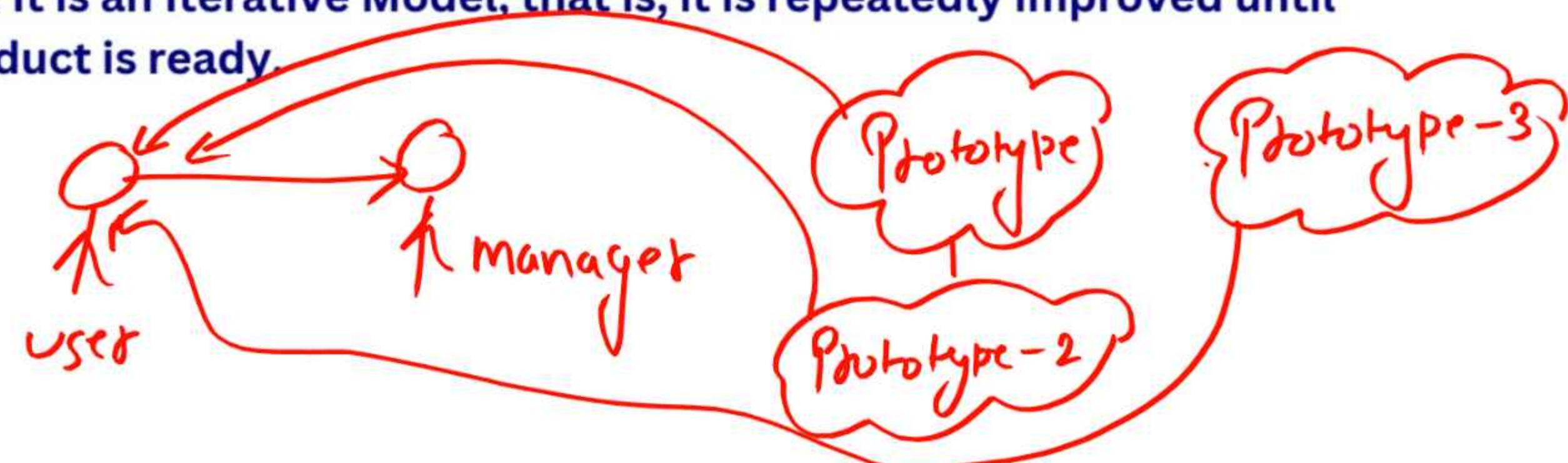
→ Advantage Iterative Waterfall Model:

- Flexibility to make changes between phases.
- चरणों के बीच बदलाव करना आसान होता है।
- Early detection and correction of errors.
- गलतियों का जल्दी पता लगाकर सही किया जा सकता है।
- Better risk management.
- Risk management बेहतर होता है।
- Increased customer feedback during the project.
- प्रोजेक्ट के दौरान ग्राहक से अधिक फीडबैक मिलता है।
- Combines clear structure with flexibility.
- Clear structure के साथ लचीलापन(flexibility)मिलता है।



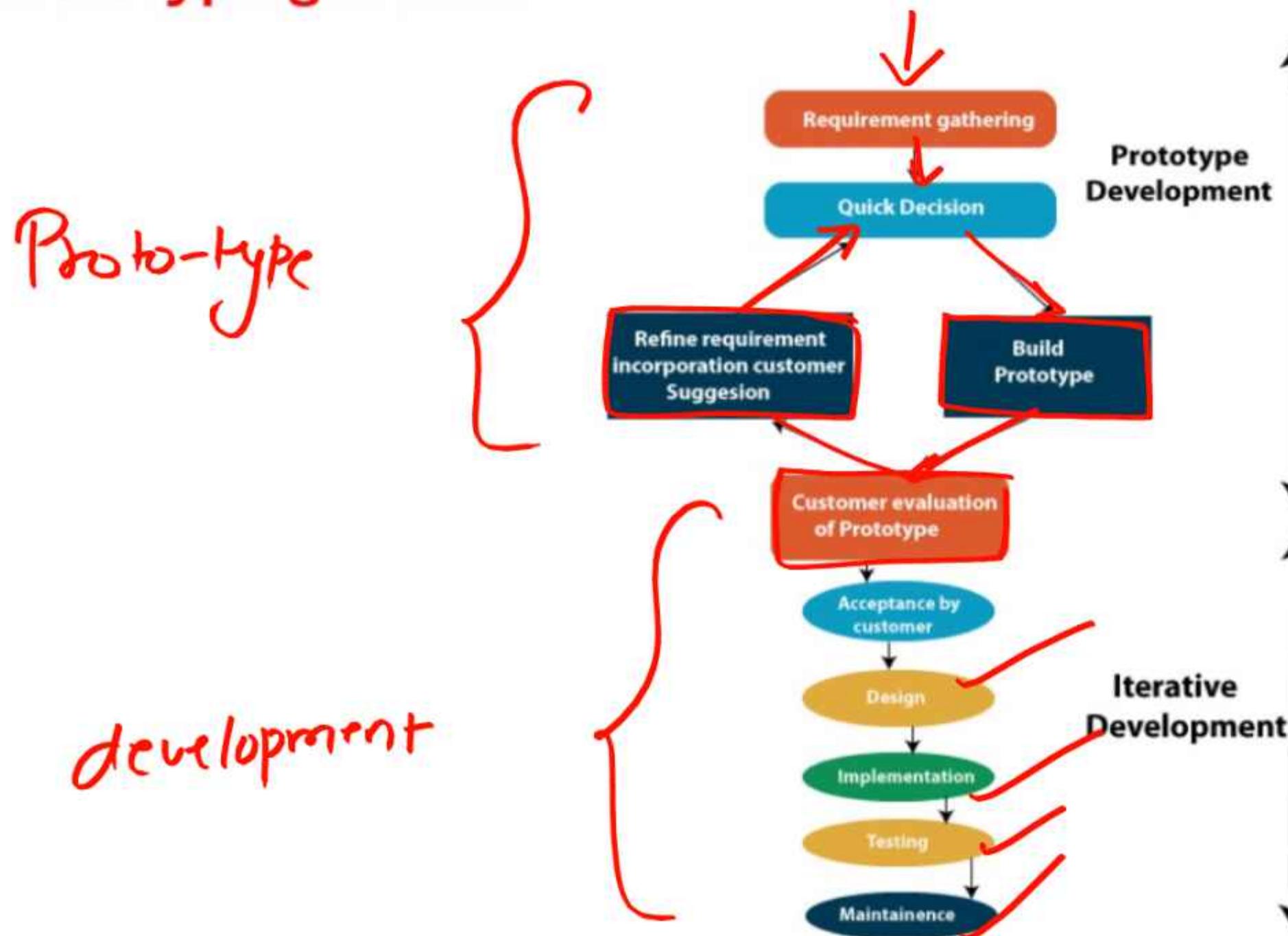
Prototyping Model:

- प्रोटोटाइप मॉडल एक सॉफ्टवेयर विकास प्रक्रिया (Software Development Process) है जिसमें एक प्रारंभिक कार्यशील मॉडल (Prototype) बनाया जाता है ताकि ग्राहक (Client) की आवश्यकताओं को अच्छे से समझा जा सके। यह एक Iterative Model है, यानी इसमें बार-बार सुधार किया जाता है जब तक अंतिम उत्पाद तैयार न हो जाए।
- The prototype model is a software development process in which an initial working model (Prototype) is created to better understand the requirements of the client. It is an Iterative Model, that is, it is repeatedly improved until the final product is ready.





Prototyping Model:





Phases:

(1): Requirement Gathering and Analysis:

- In this phase, the needs and expectations of the end users or stakeholders are gathered and documented.
- इस phase में, अंतिम उपयोगकर्ताओं(end user) या हितधारकों(stockholder) की आवश्यकताओं और अपेक्षाओं(expectation)को एकत्र किया जाता है और उनका documentation किया जाता है।

(2): Developing the Prototype:

- A prototype is created based on the gathered requirements. This prototype is a working model of the software but may not have all features fully developed. It serves to demonstrate core functionalities.
- एक प्रोटोटाइप को एकत्रित(gathered) की गई आवश्यकताओं के आधार पर बनाया जाता है। यह प्रोटोटाइप सॉफ्टवेयर का एक कार्यशील मॉडल होता है, लेकिन इसमें सभी विशेषताएँ पूरी तरह से विकसित नहीं हो सकती हैं। इसका उद्देश्य मुख्य कार्यक्षमताओं(functionality)को दिखाना होता है।



Phases:

(3) User Feedback:

- Users interact with the prototype and provide feedback. This step is important as it helps identify what works well and what needs improvement.
- उपयोगकर्ता(user) प्रोटोटाइप के साथ इंटरैक्ट करते हैं और फीडबैक देते हैं। यह कदम बहुत महत्वपूर्ण है क्योंकि इससे पता चलता है कि क्या सही है और क्या सुधारने की ज़रूरत है।

X (4) Refinement:

- Users interact with the prototype and provide feedback. This step is important as it helps identify what works well and what needs improvement.
- उपयोगकर्ता(user) प्रोटोटाइप के साथ इंटरैक्ट करते हैं और फीडबैक देते हैं। यह कदम बहुत महत्वपूर्ण है क्योंकि इससे पता चलता है कि क्या सही है और क्या सुधारने की ज़रूरत है।



Advantage of Prototyping Model:

- विकास(development) का समय कम करता है।
- विकास(development) की लागत(cost) कम करता है।
- उपयोगकर्ता की भागीदारी(invovlment) की आवश्यकता होती है।
- खोई हुई कार्यक्षमताएँ (functionality) आसानी से पहचानी जा सकती हैं।
- उपयोगकर्ता(user) की satisfaction बढ़ाता है।
- Reduces development time.
- Reduces development cost.
- Requires user involvement.
- Missing functionality can be identified easily.
- Results in higher user satisfaction.



Disadvantage of Prototyping Model:

- असली प्रोजेक्ट शुरू करने में देरी होती है।
- निवेश(cost) अधिक होता है।
- समय ज्यादा लगता है।
- क्लाइंट की भागीदारी हमेशा डेवलपर्स के लिए सही नहीं होती।
- **Delay in starting the real project**
- **Total investment is more**
- **Total time taken is more**
- **Involvement by client is not always perfect by the developers**

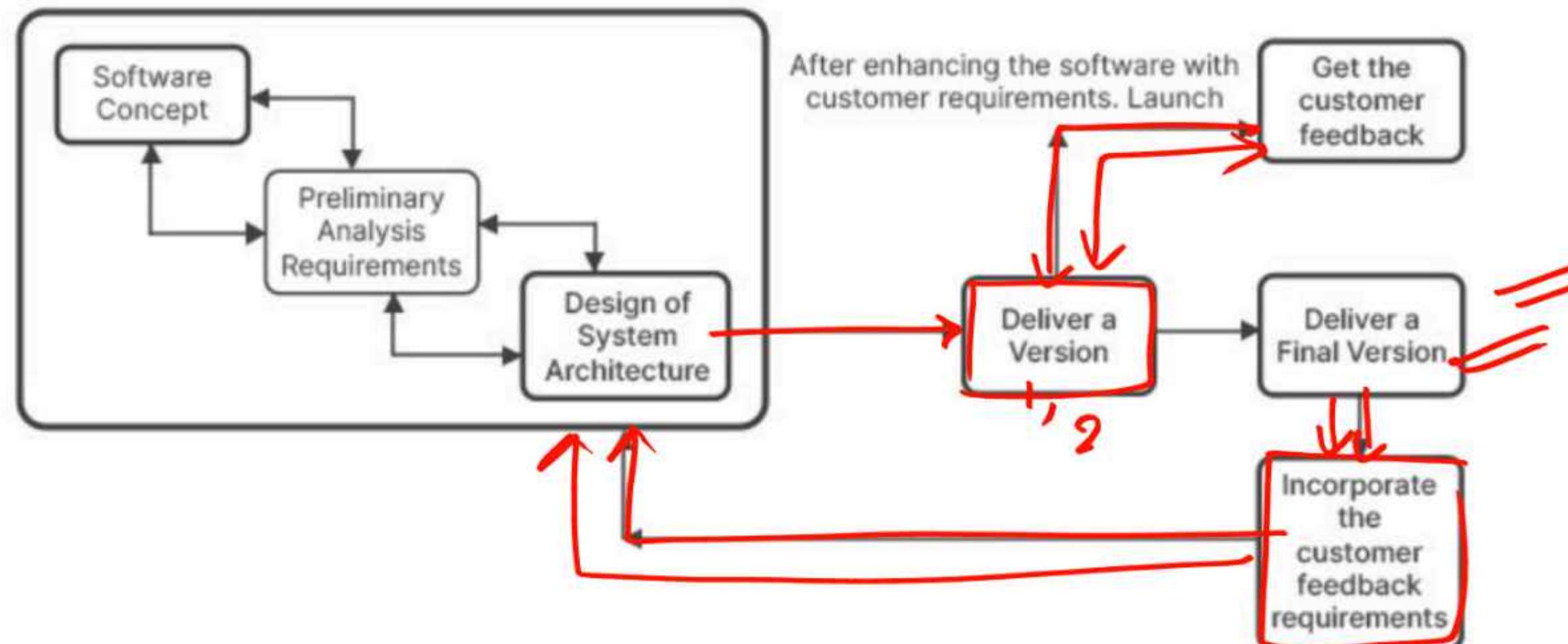


→ Evolutionary Model:

- Evolutionary Model एक सॉफ्टवेयर विकास प्रक्रिया है जिसमें सॉफ्टवेयर को धीरे-धीरे (step by step) बनाया और सुधारा जाता है, जब तक कि वह पूरी तरह से विकसित और उपयोग के योग्य न हो जाए। इसमें हर चरण के बाद उपयोगकर्ता से फीडबैक लेकर नए फीचर्स जोड़े या पुराने सुधार किए जाते हैं।
- Evolutionary Model is a software development process in which software is created and improved gradually (step by step) until it is fully developed and usable. In this, new features are added or old improvements are made by taking feedback from the user after every stage.
- यह मॉडल Prototype Model + Iterative Development का मिश्रण है।
- This model is a mixture of Prototype Model + Iterative Development.



Evolutionary Model:





Advantage Of Evolutionary Model:

- जोखिम(risk) analysis अच्छा है।
- यह बदलती ज़रूरतों को संभालता है।
- शुरुआत में काम करने का समय कम होता है।
- बड़े और जरूरी प्रोजेक्ट्स के लिए सही है।
- गलतियाँ जल्दी पकड़ी जा सकती हैं।
- जल्दी फीडबैक मिलता है, जिससे बेहतर समाधान होता है।
- खोई हुई सुविधाएँ आसानी से पता चलती हैं।
- Risk analysis is better.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.
- Errors can be detected much easier.
- Quicker user feedback is available leading to better solution.
- Missing functionality can be identified easily



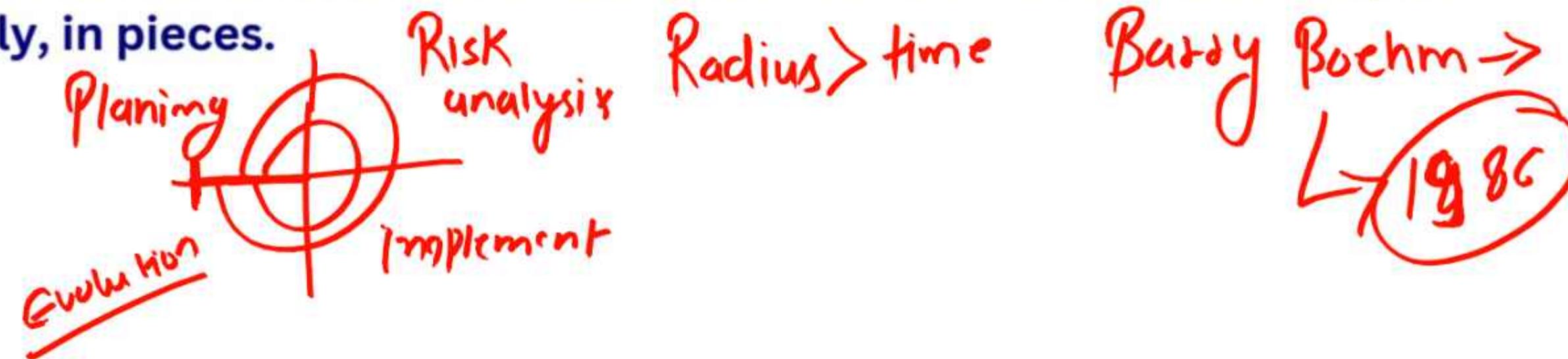
Disadvantage Of Evolutionary Model:

- छोटे प्रोजेक्ट्स के लिए सही नहीं है।
- प्रबंधन करना मुश्किल है।
- इस्तेमाल करना महंगा हो सकता है।
- जोखिम(risk) विश्लेषण(analysis) के लिए ज्यादा Skilled लोग चाहिए।
- प्रोजेक्ट की प्रगति जोखिम(risk) विश्लेषण(analysis) पर निर्भर करती है।
- Not suitable for smaller projects.
- Management complexity is more.
- Can be costly to use.
- Highly skilled resources are required for risk analysis.
- Project's progress is highly dependent upon the risk analysis phase



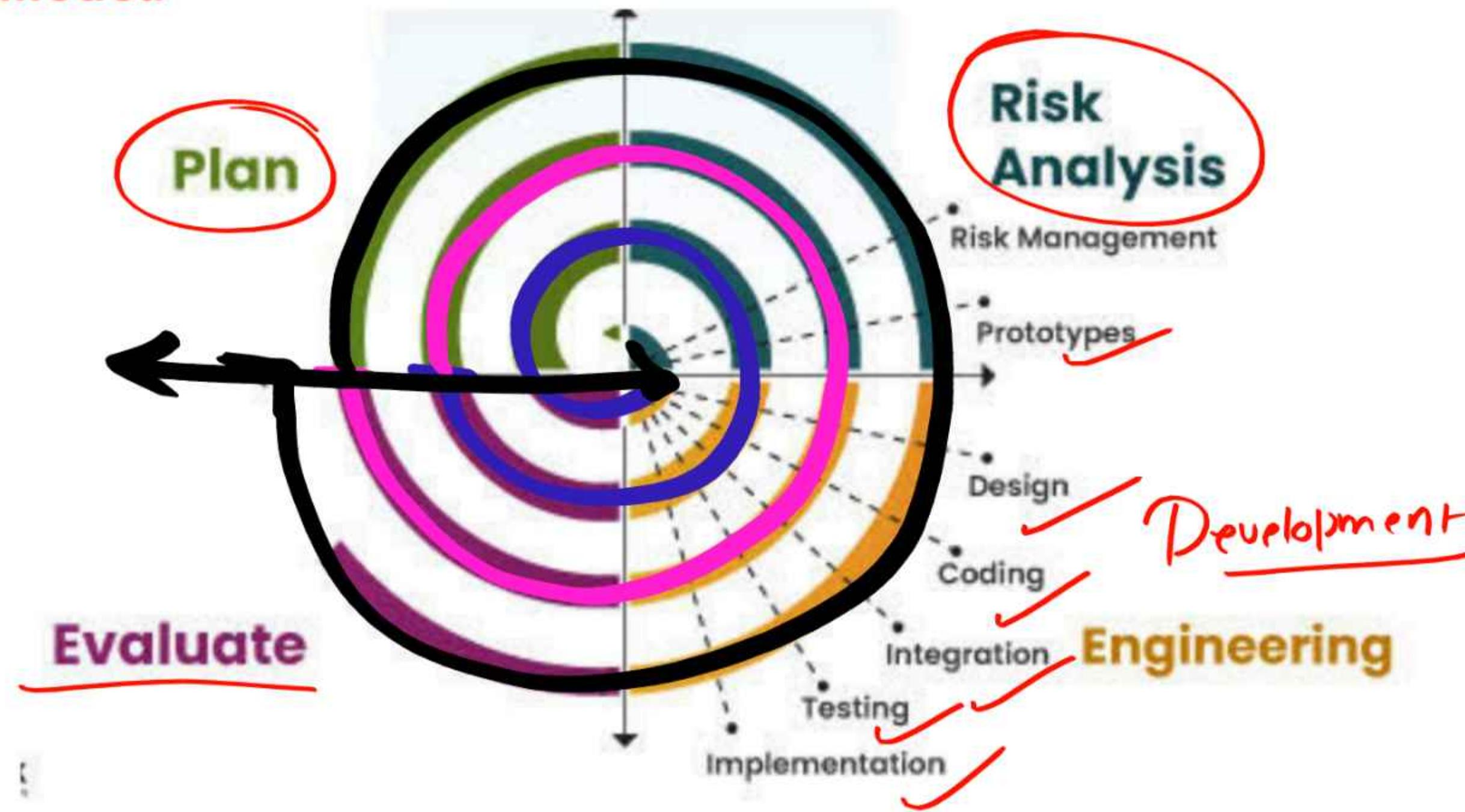
Spiral Model:

- स्पाइरल मॉडल सॉफ्टवेयर डेवलपमेंट की एक प्रक्रिया(process) है, जिसमें सॉफ्टवेयर को धीरे-धीरे विकसित(developed) किया जाता है। इसे “स्पाइरल” (घुमावदार) इसलिए कहा जाता है क्योंकि इसमें बार-बार एक जैसी स्टेप्स (चरणों) को दोहराते हुए आगे बढ़ते हैं
- The spiral model is a process of software development in which software is developed gradually. It is called "spiral" because in it one moves forward by repeating the same steps again and again.
- इस मॉडल में सॉफ्टवेयर को एक बार में पूरा नहीं किया जाता, बल्कि टुकड़ों में, धीरे-धीरे विकसित किया जाता है।
- In this model the software is not completed all at once but is developed gradually, in pieces.





→ Spiral Model:





Advantage Of Spiral Model:

- जोखिम को संभालने में मदद करता है
- बड़े प्रोजेक्ट्स के लिए उपयुक्त(suitable) होता है
- आवश्यकताओं में लचीलापन होता है
- ग्राहक संतुष्टि रहता है
- मेटा मॉडल के रूप में जाना जाता है।

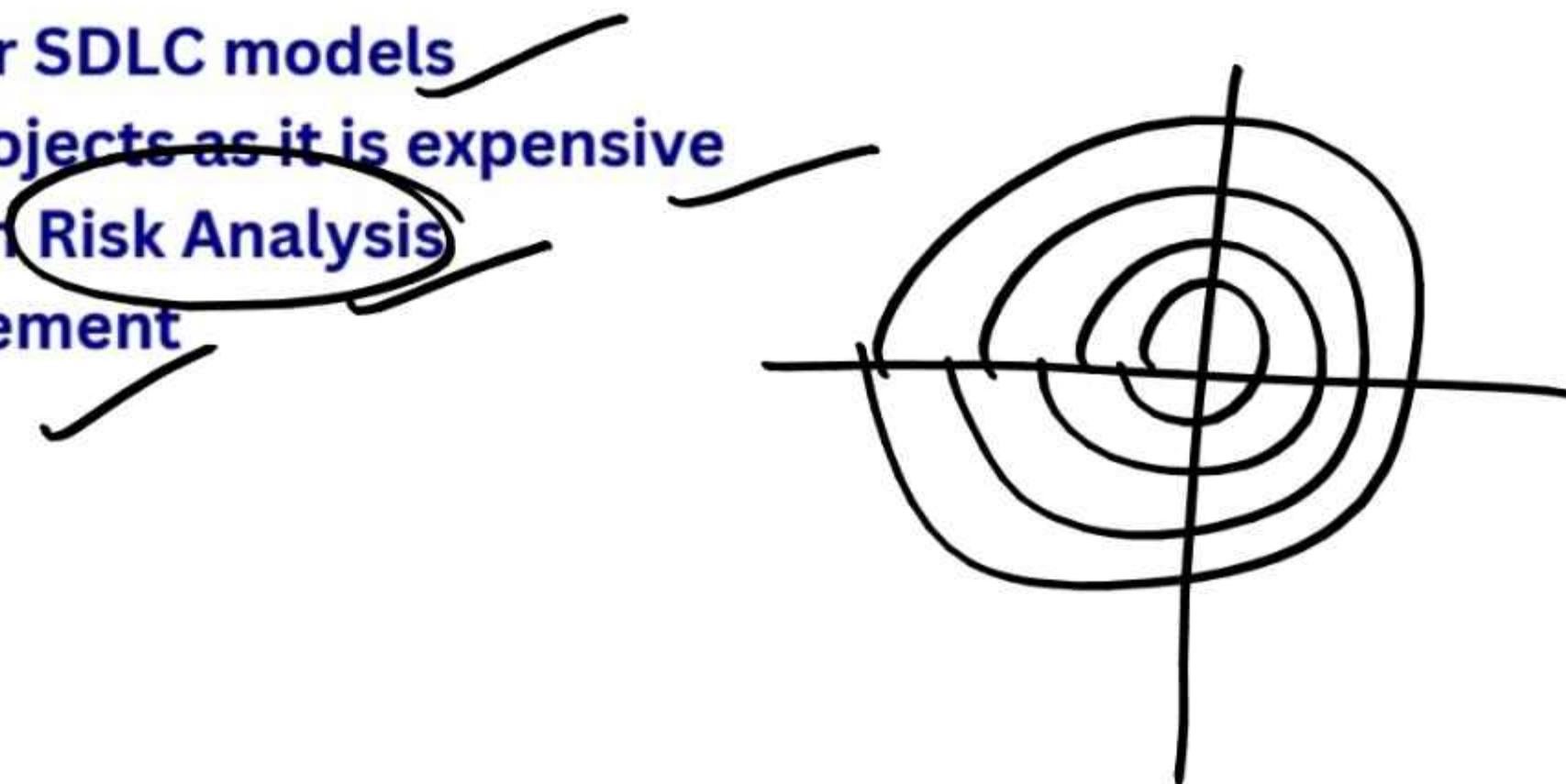
• Supports Risk Handling.

- Best suited for large projects.
- Flexibility in Requirements.
- Customer Satisfaction.
- known as Meta model .



Disadvantage Of Spiral Model:

- दूसरे SDLC मॉडल्स से ज्यादा जटिल(complex) होता है
 - छोटे प्रोजेक्ट्स के लिए महंगा होने के कारण सही नहीं है
 - जोखिम विश्लेषण(risk analysis) पर बहुत ज्यादा निर्भर करता है
 - समय का प्रबंधन(management) करना मुश्किल होता है
-
- More complex than other SDLC models
 - Not suitable for small projects as it is expensive
 - Too much dependable on Risk Analysis
 - Difficulty in time management



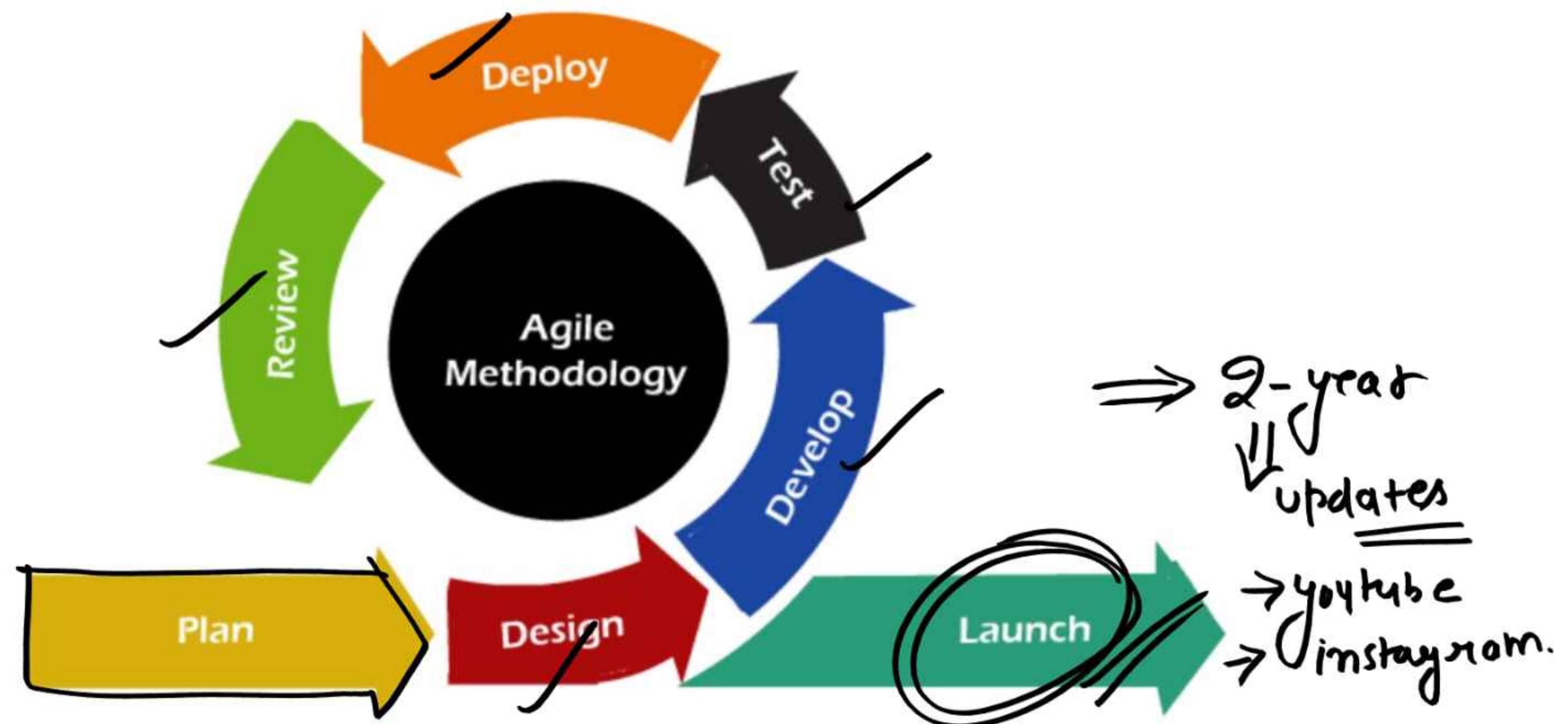


→ **Introduction to agile methodology:**

- Agile METHODOLOGY is continuous Iteration (Iteration means repetition of process) of development and testing throughout the software development life cycle of the project.
- Agile Methodology एक ऐसी प्रक्रिया है जिसमें सॉफ्टवेयर डेवलपमेंट और टेस्टिंग को बार-बार (Iteration का मतलब है प्रक्रिया को दोहराना) किया जाता है। यह पूरा प्रोजेक्ट बनने तक चलता रहता है।
- In this method development and testing activities are running together or are concurrent.
- इस तरीके में सॉफ्टवेयर बनाने और उसकी टेस्टिंग साथ-साथ होती रहती है, यानी दोनों काम एक ही समय पर चलते हैं।



→ Introduction to agile methodology:





Advantage of agile model:

- यह सभी तरीकों में सबसे प्रभावी है।
- इसे मैनेज करना आसान है।
- प्रक्रिया के किसी भी चरण(stage) में बदलाव आसानी से किए जा सकते हैं।
- यह डेवलपर्स को ज्यादा लचीलापन(flexibility) देता है।
- इसमें कम संसाधनों(resource) की जरूरत होती है।
- It is very effective among all other methods.
- Easy to manage.
- Changes at any stage of process are catered easily.
- Provide more flexibility to developers.
- Limits resources required.



Disadvantage of agile model:

- इसे छोटे प्रोजेक्ट्स के लिए इस्तेमाल नहीं किया जा सकता।
- ग्राहक की बदलती ज़रूरतों के कारण प्रोजेक्ट गलत दिशा में जा सकता है।
- क्योंकि यह कई चक्रों (साइकल्स) में चलता है, इसलिए प्रगति(progress) को मापना मुश्किल होता है।
- Cannot be used for small development projects.
- Project may be go out of track due to changing customer requirement.
- It is difficult to measure progress as process is going on multiple cycles.



Software Life Cycle Models का तुलनात्मक अध्ययन (Comparison of SDLC Models):

विशेषता / Model	Waterfall Model	Iterative Model	Spiral Model	Agile Model
Structure (संरचना)	Linear & Sequential	Repetitive Cycles	Spiral Phases with Risk Analysis	Incremental & Iterative
Flexibility in Requirement (लचीलापन)	✗ नहीं/ No	✓ हाँ/ Yes	✓ हाँ/ Yes	✓ बहुत ज्यादा/ High
Risk Management (जोखिम प्रबंधन)	✗ Poor	⚠ Moderate	✓ Excellent	⚠ Moderate
User Involvement (यूज़र की भागीदारी)	✗ कम/ Less	✓ अधिक/ More	✓ अधिक/ More	✓ बहुत ज्यादा/ Much



विशेषता / Model	Waterfall Model	Iterative Model	Spiral Model	Agile Model
Feedback Integration (प्रतिक्रिया लेना)	✗ नहीं / NO ✓	✓ हाँ / Yes ✓	✓ हाँ / Yes ✓	✓ हाँ (Regular) ✓
Best Suited For (उपयुक्त क्षेत्र)	छोटे और सरल प्रोजेक्ट (Small & Easy Projects) ✓	बड़े projects जिनमें requirement change हो सकती है ✓	Complex और high-risk projects ✓	Fast-moving और बदलती जरूरतों वाले प्रोजेक्ट ✓
Development Speed	🚶 Slow ✓	🚶♂ Medium ✓	🚶♂ Medium ✓	🏃♂ Fast ✓
Cost	₹ Low ✓	₹ Medium ✓	₹ High ✓	₹ Medium ✓

Chapter No-2 (Software Life Cycle Model)

Q 1: Explain in detail Software Development life cycle?

Q 1: सॉफ्टवेयर विकास जीवन चक्र के बारे में विस्तार से बताएं?

Q 2: What are the Spiral Model? Explain in detail?

Q 2: Spiral मॉडल क्या हैं? विस्तार से समझाइए।

Q 3: Explain prototype model with its advantages and disadvantages.

Q 3: प्रोटोटाइप मॉडल को इसके फायदे और नुकसान के साथ समझाएँ।

Q 4: Define Agile Methodology. Explain its advantages and disadvantages.

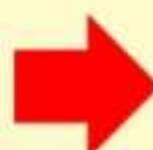
Q 4: एजाइल कार्यप्रणाली को परिभाषित करें। इसके लाभ और हानियाँ समझाएँ।

Q 5: What are the waterfall Model? Explain in detail?

Q 5: waterfall मॉडल क्या हैं? विस्तार से समझाइए।



Chapter-3 : Software Planning.

**Syllabus :**

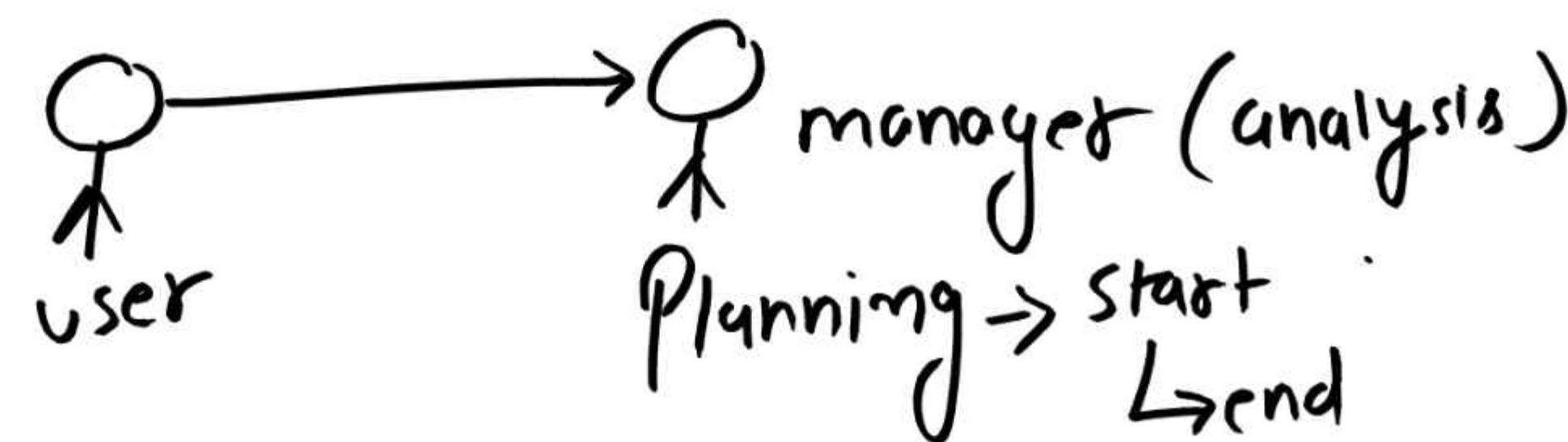
- (X) Responsibilities of Software Project Manager
- Metrics for Project Size Estimation- LOC(Lines of Code),
- (X) Function Point Metric
- Project estimation Techniques- Using COCOMO Model.

$C_0 \rightarrow$ constructive } cost
 $C_0 \rightarrow$ cost
 $M_0 \rightarrow$ Model



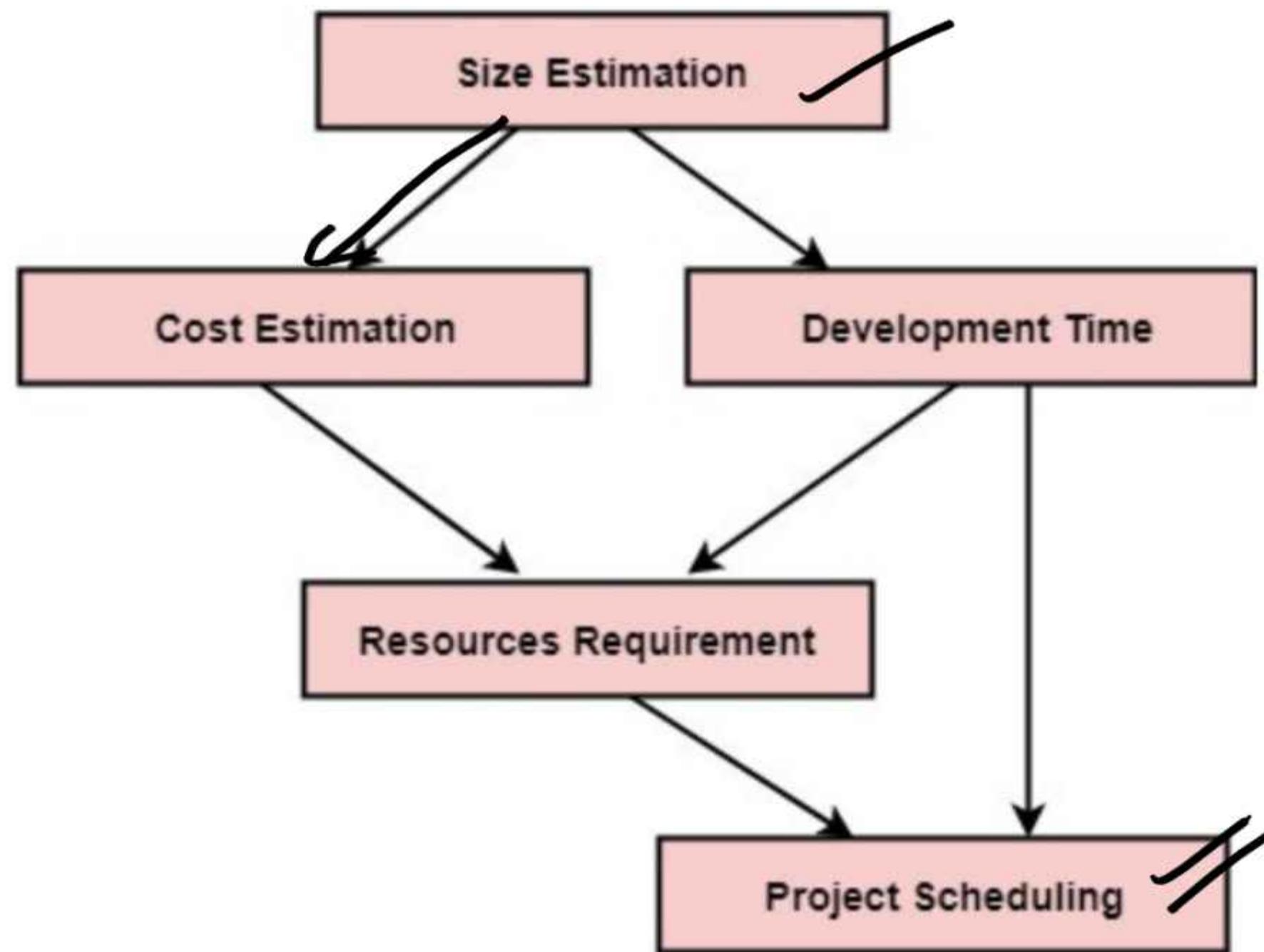
Software Planning:

- सॉफ्टवेयर प्रोजेक्ट प्लानिंग एक प्रक्रिया है जिसमें किसी सॉफ्टवेयर प्रोजेक्ट को शुरू से अंत तक व्यवस्थित किया जाता है। इसमें प्रोजेक्ट का आकार, उसकी लागत, उसे पूरा करने में लगने वाला समय, और महंनत का अनुमान लगाया जाता है।
- Software project planning is the process of organizing and preparing a software project from start to finish. It includes estimating the size of the project, figuring out how much it will cost, how long it will take, and how much effort will be needed to complete it.





Some Of the Project Planning activities are :





(a): Size Estimation of project:

- Project size estimation का मतलब है कि प्रोजेक्ट को पूरा करने में लगभग कितना समय और श्रम (effort) लगेगा। इसका उद्देश्य यह समझना कि
- Project size means how much time and effort will be approximately required to complete the project.

(b): Cost estimation of project:

- Cost estimation का मतलब है प्रोजेक्ट को पूरा करने में कितनी लागत (cost) आएगी, इसका अनुमान लगाना। इसमें सभी खर्च शामिल होते हैं, जैसे कि संसाधन (resources), श्रम (labor), सामग्री (materials), उपकरण (equipment), और अन्य आवश्यकताओं की लागत।
- Cost estimation means predicting how much expense will be incurred in completing the project. It includes all costs, such as resources, labor, materials, equipment, and other necessary requirements.



(c): Duration:

- Duration का मतलब है कि प्रोजेक्ट को पूरा करने में केवल समय के हिसाब से कितना वक्त लगेगा।
- How does it will take to complete the project only in terms of time.

(d): Project Scheduling:

- Project Scheduling का मतलब है किसी प्रोजेक्ट के कामों को समय के अनुसार व्यवस्थित करना। इसमें यह तय किया जाता है कि कौन सा काम कब और कैसे किया जाएगा ताकि प्रोजेक्ट समय पर और सही तरीके से पूरा हो सके।
- Project Scheduling means arranging the tasks of a project according to time. In this, it is decided which work will be done when and how so that the project can be completed on time and correctly.



The role and Responsibilities of a Project Manager: (प्रोजेक्ट मैनेजर की भूमिका और जिम्मेदारियाँ:)

(1): Planning:

- प्रोजेक्ट के उद्देश्यों और लक्ष्यों को निर्धारित करना और सभी कार्यों की योजना बनाना।
- Define the objectives and goals of the project and create a plan for all tasks.

(2): Team Management:

- प्रोजेक्ट टीम का चयन करना और उनके कार्यों का समन्वय(coordinate) करना। टीम के सदस्यों के साथ संवाद(Communicate) करना और उन्हें प्रेरित(motivate) करना।
- Select the project team and coordinate their activities. Communicate with team members and motivate them.



(3): Time Management:

- प्रोजेक्ट के कार्यों के लिए समय सीमा निर्धारित करना और यह सुनिश्चित करना कि सभी कार्य समय पर पूरे हों।
- Establish deadlines for project tasks and ensure that all work is completed on time.



(4): Cost Management:

- प्रोजेक्ट के लिए बजट बनाना और सभी खर्चों पर नज़र रखना, ताकि लागत नियंत्रण में रहे।
- Create a budget for the project and monitor all expenses to keep costs under control.

(5): Communication:

- प्रोजेक्ट से संबंधित सभी जानकारी को stakeholders के साथ साझा करना, जैसे कि क्लाइंट, टीम और प्रबंधन।
- Share all relevant information about the project with stakeholders, including clients, team members, and management.



(6): Problem Solving:

- प्रोजेक्ट के दौरान उत्पन्न होने वाली समस्याओं को हल करना।
- Address any issues that arise during the project.

(7): Quality Management:

10 days → 8 days

- प्रोजेक्ट की गुणवत्ता को सुनिश्चित करने के लिए मापदंडों का निर्धारण करना और उन पर ध्यान देना।
- Set quality parameters for the project and ensure adherence to them.

(8): Final Reporting:

- प्रोजेक्ट के पूरा होने पर रिपोर्ट तैयार करना और प्रोजेक्ट की सफलता का मूल्यांकन करना।
- Prepare a report upon project completion and evaluate the project's success.



Metrics for Project Size Estimation:

- Project size estimation के लिए कुछ प्रमुख metrics होते हैं, जिनसे हम प्रोजेक्ट की जटिलता, समय, और संसाधनों का अनुमान लगा सकते हैं।
- There are some key metrics for project size estimation, through which we can estimate the complexity, time, and resources of the project.

- LOC (Lines of Code). ✓
- Function Points (FP). ✓
- Story Points. ✓
- Use Case Points. ✓
- COCOMO Model (Constructive Cost Model). ✓

↳ Size

↳ cost
↳ time



→ (1): कोड की लाइनें (LOC (Lines of Code)):

- LOC का मतलब है “कोड की लाइनों की संख्या”। जब कोई डेवलपर प्रोग्राम लिखता है, तो वह कई लाइनें लिखता है, जैसे कि इंस्ट्रक्शंस, लॉजिक, फंक्शंस आदि। LOC इन्हीं लाइनों की कुल गिनती होती है। यह मेट्रिक यह मापने का तरीका है कि कितने लाइन कोड लिखे गए हैं, जो प्रोजेक्ट के आकार का एक सामान्य माप देता है।
- LOC stands for “number of lines of code”. When a developer writes a program, he writes many lines, such as instructions, logic, functions, etc. LOC is the total count of these lines. This metric is a way to measure how many lines of code have been written, which gives a general measure of the size of the project.

C lang {
 |= Python {
 |= LOC
 |= }
 |= lang. dependent.
 }
 }

Program → 1000 lines
1- employee → 100 lines
1- employee → $100 \times 10 = 1000$
time → (10 days).



LOC के प्रकार (Types of LOC):

(1): भौतिक LOC (Physical LOC):

- इसमें कोड की हर लाइन गिनी जाती है, चाहे वह कोड हो, कमेंट्स (टिप्पणियाँ), या खाली लाइन्स। इसका फायदा यह होता है कि यह कोड की पूरी लंबाई दिखाता है।
- It counts every line of code, whether it is code, comments, or blank lines. The advantage of this is that it shows the full length of the code.

(2): तार्किक LOC (Logical LOC):

- इसमें केवल उन लाइनों को गिना जाता है जो वास्तव में कोड का हिस्सा होती हैं। खाली लाइनों और कमेंट्स को इसमें शामिल नहीं किया जाता। यह ज्यादा सटीक तरीके से कोड की असल जटिलता को मापने में मदद करता है।
- It only counts the lines that are actually part of the code. Blank lines and comments are not included. This helps in measuring the actual complexity of the code in a more accurate way.



→ LOC के फायदे (Advantages of LOC):

(1). सरलता (Simplicity):

- LOC को समझना और मापना बहुत आसान होता है। बस यह देखा जाता है कि कितनी लाइनें कोड में लिखी गई हैं।
- LOC is very easy to understand and measure. It simply looks at how many lines are written in the code.

(2). उत्पादकता का माप (Measurement of Productivity):

- यह माप सकता है कि एक डेवलपर कितनी तेजी से कोड लिख रहा है। अगर डेवलपर ने बहुत ज्यादा LOC लिखी है, तो इसका मतलब हो सकता है कि उसने ज्यादा काम किया है।
- It can measure how fast a developer is writing code. If a developer has written a lot of LOC, it may mean that he has done more work.



→ LOC के नुकसान (Disadvantages of LOC):

(1). गुणवत्ता का माप नहीं (Not a measure of quality):

- LOC केवल कोड की मात्रा को मापता है, लेकिन यह कोड की गुणवत्ता या उसकी प्रभावशीलता(effectiveness) नहीं बताता। छोटे, प्रभावी कोड को कम LOC में लिखा जा सकता है, जबकि बड़े और जटिल(complex) कोड ज्यादा LOC में हो सकते हैं।
- LOC only measures the amount of code, but does not indicate the quality or effectiveness of the code. Small, efficient code can be written in less LOC, while large and complex code can have more LOC.

(2). कमेंट्स और खाली लाइनों की गिनती(Counting comments and blank lines):

- Physical LOC कमेंट्स और खाली लाइनों को भी गिनता है, जो वास्तव में कोड का हिस्सा नहीं होते।
- Physical LOC also counts comments and blank lines, which are not actually part of the code.



Function Point Estimation:

- Function Point Analysis (FPA) एक मानकीकृत (standardized) विधि है, जिसे 1983 में Allan Albrecht(एलन अल्ब्रेक्ट) द्वारा विकसित किया गया था।
- Function Point Analysis (FPA) is a standardized method And its is develop by Allan Albrecht in 1983.
- Functional Point Analysis (FPA) एक सॉफ्टवेयर मापने की तकनीक है, जिसका उपयोग किसी सॉफ्टवेयर सिस्टम के आकार और जटिलता का आकलन करने के लिए किया जाता है। यह सॉफ्टवेयर की कार्यक्षमता (functionality) पर आधारित होता है
- Functional Point Analysis (FPA) is a software measurement technique used to assess the size and complexity of a software system based on its functionality.

$$FPE = UFP * VAF$$

\downarrow
 S value

\downarrow
14-factor



(1): Unadjusted Function Points (UFP):

- गणना करने के लिए, आपको सॉफ्टवेयर के हर प्रकार के काम (जैसे इनपुट, आउटपुट, डेटा फाइल्स, इंटरफेस, आदि) की संख्या को उसके वेट (weight) से गुणा करना होता है। फिर सभी कामों के अंक जोड़ दिए जाते हैं।
- Calculate UP by multiplying the number of each type of component by its corresponding weight and summing them up.

Unadjusted Function Points (UFP) Five Factor:

- (1.) External Inputs (EI).
- (2.) External Outputs (EO).
- (3.) External Inquiries (EQ).
- (4.) Internal Logical Files (ILF).
- (5.) External Interface Files (EIF).


$$UFP = \sum_{i=0}^{i=5} factor\ value \times weight$$



(1.) External Inputs (EI):

- यह वह डाटा या जानकारी है जो उपयोगकर्ता या किसी सिस्टम द्वारा सॉफ्टवेयर में डाली जाती है।
- These are the data or information that a user or another system enters into the software.

Example:

- पंजीकरण फॉर्म, उत्पाद खोज फॉर्म, कार्ट में जोड़े फॉर्म।
- registration form, product search form, add to cart form.

①

②

③

(2.) External Outputs (EO):

- यह वह जानकारी या डाटा है जो सॉफ्टवेयर से उपयोगकर्ता को बाहर भेजी जाती है।
- These are the results or information that the software sends out to the user.

Example

- Confirmation email, order summary page, invoice generation.

①

②

③

EO → ③



(3) External Inquiries (EQ):

- जब उपयोगकर्ता सॉफ्टवेयर से कुछ सवाल करता है और तुरंत जवाब चाहता है, जैसे डेटा खोजना या देखना।
- These are queries or requests from the user that require an immediate response, like searching or viewing data.

Example:-

- Product availability check, order status check. $\Rightarrow EQ \Rightarrow 2$

(4.) Internal Logical Files (ILF):

- यह वह फाइल्स हैं जो सॉफ्टवेयर के अंदर डाटा स्टोर करती हैं और सिस्टम द्वारा नियंत्रित होती हैं।
- These are files that store data within the software and are controlled by the system.

Example :-

- User data, product catalog, order data.

 $\Rightarrow ILF \Rightarrow 3$



(5) External Interface Files (EIF):

- यह ऐसी फाइल्स होती हैं जो दूसरे सिस्टम या सॉफ्टवेयर से जानकारी प्राप्त करती हैं, लेकिन सॉफ्टवेयर खुद उन्हें कंट्रोल नहीं करता।
- These are files that receive information from other systems or software, but the software itself does not control them.

Example :-

- payment gateway interface ,third-party shipping service interface.

①

②

EIF \Rightarrow ②



Function type	Multiplier		
	Simple	Average	Complex
External data input	2	3	4
External data output	3	4	5
Logical internal file	4	7	10
External interface file	5	5	7
External enquiry file	6	3	4

$$\text{UFI} = 2 \times 3 + 3 \times 4 + 4 \times 7 + 5 \times 5 + 6 \times 3$$



→ (2): Value Adjustment Factor (VAF):

- एक कारक(factor) जो सामान्य सिस्टम विशेषताओं (GSCs) के आधार पर UP को समायोजित(adjust) करता है, जिसमें प्रदर्शन(performance), प्रयोज्यता (usability) और जटिलता (complexity) जैसे विचार शामिल होते हैं।
- A factor that adjusts the UP based on general system characteristics (GSCs), which include considerations like performance, usability, and complexity.



$$\text{VAF} \Rightarrow 0.65 + 0.01 \sum_{i=0}^{i=14} (\underline{GSC})$$

$$G_{SC} = \sum_{i=0}^{i=14} \text{Value} \times \text{degree of influence}$$



→ (2): Value Adjustment Factor (VAF):

1. Data Communications
2. Distributed Data Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Rate
6. On-Line Data Entry
7. End-User Efficiency
8. On-Line Update
9. Complex Processing
10. Reusability
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Facilitate Change



degree of influence →

0 → Null
min → < 1 → Accidental
2 → moderate
3 → Average
4 → medium.
max → 5 → high



Question:

Consider a software project with the following information domain characteristic for the calculation of function point metric.

- Number of external inputs (I) = 10 \times 4
- Number of external output (O) = 20 \times 5
- Number of external inquiries (E) = 30 \times 4
- Number of files (F) = 10 \times 3
- Number of external interfaces (N) = 10 \times 2

It is given that the complexity weighting factors for I, O, E, F, and N are 4, 5, 4, 3, and 2, respectively. It is also given that, out of fourteen value adjustment factors that influence the development effort, four factors are not applicable, each of the other four factors has value 3, and each of the remaining factor has value 4.

The computed value of the function point metric.



COCOMO Model (Constructive Cost Model):

- COCOMO मॉडल (**Constructive Cost Model**) सॉफ्टवेयर विकास परियोजनाओं की लागत का अनुमान लगाने के लिए इस्तेमाल किया जाने वाला एक गणितीय(**mathematical**) मॉडल है।
- ~~इसे 1981 में बैरी बोहेम (**Barry Boehm**)~~ ने विकसित किया था।
- COCOMO का उद्देश्य सॉफ्टवेयर के विकास में लगने वाले समय, मेहनत (**effort**) और लागत को निर्धारित करना है।
- The COCOMO Model (**Constructive Cost Model**) is a mathematical model used to estimate the cost of software development projects.
- It was developed by Barry Boehm in 1981.
- The purpose of COCOMO is to estimate the time, effort, and cost required for software development.



Project Categories (Modes):

(a) ऑर्गेनिक (Organic):

- छोटी और आसान परियोजनाएँ, जिनमें कम जटिलता होती है।
- Small and simple projects with low complexity.

(b) सेमीडिटैच्ड (Semi-Detached):

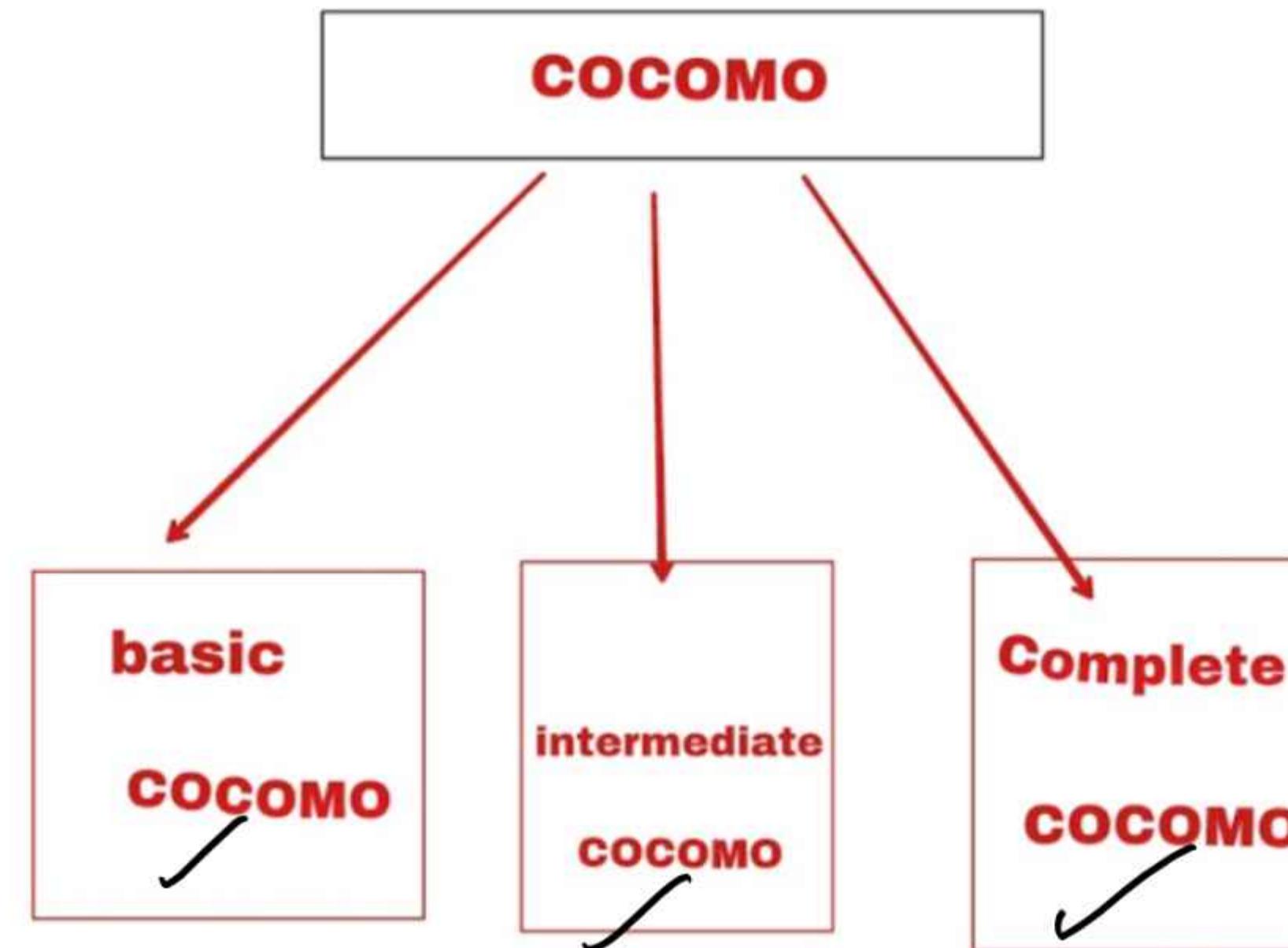
- मिडियम स्तर की परियोजनाएँ, जिनमें जटिलता और सरलता का संतुलन होता है।
- Medium-sized projects with a balance of complexity and simplicity.

(c) एंबेडेड (Embedded):

- बड़ी और जटिल परियोजनाएँ, जिनमें सख्त नियम होते हैं।
- Large and complex projects with strict requirements.



→ COCOMO Model (Constructive Cost Model):





(1): Basic COCOMO:

- यह सबसे सरल तरीका है, जिसमें केवल प्रोजेक्ट के आकार के आधार पर अनुमान लगाया जाता है। इसमें तीन प्रकार के प्रोजेक्ट होते हैं:
- This is the simplest model, where estimates are made based only on the size of the project. There are three types of projects:

→ time

→ cost

↳ faster

↳ cost ↓

↳ size (Based on size) cost



→ (2) इंटरमीडिएट COCOMO (Intermediate COCOMO):

- इसमें प्रोजेक्ट की लागत का अनुमान थोड़े विस्तार से किया जाता है। इसमें **15 कारक** (जैसे सॉफ्टवेयर की विश्वसनीयता, टीम का अनुभव) को ध्यान में रखा जाता है। इससे लागत का बेहतर अनुमान मिलता है।
- Here, project costs are estimated with more detail, considering 15 factors (such as software reliability and team experience). This gives a more accurate estimate of costs.

↳ Skill employe
↳ functionality.
↳ cost ↑



→ (3): एडवांस्ड COCOMO (Advanced COCOMO):

- यह सबसे विस्तृत तरीका है, जिसमें प्रोजेक्ट के अलग-अलग चरण (जैसे डिज़ाइन, कोडिंग, टेस्टिंग) के आधार पर अनुमान लगाया जाता है। यह सबसे सटीक अनुमान देता है क्योंकि हर छोटे हिस्से को ध्यान में रखा जाता है।
- This is the most detailed model, where estimates are based on different phases of the project (like design, coding, testing). It gives the most accurate estimate as it considers each part of the project



Advantage of COCOMO Model:

लागत, समय और मेहनत का सही अंदाज़ा

सॉफ्टवेयर के अलग-अलग प्रकार के प्रोजेक्ट्स को सपोर्ट करता है

Intermediate और Detailed मॉडल में cost drivers की मदद से और बेहतर अनुमान मिलता है
प्रोजेक्ट की योजना और संसाधनों (resources) को संभालने में मदद करता है

Detailed COCOMO में हर स्टेज के हिसाब से मेहनत का बंटवारा होता है

Accurate Estimation of Cost, Time & Effort

Support for Different Types of Software Projects

Better Accuracy with Cost Drivers in Intermediate/Detailed Models

Improves Project Planning & Resource Management

Phase-Wise Effort Distribution in Detailed COCOMO

Chapter No-3 (Software Planning)

Q 1: Define COCOMO Model. How to calculate Effort and Time in COCOMO Model?

Q 1: COCOMO मॉडल को परिभाषित करें। COCOMO मॉडल में प्रयास और समय की गणना कैसे करें?

Q 2: How function point metric analysis methodology is applied in estimation of software size? Explain.

Q 2: सॉफ्टवेयर आकार के आकलन में फ़ंक्शन पॉइंट मीट्रिक विश्लेषण पद्धति कैसे लागू की जाती है? समझाइए।

Q 3: What is LOC? Explain the process to calculate LOC?

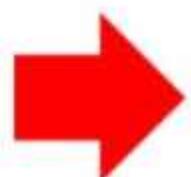
Q 3: LOC क्या है? LOC की गणना करने की प्रक्रिया बताएं?



Chapter-4 : Software Analysis Specification)

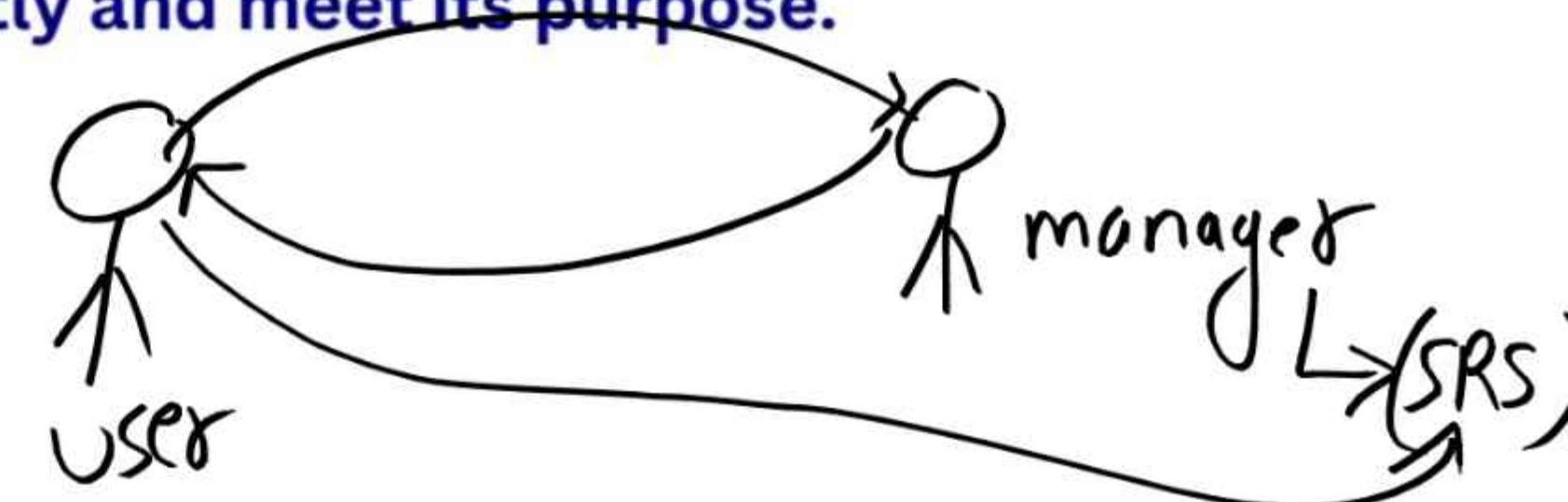
→ Syllabus :

Requirement gathering and Analysis, Software Requirement
Specifications (SRS), Characteristics of good SRS



Software Requirement:

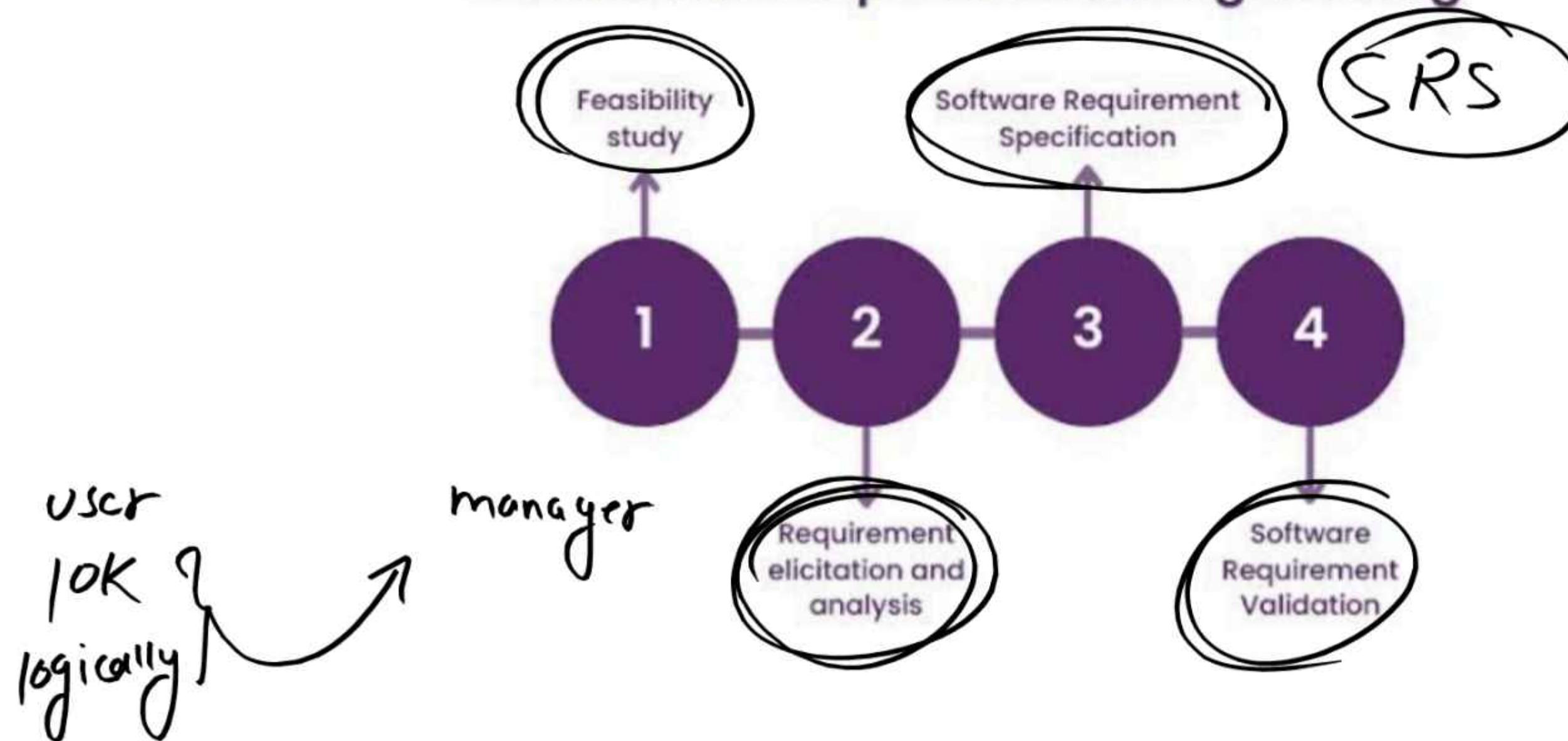
- सॉफ्टवेयर इंजीनियरिंग में, "रिक्वायरमेंट्स" (आवश्यकताएँ) का मतलब किसी सॉफ्टवेयर प्रोडक्ट के लिए पहले से तय की गई ज़रूरतों और अपेक्षाओं से होता है।
- In Software Engineering, requirements refer to the defined needs and expectations for a software product
- इन आवश्यकताओं को पहले से लिख लिया जाता है ताकि सॉफ्टवेयर को सही तरीके से विकसित (develop) किया जा सके और वह अपने उद्देश्य (purpose) को पूरा कर सके।
- These requirements are written down in advance so that the software can be developed correctly and meet its purpose.





Software Requirement:

Process of Requirements Engineering





Types of Requirements:

(1) User Requirement:

- "User Requirement वे कथन (statements) होते हैं जो यह बताते हैं कि अंतिम उपयोगकर्ता (end-user) सॉफ्टवेयर सिस्टम से क्या-क्या काम करने की उम्मीद करता है। ये उच्च-स्तरीय (high-level) आवश्यकताएँ होती हैं, जो सामान्य भाषा (natural language) में लिखी जाती हैं और उपयोगकर्ता की ज़रूरतें, उद्देश्यों और सॉफ्टवेयर से इंटरैक्शन पर केंद्रित होती हैं।"
- "User requirements are the statements that describe what the end-users expect the software system to do. These are high-level requirements written in natural language, focusing on the users' needs, goals, and interactions with the system."



(2) Functional Requirements:

- फंक्शनल रिक्वायरमेंट्स सॉफ्टवेयर के specific features and functions को परिभाषित करती हैं। ये बताती हैं कि सॉफ्टवेयर को क्या करना चाहिए
- **Functional requirements define the specific features and functions of the software, describing what the system should do.**

Example:

- For a banking software:
 - Allow users to log in and log out.
 - Enable users to check their balance.
 - Provide options to transfer funds.
 - Display transaction history.



(3) Non-Functional Requirements:

- नॉन-फंक्शनल रिक्वायरमेंट्स सॉफ्टवेयर की गुणवत्ता से जुड़ी विशेषताएँ (quality attributes) बताती हैं, जैसे - परफॉर्मेंस (गति), सुरक्षा (security), उपयोग में आसानी (usability), और विश्वसनीयता (reliability)।
- Non-functional requirements describe the quality attributes of the software, such as performance, security, usability, and reliability.

Example:

- For a banking software:
 - The response time should be under 2 seconds.
 - The software should support up to 1000 concurrent users.
 - Security must be maintained to protect against hacking.
 - The software should be available 99.9% of the time.

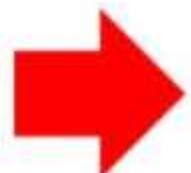


(4) System Requirements:

- सिस्टम रिक्वायरमेंट्स उन तकनीकी जानकारियों (technical specifications) को बताती हैं जो सॉफ्टवेयर को सही तरीके से चलाने के लिए ज़रूरी होती हैं। इनमें हार्डवेयर, सॉफ्टवेयर, और सॉफ्टवेयर के विकास (development), इंस्टॉलेशन, और मेंटेनेंस (maintenance) के लिए ज़रूरी पर्यावरणीय शर्तें (environmental conditions) शामिल होती हैं।
- **System requirements define the technical specifications needed for the software to run effectively, including hardware, software, and environmental conditions necessary for the system's development, installation, and maintenance.**

Example:

- For a mobile application:
 - The application should be compatible with Android 10 and iOS 13 or above.
 - The app requires a minimum of 4 GB RAM and 500 MB storage.



Requirement Gathering Techniques (आवश्यकता एकत्रीकरण तकनीकें):

- Requirements gathering एक प्रक्रिया है जिसमें किसी प्रोजेक्ट या सिस्टम के लिए आवश्यकताएं समझने और इकट्ठा करने का काम किया जाता है। यह सुनिश्चित करता है कि जो भी सिस्टम बनाया जा रहा है, वह यूजर की जरूरतों के हिसाब से सही तरीके से काम करे।
- Requirements gathering is a process of understanding and collecting the needs for a project or system to ensure it functions as expected for the users.
 - (i) Interview.
 - (ii) Surveys.
 - (iii) Brainstorming.
 - (iv) Prototyping.
 - (v) Observations.



(1): इंटरव्यू (Interviews):

- इसमें प्रोजेक्ट से जुड़े लोगों से सीधे बातचीत की जाती है। उनसे पूछा जाता है कि उन्हें सिस्टम से क्या चाहिए। यह बहुत ही सीधा तरीका है और इससे स्पष्ट जानकारी मिलती है।
- This involves directly talking to people involved in the project to ask them what they need from the system. It's a straightforward method and provides clear information.

(2) सर्वे और प्रश्नावली (Surveys and Questionnaires):

- इसमें एक फॉर्म या प्रश्नावली तैयार की जाती है, जिसमें यूजर्स से उनके विचार और आवश्यकताएं जानने के लिए सवाल पूछे जाते हैं। यह तब उपयोगी होता है जब कई लोगों से फीडबैक लेना हो।
- A form or set of questions is created to gather feedback from users about their needs and opinions. This is useful when collecting input from a large group of people.



(3): User के साथ बैठकें (Focus Groups):

- इसमें एक समूह को एक साथ बिठाया जाता है और उनसे उनकी आवश्यकताओं और विचारों पर चर्चा की जाती है। इससे एक ही समय में कई लोगों के विचार जानने में मदद मिलती है।
- A group of users is brought together to discuss their needs and ideas. It allows for understanding multiple perspectives at the same time.

(4): ऑब्जर्वेशन (Observation):

- इसमें यूजर्स को उनके काम करते समय ध्यान से देखा जाता है ताकि समझा जा सके कि वे किस तरह की समस्याओं का सामना करते हैं और उन्हें क्या चाहिए।
- Users are observed while they work to understand what challenges they face and what they require from the system.



(5): वर्कशॉप्स (Workshops):

- इसमें सभी संबंधित लोगों को एक साथ बिठाया जाता है ताकि वे खुलकर अपने विचार व्यक्त कर सकें और एक साथ आवश्यकताओं को परिभाषित किया जा सके।
- All relevant stakeholders come together in a workshop to express their thoughts and define requirements collaboratively.

(6): प्रोटोटाइपिंग (Prototyping):

- इसमें सिस्टम का एक छोटा मॉडल तैयार किया जाता है, ताकि यूजर्स को यह दिखाया जा सके कि अंतिम सिस्टम कैसा दिखेगा और काम करेगा। इससे यूजर्स अपनी आवश्यकताओं में बदलाव की सटीक जानकारी दे सकते हैं।
- A small model or mock-up of the system is created to show users what the final system might look like and how it would function. This helps users provide feedback and refine their needs accurately.



SRS (Software Requirements Specification):

- एक महत्वपूर्ण दस्तावेज़ होता है जो किसी सॉफ्टवेयर प्रोजेक्ट की सभी आवश्यकताओं को विस्तार से लिखता है।
- SRS (Software Requirements Specification) is an essential document that describes all the requirements for a software project in detail.
- SRS का उपयोग सॉफ्टवेयर डेवलपर्स, टेस्टर्स, प्रोजेक्ट मैनेजर्स और अन्य सभी स्टेकहोल्डर्स द्वारा किया जाता है
- The SRS document is used by software developers, testers, project managers, and other stakeholders to ensure that everyone understands the expectations for the software.





SRS (Software Requirements Specification):

1. Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions.
- 1.4 References
- 1.5 Overview

2. Overall Description

- 2.1 Product Perspective
- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 Constraints
- 2.5 Dependencies

3. Specific Requirements

- 3.1 Functional Requirements
- 3.2 Non-Functional Requirements
- 3.3 External Interface Requirements

DBms →

OA →



→ SRS की मुख्य बातें (Key Aspects of SRS):

1: उद्देश्य (Purpose):

- SRS का मुख्य उद्देश्य यह बताना होता है कि प्रोजेक्ट क्यों बनाया जा रहा है, इसके लक्ष्य क्या हैं, और इससे यूजर्स को क्या फायदा मिलेगा।
- The main purpose of an SRS is to explain why the project is being developed, its objectives, and how it will benefit users.

2: दायरा (Scope):

- इसमें प्रोजेक्ट के दायरे को समझाया जाता है, यानी यह बताया जाता है कि सॉफ्टवेयर क्या करेगा और क्या नहीं करेगा। यह सीमाएँ तय करने में मदद करता है।
- This defines the scope of the project, describing what the software will do and what it won't do. This helps in setting .



3: फंक्शनल आवश्यकताएँ (Functional Requirements):

- इसमें बताया जाता है कि सॉफ्टवेयर क्या-क्या काम करेगा, जैसे कि यूजर लॉगिन कर सकेगा, डेटा सेव कर ~~सकेगा~~, रिपोर्ट जनरेट कर सकेगा आदि।
- These detail what tasks the software will perform, such as allowing user login, saving data, generating reports, etc.

4: नॉन-फंक्शनल आवश्यकताएँ (Non-Functional Requirements):

- इसमें सॉफ्टवेयर की क्वालिटी, परफॉरमेंस, सुरक्षा, और यूजर एक्सपीरियंस जैसी चीज़ें शामिल होती हैं। जैसे, सॉफ्टवेयर को तेज़ी से काम करना चाहिए, या यह कितने समय तक उपलब्ध रहेगा।
- This covers aspects like quality, performance, security, and user experience. For example, the software should operate quickly and be highly available.



5: यूजर इंटरफेस आवश्यकताएँ (User Interface Requirements):

- इसमें सॉफ्टवेयर का लुक और फील, जैसे बटन, मेनू, और स्क्रीन की डिज़ाइन के बारे में जानकारी होती है, ताकि यूजर्स को आसानी हो।
- This describes the look and feel of the software, including elements like buttons, menus, and screen layouts to ensure user-friendliness.**

6: सिस्टम आवश्यकताएँ (System Requirements):

- इसमें बताया जाता है कि सॉफ्टवेयर किस प्लेटफॉर्म पर चलेगा, जैसे कि Windows, Linux, या मोबाइल। इसके साथ ही हार्डवेयर और नेटवर्क की भी आवश्यकताएँ बताई जाती हैं।
- This section specifies the platform on which the software will run, such as Windows, Linux, or mobile, and includes hardware and network requirements.**



7: प्रदर्शन आवश्यकताएँ (Performance Requirements):

- सॉफ्टवेयर की स्पीड, रेस्पॉन्स ट्रूम, और मैमोरी उपयोग जैसी चीजों को कवर किया जाता है ताकि यह सुनिश्चित हो सके कि सॉफ्टवेयर अच्छा प्रदर्शन करेगा।
- **Performance aspects like speed, response time, and memory usage are outlined to ensure that the software will perform well.**

8: सुरक्षा आवश्यकताएँ (Security Requirements):

- इसमें सॉफ्टवेयर की सुरक्षा, जैसे डेटा एन्क्रिप्शन, पासवर्ड प्रोटेक्शन, और यूजर ऑथेंटिकेशन जैसी बातें शामिल होती हैं।
- **This includes details on security measures like data encryption, password protection, and user authentication.**



Characteristics of Good SRS:

- SRS (Software Requirements Specification) की विशेषताएँ यह सुनिश्चित करती हैं कि सॉफ्टवेयर की आवश्यकताएँ सही तरीके से लिखी गई हैं और सभी स्टेकहोल्डर्स के लिए समझ में आने वाली हैं। एक अच्छी SRS की कुछ विशेषताएँ निम्नलिखित हैं:
- The characteristics of an SRS (Software Requirements Specification) ensure that software requirements are documented clearly and are understandable to all stakeholders. Here are the key characteristics of a good SRS:



Characteristics of Good SRS:





1: संपूर्णता (Completeness):

- एक अच्छी SRS में सभी आवश्यक जानकारी होनी चाहिए जो सॉफ्टवेयर को विकसित करने के लिए आवश्यक है। इसमें हर एक फीचर, कार्यक्षमता और सिस्टम के व्यवहार का विस्तार से उल्लेख होना चाहिए ताकि किसी भी तरह की जानकारी की कमी न रहे।
- A good SRS should contain all the necessary information needed to develop the software. It should include every feature, functionality, and behavior of the system in detail to ensure nothing is missing.

2: स्पष्टता (Clarity):

- SRS में भाषा clear और समझने में आसान होनी चाहिए। इसमें किसी भी तरह की अस्पष्टता नहीं होनी चाहिए ताकि सभी लोग इसे एक ही तरीके से समझें। उदाहरण के लिए, तकनीकी शब्दावली का उपयोग सीमित होना चाहिए, और सभी आवश्यकताओं को सरल भाषा में लिखा जाना चाहिए।
- The language in the SRS should be clear and easy to understand, with no ambiguity so that everyone interprets it the same way. For instance, technical jargon should be minimized, and all requirements should be written in simple language.



3: सुसंगतता (Consistency):

- SRS में सभी आवश्यकताएँ एक-दूसरे के साथ संगत होनी चाहिए। किसी भी तरह का विरोधाभास(contradictions) नहीं होना चाहिए, जैसे अगर एक स्थान पर कहा गया है कि यूजर को पासवर्ड 8 अक्षरों का डालना है, तो हर जगह यही नियम होना चाहिए।
- All requirements in the SRS should be consistent with each other. There should be no contradictions; for example, if one section states the password must be 8 characters long, this rule should be followed throughout the document.

4: संवेदनशीलता (Modifiability)

- SRS को इस प्रकार लिखा जाना चाहिए कि अगर भविष्य में किसी आवश्यकता में बदलाव की जरूरत हो तो इसे आसानी से संशोधित किया जा सके। इसमें किसी भी बदलाव के लिए जगह होनी चाहिए ताकि बिना अन्य हिस्सों को प्रभावित किए सुधार किया जा सके।
- The SRS should be written in a way that allows easy modification if any requirements change in the future. It should allow changes without impacting other sections unnecessarily.



5: सत्यापन योग्य (Verifiable):

- SRS की आवश्यकताएँ इस प्रकार लिखी जानी चाहिए कि इन्हें परखा या सत्यापित किया जा सके। प्रत्येक आवश्यकता को इस तरह से व्यक्त किया जाना चाहिए कि परीक्षण के दौरान यह सुनिश्चित किया जा सके कि सॉफ्टवेयर आवश्यकताओं को पूरा कर रहा है।
- The requirements in the SRS should be written so they can be verified or tested. Each requirement should be expressed in a way that allows it to be confirmed during testing to ensure the software meets the requirements.

6: सुरक्षितता (Traceability):

- SRS में प्रत्येक आवश्यकता का एक unique पहचानकर्ता होना चाहिए ताकि उसे आसानी से ट्रैक किया जा सके। यह सुविधा यह सुनिश्चित करती है कि किसी भी आवश्यकता को कहीं भी खोजा जा सके, और बाद में यह भी देखा जा सके कि कौन-सा कोड किस आवश्यकता के अनुसार लिखा गया है।
- Each requirement in the SRS should have a unique identifier, making it easy to trace and reference. This feature ensures that each requirement can be tracked, and later, it can be checked to see which code corresponds to which requirement.



7: उपयोगिता (Usability):

- एक अच्छी SRS में आवश्यकताएँ इस प्रकार लिखी जाती हैं कि सभी स्टेकहोल्डर्स, जैसे कि डेवलपर्स, टेस्टर्स, और मैनेजर्स, इसका सही उपयोग कर सकें। इसे पढ़ने और समझने में आसान होना चाहिए ताकि सभी लोग इसे अपनी जरूरत के अनुसार उपयोग कर सकें।
- A good SRS should be usable by all stakeholders, such as developers, testers, and managers. It should be easy to read and understand, enabling everyone to use it as needed.

8: प्रभावशीलता (Feasibility):

- SRS में केवल उन्हीं आवश्यकताओं को शामिल किया जाना चाहिए जिन्हें व्यवहार में लागू करना संभव हो। इसका अर्थ है कि आवश्यकताएँ व्यावहारिक और वास्तविक होनी चाहिए ताकि प्रोजेक्ट समय पर पूरा हो सके।
- The SRS should only include requirements that are practically achievable. Requirements should be realistic and implementable to ensure the project can be completed within the given time frame.

Chapter No-4 (Requirement Analysis and Implementation)

Q 1: Explain some basic components of SRS.

Q 1: SRS के कुछ बुनियादी घटकों की व्याख्या करें। ✓

Q 2: List the characteristics of good SRS document and their components.

Q 2: अच्छे एसआरएस दस्तावेज़ और उनके घटकों की विशेषताओं की सूची बनाएं। ✓

Q 3: Explain process of Requirement Gathering and Analysis.

Q 3: आवश्यकता एकत्रीकरण एवं विश्लेषण की प्रक्रिया समझाइए। ✓

Q 4: Discuss about various techniques of requirement gathering.

Q 4: आवश्यकता एकत्रीकरण की विभिन्न तकनीकों के बारे में चर्चा करें। ✓



Chapter No-5(Soft Design and implementation)

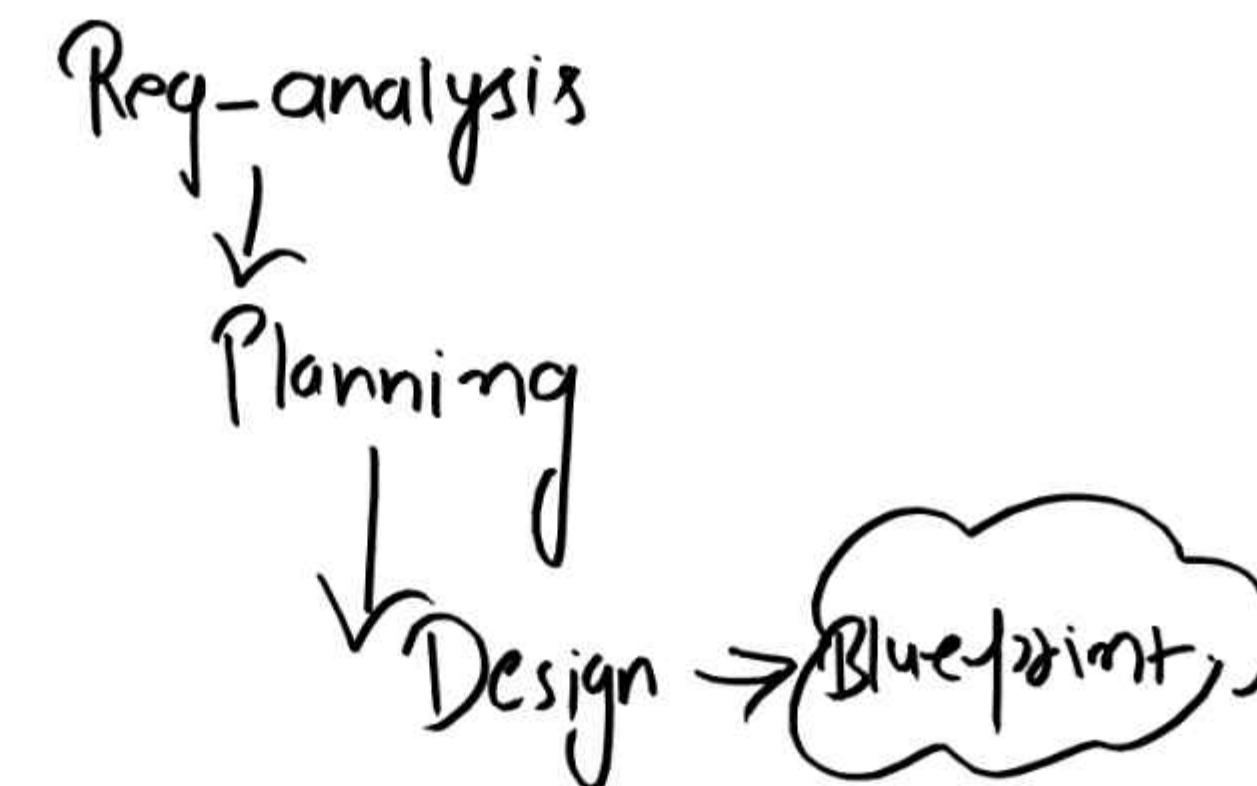
→ Syllabus :

- Characteristics and features of good Software Design.
- Cohesion and Coupling.
- Software design Approaches- Function Oriented Design (Data flow diagrams, Data dictionary, Decision Trees and tables), Object Oriented Design, Structured Coding Techniques, Coding Styles, and documentation.



Software Design:

- सॉफ्टवेयर डिज़ाइन वह प्रक्रिया है जिसमें सॉफ्टवेयर की आवश्यकताओं (requirements) को एक संरचना (structure) या योजना (blueprint) में बदला जाता है। इसमें सॉफ्टवेयर की आंतरिक संरचना, modules, interface, और data flow शामिल होता है।
- **Software Design is the process of transforming requirements into a blueprint for constructing the software, which includes architecture, components, interfaces, and data.**





Characteristics / Features of a Good Software Design

1. Correctness (सही होना):

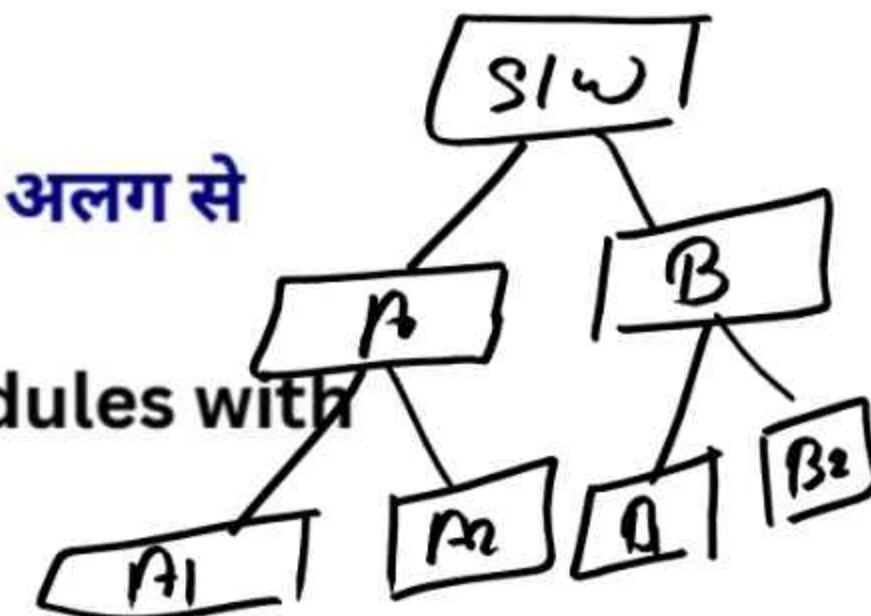
- सॉफ्टवेयर डिज़ाइन को सभी आवश्यक कार्यों को सही तरीके से पूरा करना चाहिए।
- The design should correctly implement all the required functionalities.

2. Understandability (समझने में आसान)

- डिज़ाइन इतना सरल और स्पष्ट होना चाहिए कि कोई भी डेवलपर उसे आसानी से समझ सके
- A good design should be easy to read and understand for developers.

3. Modularity (मॉड्यूल आधारित होना)

- सॉफ्टवेयर को छोटे-छोटे भागों (modules) में बाँटा जाना चाहिए जिससे उन्हें अलग से संभालना और समझना आसान हो।
- The system should be broken into smaller, manageable modules with clear functionality.





4. Reusability (पुनः उपयोग करने योग्य)

- डिज़ाइन के हिस्से दोबारा उपयोग में लिए जा सकें, यह एक अच्छा गुण है। ✓
- Components of the design should be reusable in other projects or modules.

5. Flexibility (लचीलापन) ✓

- सॉफ्टवेयर डिज़ाइन को आसानी से बदलने या सुधारने की क्षमता होनी चाहिए।
- Design should allow easy modifications and enhancements in future.

6. Maintainability (रखरखाव योग्य) ✓

- डिज़ाइन ऐसा होना चाहिए कि उसमें आसानी से सुधार किया जा सके या त्रुटियाँ ठीक की जा सकें।
- Design should support fixing bugs and making improvements easily. ✓

7. Efficiency (कुशलता) ✓

- सॉफ्टवेयर डिज़ाइन को संसाधनों (जैसे मेमोरी, प्रोसेसर) का बेहतर उपयोग करना चाहिए।
- The design should make good use of system resources like memory and CPU.



8. Scalability (विस्तारित करने योग्य)

- डिज़ाइन को भविष्य में उपयोगकर्ताओं या डाटा बढ़ने पर भी बिना बदलाव के काम करना चाहिए।
- It should support growth in data or users without major redesign.

9. Security (सुरक्षा)

- डिज़ाइन ऐसा होना चाहिए जो डेटा को गलत उपयोग या हैकिंग से बचाए।
- A good design protects the system from unauthorized access or misuse.

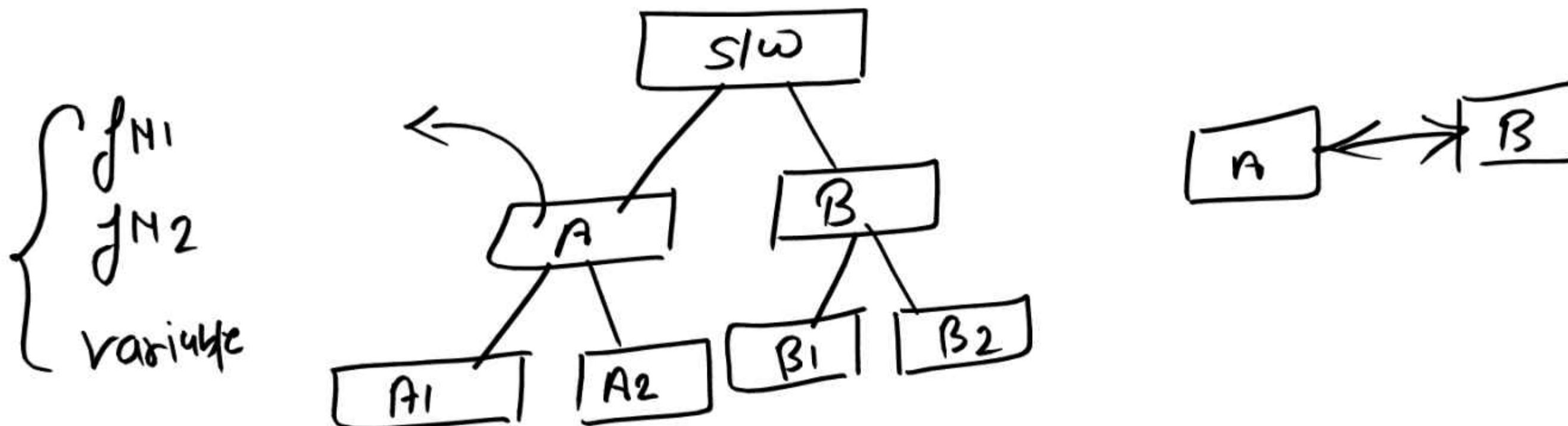
10. Testability (जाँचने योग्य)

- डिज़ाइन ऐसा होना चाहिए कि उसके प्रत्येक भाग को अलग या मिलाकर आसानी से टेस्ट किया जा सके।
- It should be easy to test the components individually or in combination.



Cohesion and Coupling:

- जब एक सॉफ्टवेयर प्रोग्राम को कई मॉड्यूल्स में विभाजित किया जाता है, तो इन मॉड्यूल्स के बीच परस्पर क्रिया (इंटरएक्शन) और उनके डिज़ाइन की गुणवत्ता को मापा जा सकता है। इन मापदंडों(measures) को कोहेशन और कपलिंग कहा जाता है।
- When a software program is divided into several modules, interaction between these modules and their design quality can be measured. These measures are called **Cohesion and Coupling**.



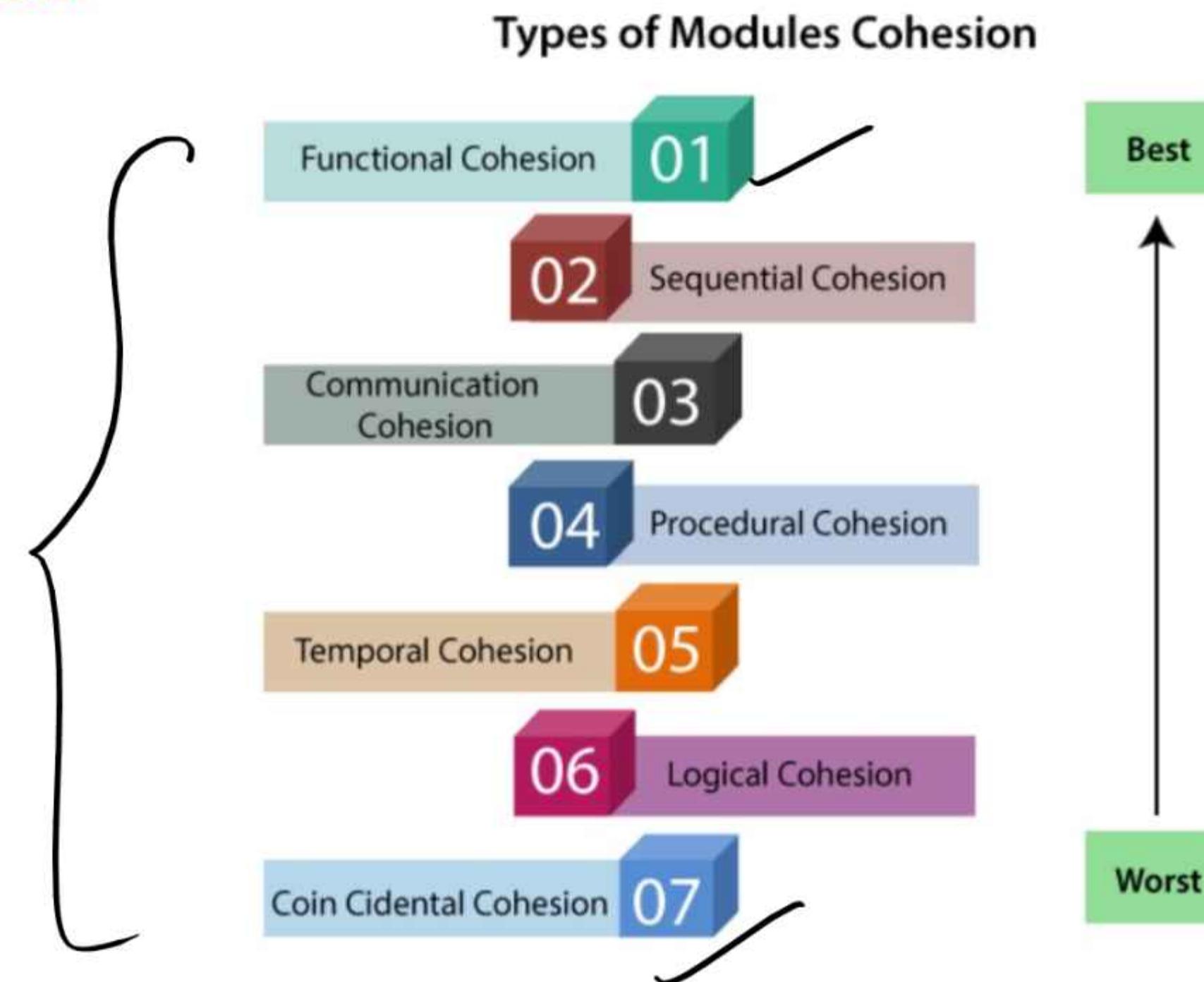


Cohesion:

- Cohesion (कोहेशन) एक माप है जो यह दर्शाता है कि किसी मॉड्यूल (module) के अंदर मौजूद कार्य या एलिमेंट्स आपस में कितनी अच्छी तरह से जुड़े हुए हैं।
- जब किसी मॉड्यूल के सभी कार्य एक ही उद्देश्य को पूरा करते हैं और आपस में गहराई से जुड़े होते हैं, तो उसे High Cohesion कहा जाता है।
- Cohesion is the degree to which the elements inside a module belong together.
- A module is said to have high cohesion if its responsibilities are strongly related and focused on a single task. High cohesion makes the module more understandable, maintainable, and reusable.



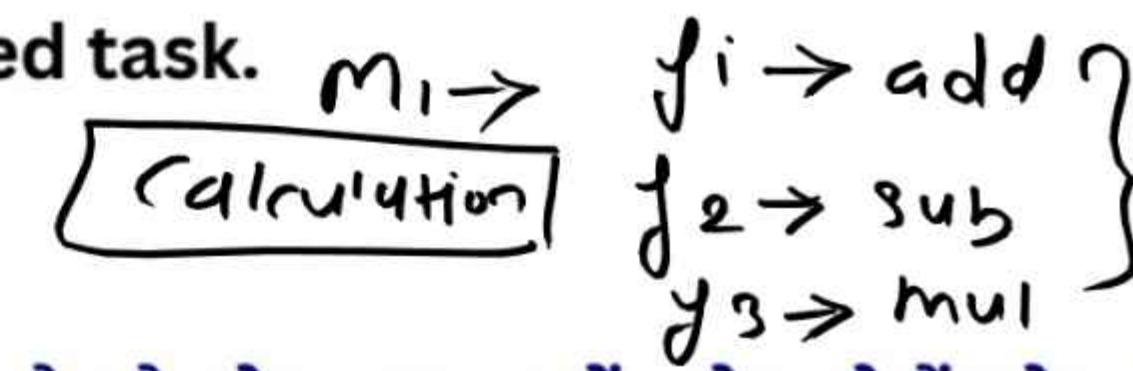
Type of Cohesion:





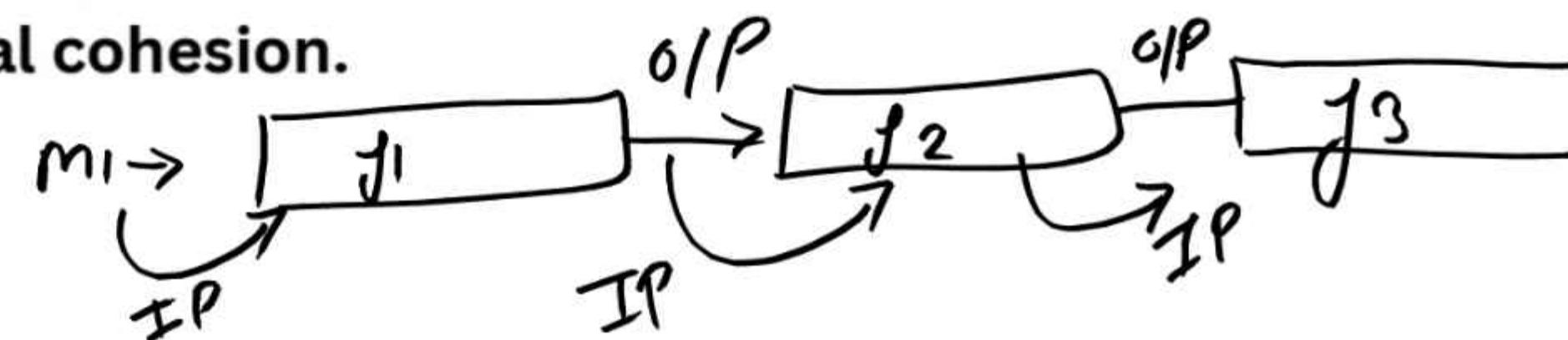
1. Functional Cohesion (फंक्शनल कोहेशन):

- जब एक module का हर भाग एक ही काम को पूरा करने के लिए आपस में जुड़ा होता है, तो उसे functional cohesion कहते हैं। Best cohesion माना जाता है
- A module is said to have functional cohesion when all parts of the module work together to achieve a single, well-defined task.



2. Sequential Cohesion (सीक्वेंशियल कोहेशन)

- जब एक task का output, दूसरे task का input बनता है और ये काम क्रम में जुड़े रहते हैं, तो उसे sequential cohesion कहते हैं।
- When the output from one part of the module becomes the input for the next part, it's called sequential cohesion.



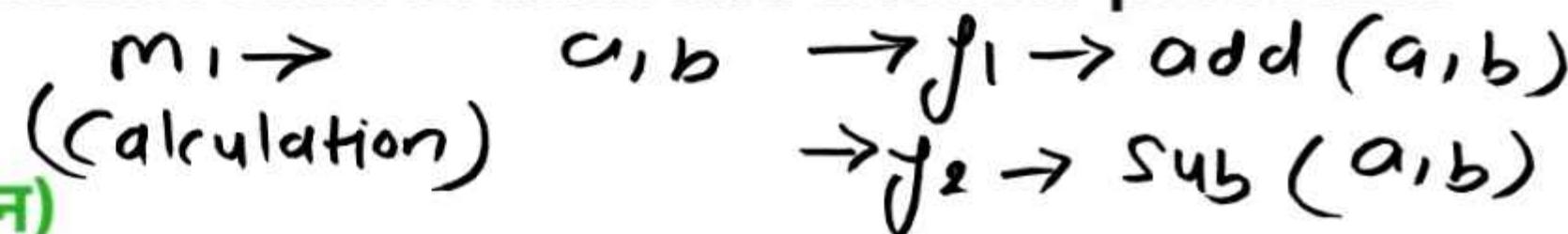


3. Communicational Cohesion (कम्युनिकेशनल कोहेशन)

- जब module के सारे काम एक ही डेटा पर आधारित होते हैं या एक ही इनपुट/आउटपुट को यूज़ करते हैं।
उदाहरण के लिए, अगर एक फ़ंक्शन डेटा को प्राप्त करता है और दूसरा उसे संसाधित करता है।
- A module has **communicational cohesion** when its tasks use the same input or output data. For example, if one function receives data and another processes it.

4. Procedural Cohesion (प्रोसीजरल कोहेशन)

- जब मॉड्यूल के हिस्से एक निश्चित अनुक्रम में काम करते हैं, लेकिन उनका उद्देश्य अलग-अलग हो सकता है। जैसे, एक प्रक्रिया जो कई प्रकार के कार्य करती है, लेकिन उनका उद्देश्य एक जैसा नहीं होता।
- When parts of a module work in a certain sequence but their purpose may be different. For example, a procedure that performs several tasks but does not have the same purpose.





5. Temporal Cohesion (टेम्पोरल कोहेशन)

- जब मॉड्यूल के हिस्से एक साथ एक निश्चित समय पर कार्य करते हैं, जैसे सिस्टम की शुरुआत में कुछ काम करना।
- When parts of a module work together at a certain time, such as doing some work at the start of the system.

6. लॉजिकल संगति (Logical Cohesion):

- जब एक मॉड्यूल के सभी हिस्से एक ही प्रकार के काम करते हैं, लेकिन उनमें से कोई भी हिस्सा अलग-अलग काम कर सकता है। उदाहरण के लिए, एक फ़ंक्शन जो विभिन्न गणनाओं को कर सकता है, लेकिन इन गणनाओं का उद्देश्य अलग-अलग हो सकते हैं।
- When all parts of a module perform the same type of work, but any of those parts can perform different tasks. For example, a function that can perform different calculations, but the purpose of these calculations can be different.



7. संयोगात्मक संगति (Coincidental Cohesion):

- यह सबसे खराब प्रकार है, जब मॉड्यूल के हिस्से किसी विशेष कारण से एक साथ रखे जाते हैं, और उनका आपस में कोई संबंध नहीं होता। जैसे, एक फ़ंक्शन जिसमें बहुत सारे अलग-अलग कार्य एक साथ होते हैं, जिनका आपस में कोई तालमेल नहीं होता।
- This is the worst kind, when parts of a module are put together for a specific reason, and they have no relationship with each other. For example, a function that puts together a lot of different tasks that have no coordination with each other.



Coupling:

- कपलिंग विभिन्न मॉड्यूल्स के बीच परस्पर संपर्क को मापता है। यह दर्शाता है कि दो मॉड्यूल्स एक-दूसरे के साथ कितनी मजबूती से जुड़े हए हैं। जब दो मॉड्यूल्स में उच्च कपलिंग होती है, तो इसका मतलब है कि वे एक-दूसरे पर बहुत अधिक निर्भर होते हैं।
- Coupling measures the interaction between different modules. It shows how strongly two modules are interconnected with each other. When two modules are in high coupling it means they are highly dependent on each other.**



Types of Coupling:

- content coupling. (worst)
 - common coupling.
 - external coupling.
 - control coupling.
 - stamp coupling
 - data coupling. Best
- Out of these Data Coupling is the best form of coupling and Content Coupling is worst form of coupling.



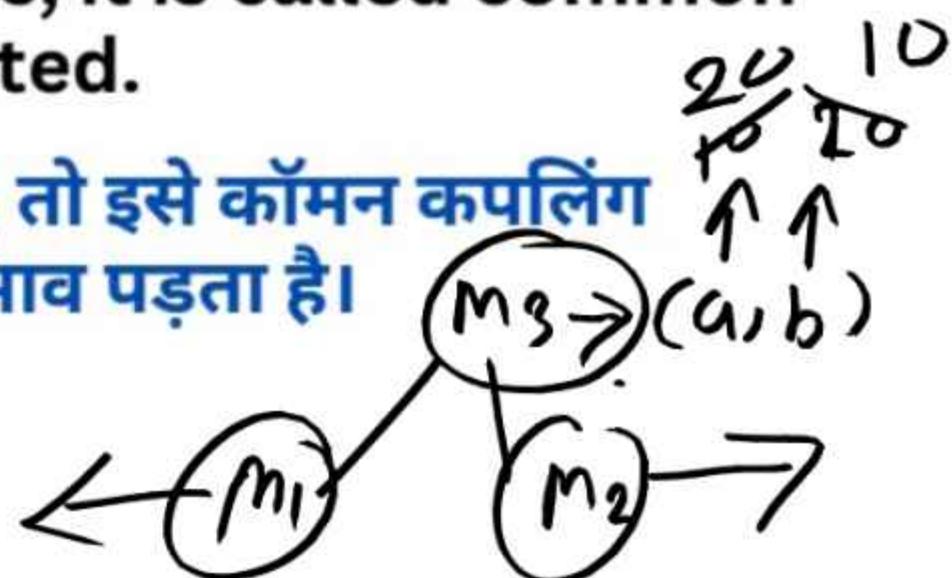
1. कंटेंट कपलिंग (Content Coupling):

- When one module directly accesses or changes the code of another module, it is called content coupling. This is the worst type of coupling because it leads to excessive dependencies between modules.
- जब एक मॉड्यूल सीधे दूसरे मॉड्यूल के कोड को एक्सेस करता है या उसे बदलता है, तो इसे कंटेंट कपलिंग कहते हैं। यह सबसे खराब प्रकार का कपलिंग है क्योंकि इससे मॉड्यूल्स के बीच अत्यधिक निर्भरता होती है।



2. कॉमन कपलिंग (Common Coupling):

- When two or more modules use the same global variable, it is called common coupling. If that variable changes, all modules are affected.
- जब दो या अधिक मॉड्यूल्स एक ही ग्लोबल वेरिएबल का उपयोग करते हैं, तो इसे कॉमन कपलिंग कहते हैं। अगर उस वेरिएबल में बदलाव होता है, तो सभी मॉड्यूल्स पर प्रभाव पड़ता है।





3. एक्स्टर्नल कपलिंग (External Coupling):

- When two modules depend on external systems or devices (e.g. files, hardware), it is called external coupling.
- जब दो मॉड्यूल्स बाहरी सिस्टम या डिवाइस (जैसे फ़ाइल, हार्डवेयर) पर निर्भर होते हैं, तो इसे एक्स्टर्नल कपलिंग कहते हैं।



4. कंट्रोल कपलिंग (Control Coupling):

- When one module sends some control data (such as flags or switches) to another module to make it work, it is called control coupling.
- जब एक मॉड्यूल दूसरे मॉड्यूल को काम करने के लिए कुछ नियंत्रण डेटा (जैसे फ्लैग या स्विच) भेजता है, तो इसे कंट्रोल कपलिंग कहते हैं।





5. स्टैम्प कपलिंग (Stamp Coupling):

- जब दो मॉड्यूल्स बड़े डेटा स्ट्रक्चर (जैसे रिकॉर्ड या ऑब्जेक्ट) का इस्तेमाल करते हैं, लेकिन पूरे डेटा की बजाय केवल कुछ हिस्से की जरूरत होती है, तो इसे स्टैम्प कपलिंग कहते हैं।
- When two modules use a large data structure (such as records or objects) but need only portions of the data rather than the entire data, this is called stamp coupling.

6. डेटा कपलिंग (Data Coupling):

- जब मॉड्यूल्स केवल आवश्यक डेटा (प्रॉपर इनपुट और आउटपुट) का आदान-प्रदान करते हैं, तो इसे डेटा कपलिंग कहते हैं। यह सबसे अच्छा प्रकार का कपलिंग है क्योंकि इसमें मॉड्यूल्स स्वतंत्र रहते हैं।
- When modules exchange only the required data (proper inputs and outputs), it is called data coupling. This is the best type of coupling because the modules remain independent.





Software Design Approach:

Function Oriented Design:

- Function-Oriented Design) सॉफ्टवेयर डिज़ाइन का एक तरीका है, जिसमें हम सॉफ्टवेयर को उसके कार्यों (functions) के आधार पर डिज़ाइन करते हैं, न कि उसके डेटा के आधार पर। इसमें हम यह देखते हैं कि सॉफ्टवेयर के अलग-अलग कार्य कैसे डेटा को प्रोसेस करते हैं और एक दूसरे के साथ कैसे इंटरएक्ट करते हैं। इस डिज़ाइन में चार मुख्य तकनीकों का उपयोग किया जाता है:
- Function-Oriented Design is an approach to software design in which we design software based on its functions, not its data. In this, we look at how different functions of the software process data and interact with each other. Four main techniques are used in this design.

- Data Flow Diagram - DFD
- Data Dictionary
- Decision Tree
- Decision Table



1: डेटा फ्लो डायग्राम (Data Flow Diagram - DFD):

- DFD एक चित्र होता है, जो दिखाता है कि डेटा सिस्टम के अंदर कैसे यात्रा(travel) करता है। यह यह भी बताता है कि डेटा कहां से आता है, कहां जाता है और किस तरह से प्रोसेस होता है।
- A DFD is a diagram that shows how data travels within a system. It also shows where the data comes from, where it goes, and how it is processed.
- प्रत्येक circle या bubble एक प्रोसेस को दर्शाता है, और तीर (arrows) यह दिखाते हैं कि इन step के बीच डेटा कैसे चलता है।
- Each circle or bubble shows a process, and arrows show the movement of data between these steps.



Component of DFD:

(1): Entity:

- एंटिटी information या डेटा के source और destination होते हैं, या हम कह सकते हैं डेटा के इनपुट या आउटपुट होते हैं। इन्हें आयत (rectangles) द्वारा दर्शाया जाता है।
- Entities are source and destination of information or data, or we can say input or output of data. They are represented by rectangles.

(2) Process:

- process इनपुट को आउटपुट में transform करती है। यह डेटा पर की गई operation होती है। इसे circle या गोल कोनों वाले आयतों द्वारा दर्शाया जाता है।
- Process transform input into output. It is operation done on data. It is shown by circle or round cornered rectangles.



(3). Data flow:

- यह system के एक part/step से दूसरे part/step तक information /डेटा/सामग्री का transfer को show करता है। इसे तीर(arrow) द्वारा represent किया जाता है। तीर flow की दिशा को दिखाता है।
- It shows transfer of information/data/material from one part/stage of system to another part/stage. It is represented by arrow. The arrow show the direction of flow

(4). Warehouse/database:

- process डेटाबेस का उपयोग डेटा को फ़ाइलों या अन्य रूपों में future में उपयोग के लिए store करने के लिए किया जाता है। इसे दो समांतर रेखाओं द्वारा दर्शाया जाता है, और इन रेखाओं के बीच में स्टोरेज डिवाइस का नाम लिखा जा सकता है।
- process Database is used to store data in files or other forms for future use. It is represented by two parallel lines, and the name of the storage device can be written between these lines.



(5).Terminator:

- यह एक बाहरी एंटिटी होती है जो सिस्टम के साथ **communicates** रती है, लेकिन सिस्टम का हिस्सा नहीं होती। यह **system** का बाहरी **element** होता है।
- It is an external entity which communicate with system but not a part of system. It is outside element of a system

Level in a Data Flow Diagram (DFD):

- (i) Level 0 DFD.
- (ii) Level 1 DFD.
- (iii) Level 2 DFD.



Rules for DFD:

- प्रत्येक process में कम से कम एक इनपुट और एक आउटपुट होना चाहिए।
- प्रत्येक डेटा स्टोर में कम से कम एक डेटा फ्लो अंदर और एक डेटा फ्लो बाहर होना चाहिए।
- किसी भी सिस्टम में store डेटा को एक process से होकर गुजरना चाहिए।
- DFD (डेटा फ्लो डायग्राम) में सभी processes या तो दूसरी process या डेटा store की ओर जाती हैं।
- Each process should have at least one input and an output.
- Each data store should have at least one data flow in and one data flow out.
- Data stored in a system must go through a process.
- All processes in a DFD go to another process or a data store.



(i) Level 0 DFD:

- इसे संदर्भ आरेख (Context Diagram) भी कहा जाता है। इसमें एक ही process होती है और इनपुट और आउटपुट केवल इसी एकल process से जुड़े होते हैं।
- It is also called context diagram. Here there is a single process and input and output are linked with this single process only.



(ii) Level 1 DFD:

- इसमें multiple processes होती हैं। पूरी process को कई sub-process में विभाजित किया जाता है।
- It has multiple processes. The whole process to divided into many sub-process.



(iii) Level 2 DFD:

- यह process की अधिक जानकारी प्रदान करता है। यह process के अधिक detail को represents करता है। इसका उपयोग system की specific structure को रिकॉर्ड करने या plan बनाने के लिए किया जा सकता है।
- It goes into more details of process. It presents more detail of the process. It can be used to record or plan specific makeup of a system



Data Dictionary:

- Data Dictionary एक centralized repository होती है, जिसमें सिस्टम में उपयोग किए गए सभी data elements की definitions और descriptions Store होती हैं। इसमें प्रत्येक data item का name, type, length, valid values, default value, और उसका usage context शामिल होता है।
- A Data Dictionary is a centralized repository of metadata that provides definitions and descriptions for all data elements used in a system. It includes the name, type, length, valid values, default value, and usage context of each data item.

int a → 2 byte



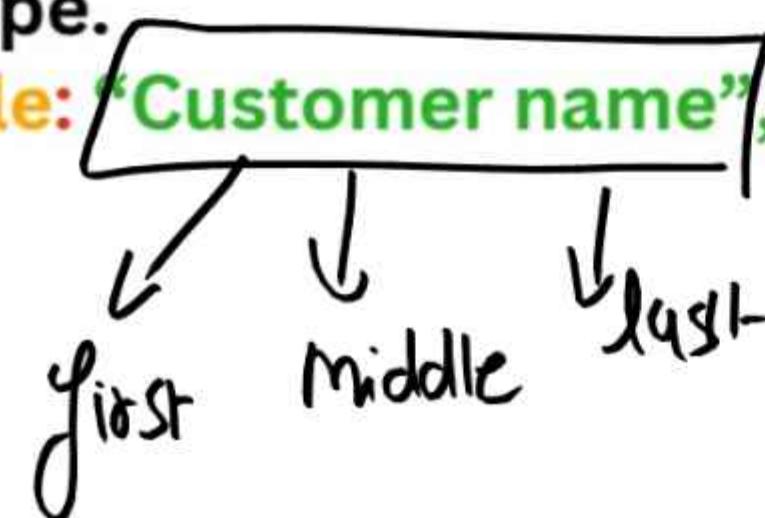
डेटा डिक्शनरी के मुख्य घटक (Components of Data Dictionary):

1. Data Entity:

- यह एक डेटा तत्व या वस्तु को दर्शाता है, जिसका उपयोग सॉफ्टवेयर सिस्टम में किया जाता है।
- It represents a data element or object that is used in a software system
- Example: “customer”, “order”, “product” etc.

2. डेटा एलिमेंट (Data Element):

- यह एक छोटा data item होता है जिसे एक data entity के रूप में store किया जाता है। यह किसी विशेष data type को represent करता है।
- It is a small data item that is stored as a data entity. It represents a particular data type.
- Example: “Customer name”, “Order date”, “Product price” etc.





3. डेटा टाइप (Data Type):

- यह डेटा के प्रकार को दर्शाता है, जैसे कि संख्यात्मक (numeric), वर्णमाला (alphanumeric), तार (string) आदि।
- It indicates the type of data, such as numeric, alphanumeric, string etc.
- Example: “customer name” – Type: string, “order number” – Type: integer.

4. डेटा आकार (Data Size):

- डेटा के आकार का मतलब उस डेटा तत्व को करने के लिए आवश्यक मेमोरी स्थान से है।
- Data size means the memory space required to store that data element.
- Example: “Customer Name” size can be 50 characters.

5. डेटा सीमा (Data Range):

- यह डेटा के मानों की सीमा को निर्दिष्ट करता है, यानी उस डेटा का न्यूनतम और अधिकतम मान क्या हो सकता है।
- It specifies the range of values of the data, i.e. what is the minimum and maximum value that data can have.



6. डेटा का विवरण (Description of Data):

- यह डेटा के बारे में एक संक्षिप्त विवरण देता है कि इसका उपयोग कैसे किया जाएगा और यह सिस्टम में किस उद्देश्य से मौजूद है। ✓
- It gives a brief description about the data, how it will be used and for what purpose it is present in the system.
- Example: “Customer Name” – This is used to identify the customer.

7. डेटा का स्रोत (Source of Data):

- यह बताता है कि डेटा कहाँ से प्राप्त किया जाएगा या यह कहाँ से आ रहा है।
- It tells where the data will be retrieved from or where it is coming from



Advantages of Data Dictionary:

1: Centralized Metadata Storage:

- सभी data elements की जानकारी एक ही जगह store रहती है, जिससे सिस्टम को समझना आसान होता है।
- Information about all data elements is stored in one place, which makes the system easier to understand.

2: Improves Consistency:

- एक ही data item के लिए एक जैसा definition और format सुनिश्चित करता है, जिससे data redundancy और गलतियों से बचा जा सकता है।
- Ensures consistent definition and format for the same data item, thereby avoiding data redundancy and errors.



3. Enhances Communication:

- डेवलपर्स, डिजाइनर्स और यूजर्स के बीच common understanding बनती है क्योंकि सभी को एक जैसा reference मिलता है।
- A common understanding is created between developers, designers and users as everyone gets the same reference.

4. Facilitates Maintenance:

- सिस्टम के किसी हिस्से को बदलना या अपडेट करना आसान हो जाता है, क्योंकि सभी data definitions एक जगह मिलती हैं।
- It becomes easier to change or update any part of the system, as all data definitions are available in one place.

5. Supports Database Design:

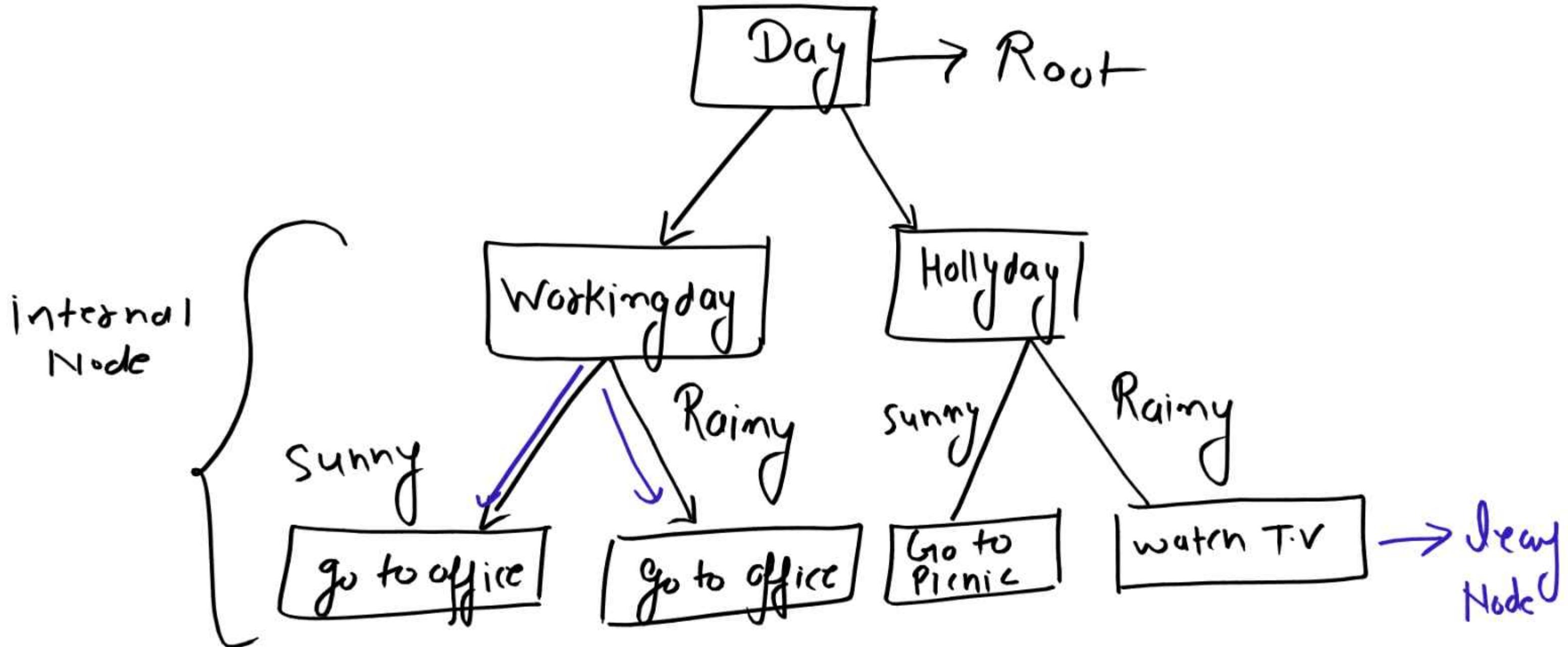
- यह database schema design और normalization में गाइड करता है।
- It guides in database schema design and normalization.



Decision Tree:

- यह एक tree structure (वृक्ष संरचना) वाला diagram होता है जो किसी समस्या को हल करने के लिए decisions (निर्णयों) और उनके possible outcomes (संभावित परिणामों) को दर्शाता है।
- It is a diagram with a tree structure that shows the decisions and their possible outcomes to solve a problem.

Element	Description
Root Nodes	यह शुरुआती निर्णय (initial decision) को दर्शाता है।
Internal Nodes	ये nodes decisions को दर्शाते हैं जो आगे और विकल्पों की ओर ले जाते हैं।
Branches Nodes	प्रत्येक branch एक विकल्प (option) या निर्णय का परिणाम (result) दिखाता है।
Leaf Nodes	यह अंतिम output (result) को दर्शाता है।





Advantage of Decision Tree:

- इसका स्ट्रक्चर सरल और विज़ुअल होता है, जिसे कोई भी आसानी से समझ सकता है।
- जटिल समस्याओं को छोटे-छोटे हिस्सों में बांटकर निर्णय लेना आसान बनाता है।
- यह डेटा को व्यवस्थित तरीके से प्रस्तुत करता है और संभावित परिणाम दिखाता है।
- इसे समझने और इस्तेमाल करने के लिए ज्यादा तकनीकी जानकारी की जरूरत नहीं होती।
- विकल्प और उनके परिणाम स्पष्ट रूप से दिखाने से निर्णय जल्दी लिए जा सकते हैं।
- Its structure is simple and visual, which anyone can easily understand.
- It makes decision making easier by breaking down complex problems into smaller parts.
- It presents data in an organized manner and shows possible outcomes.
- It does not require much technical knowledge to understand and use it.
- Decisions can be made quickly by clearly showing options and their outcomes.



Decision Table:

- Decision Table एक ऐसा टूल है जिसका उपयोग निर्णय लेने के लिए नियमों और उनके परिणामों को तालिका (table) या मैट्रिक्स के रूप में दिखाने के लिए किया जाता है। यह एक विज़ुअल तरीका है, जो बताता है कि किसी विशेष स्थिति (condition) के आधार पर कौन-सी क्रिया (action) करनी है।
- Decision Table is a tool used to show decision making rules and their results in the form of a table or matrix. It is a visual way to tell which action to take based on a particular condition.

**Example:**

- जब हम किसी वेबसाइट या पोर्टल में लॉगिन करते हैं, तो बैकग्राउंड में यह जांच की जाती है कि उपयोगकर्ता नाम (username) और पासवर्ड सही हैं या नहीं। यदि दोनों का संयोजन सही होता है, तो लॉगिन की अनुमति दी जाती है।
- When we login into any website or any portal at background it checks whether user name or password is correct and allow to login if combination of both is correct.**

Condition	Case 1	Case 2	Case 3	Case 4
1. User Name 2. Password	(✓) (✗)	(✗) (✓)	(✗) (✗)	(✓) (✓)
Action	fail	fail	fail	Pass · login



Advantage of Decision table:

- यह जटिल नियमों और शर्तों को आसान और व्यवस्थित तरीके से दिखाता है।
- इसे किसी भी क्षेत्र में इस्तेमाल किया जा सकता है, जैसे बैंकिंग, ई-कॉमर्स, और आईटी सिस्टम।
- विभिन्न शर्तों और उनके प्रभाव को आसानी से तुलना करने में मदद करता है।
- प्रोग्रामिंग और लॉजिक सिस्टम को डिजाइन करने में Decision Table बहुत काम आता है।
- सभी स्थितियों को कवर करने से गलतियों के चांस कम हो जाते हैं।
- It shows complex rules and conditions in an easy and organized way.
- It can be used in any field, such as banking, e-commerce, and IT systems.
- It helps in comparing different conditions and their effects easily.
- Decision Table is very useful in designing programming and logic systems.
- Covering all the conditions reduces the chances of mistakes.



Object Oriented Design (OOD):

- Object Oriented Design (OOD) एक ऐसी डिज़ाइन तकनीक है जिसमें किसी सॉफ्टवेयर सिस्टम को आपस में इंटरैक्ट करने वाले objects के रूप में मॉडल किया जाता है। प्रत्येक object एक class का instance होता है, जो अपने भीतर data (attributes) और उन पर काम करने वाले functions या methods को समाहित करता है।
- Object Oriented Design (OOD) is a design methodology in software engineering that models a system as a collection of interacting objects, where each object is an instance of a class, encapsulating both data and the operations (methods) that manipulate the data.



Main Concepts (मुख्य अवधारणाएं):

Class	Blueprint of an object. (Object की संरचना तय करता है)
Object	Instance of a class. (Class से बना एक वास्तविक इकाई)
Encapsulation	Wrapping of data and functions into one unit (Class). (डेटा और फंक्शन्स को एक साथ बाँधना)
Abstraction	Hiding internal details and showing only essential features. (जरूरी चीज़ें दिखाना, बाकी छुपाना)
Inheritance	Reusing properties and behavior of one class in another. (एक class से दूसरी class में गुण लेना)
Polymorphism	One function behaving differently based on context. (एक ही नाम से अलग-अलग काम)



Advantages of Object Oriented Design (OOD):

Code Reusability (कोड का पुनः उपयोग):

- Inheritance के माध्यम से एक बार लिखे गए code को बार-बार उपयोग किया जा सकता है।
- Through inheritance, code written once can be used again and again.

Easy Maintenance & Updates (आसान रख-रखाव और अपडेट):

- Modular structure होने से software को maintain करना और update करना आसान होता है।
- Having a modular structure makes it easy to maintain and update the software.

Better Scalability (बेहतर विस्तार योग्यता):

- OOD बड़े systems को आसानी से expand या grow करने में सहायक होता है।
- OOD helps to expand or grow large systems easily.

Natural Mapping of Real-World Problems (वास्तविक समस्याओं का स्वाभाविक अनुकरण):

- OOD में real-world entities को object के रूप में directly represent किया जाता है।
- In OOD, real-world entities are directly represented as objects.



Structured Coding Techniques:

- Structured coding techniques वे programming practices हैं जो clean, readable और maintainable code लिखने पर ध्यान देती हैं। इसमें structured programming constructs जैसे sequences, conditionals (if-else) और loops (for, while) का उपयोग किया जाता है, और unstructured control flow (जैसे goto statement) से बचा जाता है।
- Structured coding techniques are programming practices that focus on writing clean, readable, and maintainable code using structured programming constructs such as sequences, conditionals (if-else), and loops (for, while), while avoiding unstructured control flow.

```
{ int a,b;  
    a=10;  
    b=20;  
    c=a+b;
```



Techniques:

1. Sequence Structure (क्रमिक संरचना):

- Sequence का अर्थ है कि सभी स्टेटमेंट्स को उसी क्रम में एक-एक करके चलाया जाए, जैसे वे कोड में लिखे गए हैं। इसमें कोई decision-making या loop नहीं होता।
- Sequence means that all the statements are run one by one in the same order as they are written in the code. There is no decision-making or loop in this.

Example: // c program

① int a = 10;
② int b = 20;
③ int sum = a + b;
④ printf("Sum = %d", sum);

(3) X
(4) ✓



2. Selection Structure (चयन संरचना):

- इस संरचना में हम किसी शर्त (condition) के आधार पर तय करते हैं कि कौन-सा कोड चलेगा। यानी निर्णय लेना।
- In this structure, we decide which code will run based on some condition. That is, taking a decision.

Types:

- if स्टेटमेंट
- if-else स्टेटमेंट
- else-if लैडर
- switch स्टेटमेंट

Example:

```
int marks = 75;
if (marks >= 50) {
    printf("Pass");
}
else {
    printf("Fail");
}
```

43 >= 50 → Fail
75 >= 50 → Pass



3. Iteration Structure (पुनरावृत्ति संरचना):

- जब कोई काम बार-बार दोहराना हो, तब हम loop का उपयोग करते हैं। इसे Iteration कहते हैं।
- When some work has to be repeated again and again, then we use a loop.
This is called Iteration.

Types:

- for loop - जब कितनी बार दोहराना है यह पहले से पता हो
- while loop - पहले condition चेक करता है
- do-while loop - पहले एक बार काम करता है, फिर condition चेक करता है

① `for(initialization; condition; inc/dec) {`

`while((condition)){`
 \equiv
 `inc/dec;`
 `}`

`do {`
 \equiv
`}`
`} while((condition));`



5. Meaningful Naming (सार्थक नामकरण):

- वेरिएबल, फंक्शन और क्लासेज़ के नाम ऐसे होने चाहिए जिससे उनका मतलब समझ में आए।
- Variables, functions, and classes should have names that make sense.

$a = 10$ $Sum = a + b /$ $c = a + b;$

6. Proper Indentation & Comments (सही इंडेंटेशन और टिप्पणियाँ):

- कोड को सही तरीके से format करना (जैसे spacing, line breaks) और जहाँ ज़रूरी हो वहाँ comments लिखना जिससे कोड समझना आसान हो।
- Format the code properly (like spacing, line breaks) and write comments where necessary to make the code easier to understand.



Coding Styles:

- Coding Style या Programming Style एक ऐसा नियमों का समूह (set of rules) होता है जो यह निर्धारित करता है कि प्रोग्रामर किस तरीके से कोड लिखेगा। यह नियम कोड के formatting, naming conventions, indentation, spacing, commenting, और code structure को लेकर होते हैं, ताकि कोड साफ-सुथरा, पढ़ने योग्य (readable), और maintain करने योग्य (maintainable) रहे।
- Coding Style or Programming Style is a set of rules that determines how a programmer should write code. These rules are about code formatting, naming conventions, indentation, spacing, commenting, and code structure, so that the code remains clean, readable, and maintainable.



Coding Style के मुख्य तरीके / तत्व (Elements or Techniques):

1. Indentation (इंडेंटेशन):

- Code में lines के बीच में सही space (tab या spaces) देना, ताकि nested structures (जैसे loops, if-else) साफ़ दिखें।
- Provide proper spacing (tabs or spaces) between lines in code, so that nested structures (like loops, if-else) are clearly visible.

```
for(i=0; i<=5; i++){
    printf();
}
```

```
for(int i=0; i<=5; i++){
    — printf("Hello");
}
```



2. Naming Conventions (नाम रखने के नियम):

- **Variables, functions, classes आदि के लिए meaningful और एक consistent नामों का उपयोग करना।**
- Using meaningful and consistent names for variables, functions, classes, etc.

3: Braces Style (ब्रेसेज़ का तरीका):

- **Control structures (जैसे if, for, while) में braces ({}) को एक समान तरीके से लिखना।**
- Writing braces ({}) in control structures (like if, for, while) in a uniform way.



4: Comments (टिप्पणियाँ लिखना):

- कोड में ऐसे शब्द लिखना जो केवल समझाने के लिए हों, execution में शामिल न हों।
- Writing words in the code that are only for explanation and not involved in the execution.

5. Line Length (लाइन की लंबाई):

- कोड की एक लाइन की अधिकतम लंबाई निर्धारित करना (जैसे 80 या 100 characters) ताकि कोड टूटे नहीं और side scroll न करना पड़े।
- Setting a maximum length for a line of code (e.g. 80 or 100 characters) so that the code doesn't break and you don't have to side scroll.





6. White Space Usage (वाइट स्पेस का सही प्रयोग):

- कोड को readable बनाने के लिए variables, operators के बीच space देना।
- Giving space between variables, operators to make the code readable.

int sum = a + b;

// सही
int sum=a+b;

int sum=a+b;

7. File Naming (फाइल के नाम):

- फाइलों को इस तरह नाम देना कि उनके उद्देश्य का पता चले।
- Naming files in a way that indicates their purpose.



Software Documentation:

- जब एक सॉफ्टवेयर विकसित किया जाता है, तो हमें इसके साथ विभिन्न दस्तावेज़ तैयार करने होते हैं, जैसे उपयोगकर्ता मैनुअल, इंस्टॉलेशन मैनुअल, सुरक्षा शीट्स, एसआरएस दस्तावेज़, डिज़ाइन दस्तावेज़, परीक्षण दस्तावेज़ आदि।
 - When a software is developed we have to prepare various documents with software like user manual, installation manual, safety sheets. SRS document, design documents, test documents, etc.
- इस दस्तावेज़ के आधार पर, यह दो प्रकार के होते हैं: आंतरिक (Internal) और बाहरी (External)।
- Based on this document are of two types Internal and External.



(1): आंतरिक दस्तावेज़ (Internal Documents):

- आंतरिक दस्तावेज़ (Internal Documents) सॉफ्टवेयर विकास प्रक्रिया में उपयोग होने वाले दस्तावेज़ हैं, जो मुख्य रूप से डेवलपर्स, टेस्टर्स और तकनीकी टीम के लिए बनाए जाते हैं। ये दस्तावेज़ सॉफ्टवेयर की डिज़ाइन, कोडिंग, परीक्षण और रखरखाव से संबंधित जानकारी प्रदान करते हैं।
- Internal documents are documents used in the software development process, primarily created for developers, testers, and the technical team. These documents provide information related to the design, coding, testing, and maintenance of the software.

ex → SRS document
→ Testing document
→ Designing " } internal document



(2): External Documents:

- सॉफ्टवेयर इंजीनियरिंग में एक्स्टर्नल डॉक्यूमेंट (External Document) का तात्पर्य उन दस्तावेजों से होता है जो सॉफ्टवेयर को समझने, उपयोग करने, या उसका रख-रखाव करने के लिए उपयोगकर्ताओं और ग्राहकों को जानकारी प्रदान करते हैं।
- External documents in software engineering refer to documents that provide information to users and customers to understand, use, or maintain the software.

① → User manual
→ Installation manual
→ Guideline document
→ Help document

} External document

Chapter No-5 (Requirement Design and Implementation)

Q 1: What are the characteristics of good software design?

Q 1: अच्छे सॉफ्टवेयर डिजाइन की विशेषताएं क्या हैं?

Q 2: Differentiate between Cohesion and Coupling?

Q 2: Cohesion और Coupling के बीच अंतर बताइए।

Q 3 What is the purpose of DFD? What are the notations used for the DFD?

Q 3: DFD का उद्देश्य क्या है? DFD के लिए कौन से संकेतन प्रयुक्ति किए जाते हैं?

Q 4: Differentiate between Data Flow Diagram and Data Dictionary.

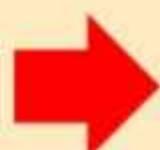
Q 4: Data Flow आरेख और डेटा Dictionary. के बीच अंतर बताएं।

Q 5: Write short note on structured Coding Techniques.

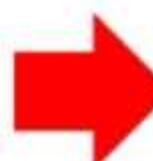
Q 5: Structure कोडिंग तकनीकों पर संक्षिप्त टिप्पणी लिखें।



Chapter No-6 (Software Testing.)

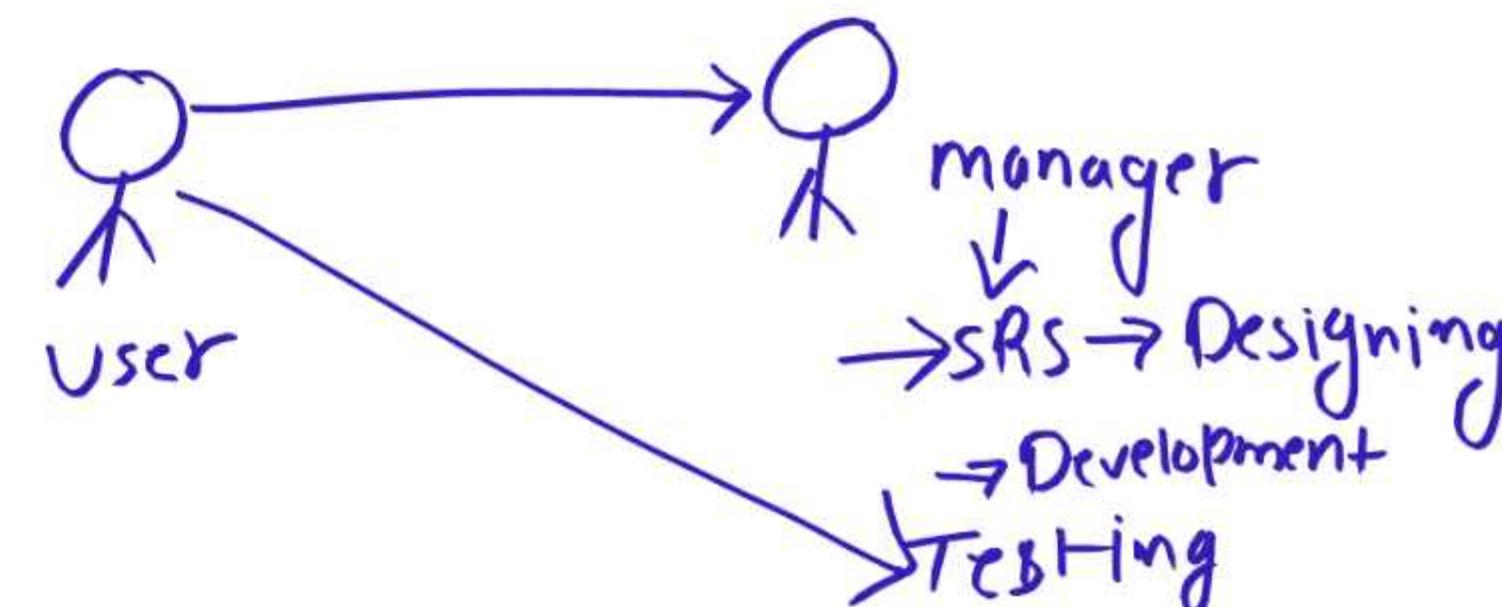
**Syllabus :**

- concept of Testing
- * • Testing type cycle (V-Model)
- * • Verification V/s Validations
- * • Unit Testing, Black Box Testing
- White Box Testing
- Integration testing
- System testing
- Configuration management
- Overview of test
- cases.



Concept of Testing:

- सॉफ्टवेयर परीक्षण एक प्रक्रिया है, जिसमें सॉफ्टवेयर सॉफ्टवेयर (कार्यक्षमता) का आकलन शामिल है। ✓
- इसका उद्देश्य यह पता लगाना है कि विकसित सॉफ्टवेयर बिना किसी त्रुटि (दोष) के काम कर रहा है और ग्राहक द्वारा निर्धारित आवश्यकताओं (आवश्यकताओं) को पूरा किया जा रहा है या नहीं।
- Software testing is a process that involves assessing the functionality of a software.
- Its purpose is to find out whether the developed software is working without any errors (defects) and whether the requirements set by the customer are being met or not.





Some properties that a testing process indicates are :

(टेस्टिंग प्रक्रिया से जो बातें पता चलती हैं, वे हैंः)

- क्या सिस्टम अपने डिज़ाइन और विकास के अनुसार सभी आवश्यकताओं को पूरा करता है?
- Does the system meet all the requirements as per its design and development?

- क्या सिस्टम हर प्रकार के इनपुट का सही तरीके से जवाब देता है?
- Does the system respond properly to all types of inputs?

- क्या सिस्टम अपने काम को उचित समय सीमा में पूरा करता है?
- Does the system complete its task within a reasonable time frame?

- क्या सिस्टम उपयोग करने में आसान है?
- Is the system easy to use?

- क्या सिस्टम को उसके निर्धारित environment में स्थापित और चलाया जा सकता है?
- Can the system be installed and run in its intended environment?

- क्या सिस्टम वांछित परिणाम प्रदान करता है?
- Does the system provide the desired results?



Software Testing is of two types :

(1): Manual Testing:

(2): Automatic Testing

(1): Manual Testing:

- मैन्युअल टेस्टिंग में टेस्ट केस को बिना किसी टूल या स्क्रिप्ट की मदद के हाथ से किया जाता है।
मैन्युअल टेस्टिंग को एक QA एनालिस्ट द्वारा किया जाता है। टेस्टर दिए गए ऐप्लिकेशन या सॉफ्टवेयर की सभी जरूरी फीचर्स को जांचता है।
- In manual testing, test cases are created by hand without the help of any tools or scripts. Manual testing is done by a QA analyst. The tester checks all the necessary features of the given application or software.

↳ costly
↳ Time consuming



(2): Automatic Testing:

- ऑटोमेटेड टेस्टिंग में कोड या स्क्रिप्ट लिखकर टेस्ट को अपने आप चलाया जाता है।
 - इसमें टेस्टिंग करने के लिए ऑटोमेटेड टूल्स का उपयोग किया जाता है।
 - यह मैन्युअल टेस्टिंग के मुकाबले तेज और सटीक होता है।
 - बार-बार दोहराए जाने वाले टेस्ट में समय और मेहनत बचाता है।
 - बड़े और जटिल प्रोजेक्ट्स के लिए यह अधिक उपयोगी है।
-
- In automated testing, tests are run automatically by writing code or scripts.
 - Automated tools are used for testing.
 - It is faster and more accurate than manual testing.
 - It saves time and effort in repeated tests.
 - It is more useful for large and complex projects.

↳ low cost
↳ high accuracy.
↳ speed ↑



Difference between Manual testing and Automated testing:

Manual Testing.	Automated Testing
It is less accurate as it involves Manual Interface	It is more accurate as it is performed by Tools/Scripts
It is a Time-Consuming Process.	It is comparatively a Fast Process.
Manual Testing ensures a good user experience to end user as system is tested by human observations	It can ensure good user experience, since machine lacks in human observations
Random Testing is possible in Manual Testing.	Automation does not allow random testing.



Manual Testing.	Automated Testing
यह कम सटीक है क्योंकि इसमें मैन्युअल इंटरफ़ेस शामिल है	यह अधिक सटीक है क्योंकि यह टूल/स्क्रिप्ट द्वारा निष्पादित किया जाता है
यह एक समय लेने वाली प्रक्रिया है।	यह तुलनात्मक रूप से एक तेज़ प्रक्रिया है।
मैनुअल परीक्षण अंतिम उपयोगकर्ता के लिए एक अच्छा उपयोगकर्ता अनुभव सुनिश्चित करता है क्योंकि सिस्टम का परीक्षण मानवीय अवलोकन द्वारा किया जाता है	यह अच्छा उपयोगकर्ता अनुभव सुनिश्चित कर सकता है, क्योंकि मशीन में मानवीय अवलोकन की कमी होती है
मैनुअल परीक्षण में रैंडम परीक्षण संभव है।	Automatically random परीक्षण की अनुमति नहीं देता है।



Manual Testing.	Automatic Testing
No need for programming in manual testing.	Programming Knowledge is must in Automating Testing.
Manual Testing needs more costs as it involves hiring of expert professionals to perform testing	It saves cost as once system is developed, it can be used for a long time.



Manual Testing.	Automatic Testing
मैन्युअल परीक्षण में प्रोग्रामिंग की कोई आवश्यकता नहीं	प्रोग्रामिंग Language automatic test होती है।
मैनुअल परीक्षण में अधिक लागत लगती है क्योंकि इसमें परीक्षण करने के लिए विशेषज्ञ पेशेवरों को काम पर रखना पड़ता है	इससे लागत बचती है, क्योंकि एक बार सिस्टम विकसित हो जाने पर इसका उपयोग लम्बे समय तक किया जा सकता है।



Difference Between Static & Dynamic Testing.

Static Testing.	Dynamic Testing
It is less accurate as it involves Manual Interface	It is performed at the later stage of software development.
No execution required	Process execution required
Static Testing prevent the defects.	Dynamic Testing find and fix the defects
It is less Costly Testing.	Dynamic Testing is Highly Costly
It takes shorter time.	It takes longer time.



Static Testing.	Dynamic Testing
यह कम स्टीक है क्योंकि इसमें मैन्युअल इंटरफ़ेस शामिल है।	यह सॉफ्टवेयर विकास के बाद के चरण में किया जाता है।
सॉफ्टवेयर को चलाने की ज़रूरत नहीं।	सॉफ्टवेयर को चलाने की ज़रूरत होती है।
Static परीक्षण दोषों को रोकता है।	Dynamic परीक्षण दोषों का पता लगाता है और उन्हें ठीक करता है।
यह कम खर्चीला परीक्षण है।	Dynamic परीक्षण अत्यधिक महंगा है।
इसमें कम समय लगता है।	इसमें अधिक समय लगता है।



Difference Between Verification & Validation Testing.

Verification	Validation
Verification is static process of verifying documents, design, codes and program.	Validation is dynamic process of validating and testing the actual product
Verification checks whether software confirm to specifications.	Validation checks whether software is providing desired results
Verification is done by QA (Quality Assurance team.	Validation is done by Testing Team
Verification uses methods like Review, Inspections, Walk-through etc	It uses methods like Black Box Testing, White Box Testing etc.



Verification	Validation
It comes before Validation.	It comes after Verification.
It can detect errors not detected by Validation	It also detects errors that Verification cannot catch
It targets specification Designs and Software Architecture	It targets complete unit (product), module and final product



Verification	Validation
Verification दस्तावेजों, डिजाइन, कोड और प्रोग्राम के सत्यापन की स्थैतिक प्रक्रिया है।	Validation वास्तविक उत्पाद के सत्यापन और परीक्षण की गतिशील प्रक्रिया है
Verification यह जांचता है कि सॉफ्टवेयर विनिर्देशों के अनुरूप है या नहीं।	Validation यह जांचता है कि सॉफ्टवेयर Desired परिणाम प्रदान कर रहा है या नहीं
Verification QA (गुणवत्ता आश्वासन टीम) द्वारा किया जाता है।	Validation परीक्षण टीम द्वारा किया जाता है
Verification में समीक्षा, निरीक्षण, वॉक-थ्रू आदि विधियों का उपयोग किया जाता है	इसमें ब्लैक बॉक्स टेस्टिंग, व्हाइट बॉक्स टेस्टिंग आदि विधियों का उपयोग किया जाता है।



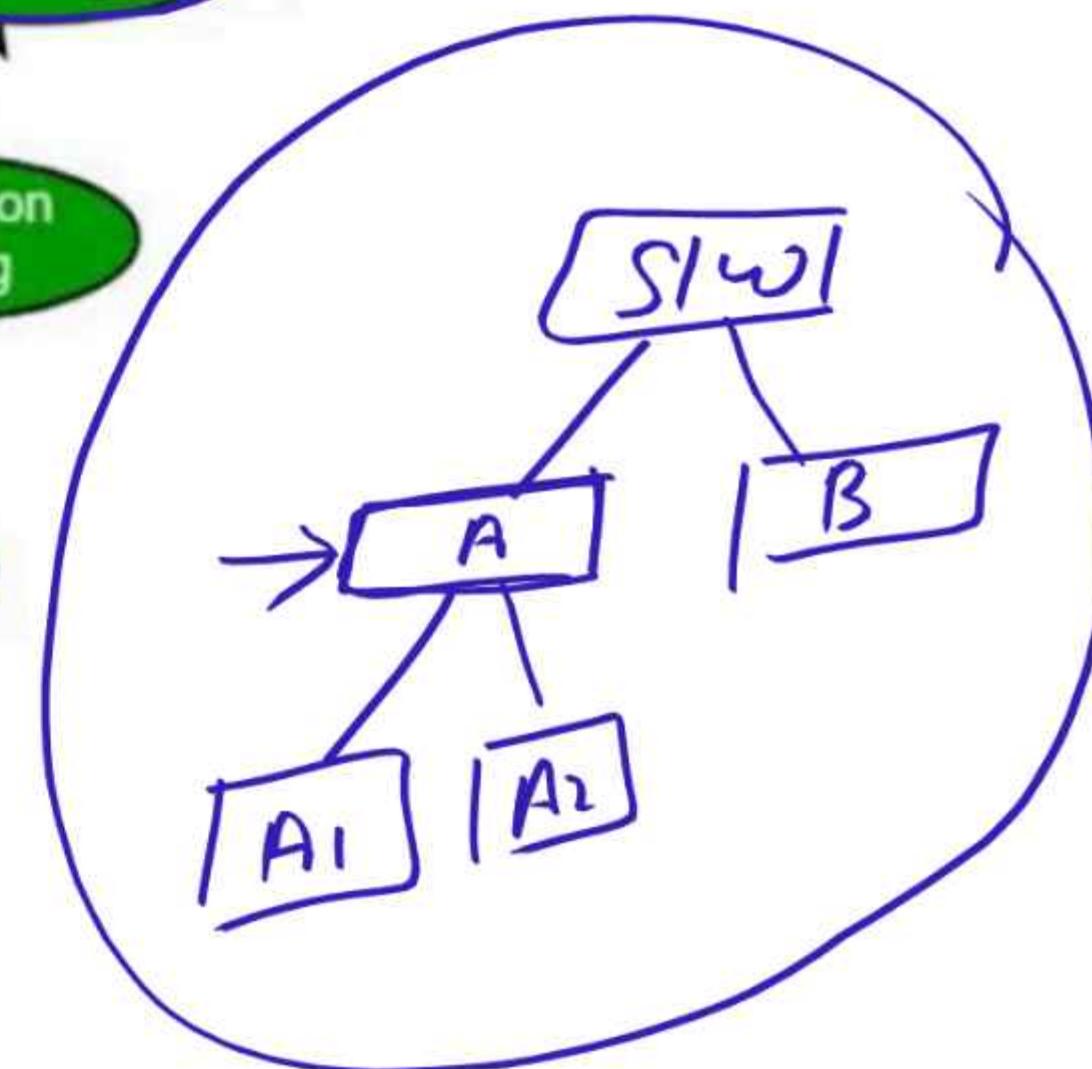
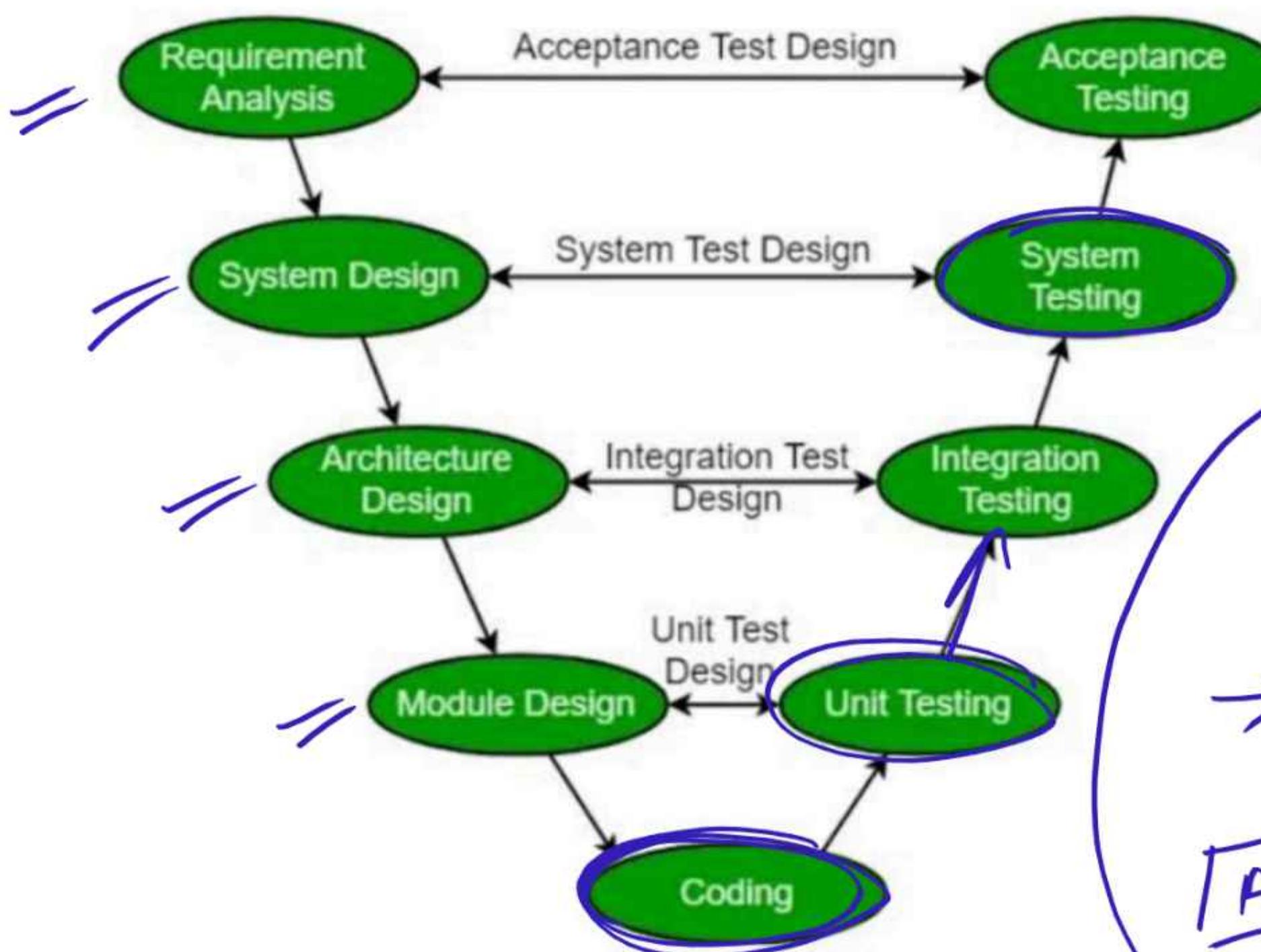
Verification	Validation
यह Verification से पहले आता है।	यह Validation के बाद आता है।
यह Verification द्वारा पता न लगाई गई त्रुटियों का पता लगा सकता है	यह उन त्रुटियों का भी पता लगाता है जिन्हें Validation नहीं पकड़ सकता
यह specification डिजाइन और सॉफ्टवेयर architecture target करता है	यह सम्पूर्ण इकाई (उत्पाद), मॉड्यूल और अंतिम उत्पाद को लक्ष्य करता है



Testing Type Cycle(V—Model):

- लेकिन एक ऐसा SDLC मॉडल है जो मुख्य रूप से टेस्टिंग पर ध्यान केंद्रित करता है। इसे V-Model कहा जाता है। इसे वेरिफिकेशन और वेलिडेशन मॉडल भी कहते हैं। यह वॉटरफॉल मॉडल का एक विस्तार है, जिसमें हर डेवलपमेंट स्टेज के साथ एक टेस्टिंग चरण जुड़ा होता है।
- there is one SDLC Model which primarily focuses on testing. This is called V-Model. It is also known as verification and Validation Model. It is extension of the Waterfall Model and a testing phase is associated with each corresponding development stage.
- सॉफ्टवेयर डिजाइन जीवन चक्र का वी-मॉडल अक्षर 'वी' जैसा दिखता है, एक तरफ सत्यापन चरण होता है और दूसरी तरफ सत्यापन चरण होता है।
- The V-Model of Software Design Life Cycle resembles Letter 'V', on one side is a verification step and on another side is a validation step.

Verification → Validation





1. आवश्यकता विश्लेषण (Requirement Analysis):

Objective:

- Understanding user needs.
- Finding out what the expectations are from the software.
- उपयोगकर्ता की जरूरतों को समझना।
- यह पता लगाना कि सॉफ्टवेयर से क्या अपेक्षाएं हैं।

Acceptance Testing:

- It checks whether the finished software meets the customer's requirements.
- यह चेक करता है कि तैयार सॉफ्टवेयर ग्राहक की आवश्यकताओं को पूरा करता है या नहीं।



2. सिस्टम डिजाइन (System Design):

Objective:

- Defining the overall structure and functionalities of the software.
- Determining the hardware and software requirements.
- सॉफ्टवेयर की पूरी संरचना और कार्यात्मकताओं को परिभाषित करना।
- हार्डवेयर और सॉफ्टवेयर की आवश्यकताओं को निर्धारित करना।

System Testing:

- The entire system is tested to see whether it is working as per all the requirements or not.
- पूरे सिस्टम को टेस्ट करके यह देखा जाता है कि वह सभी आवश्यकताओं के अनुसार काम कर रहा है या नहीं।



3. आर्किटेक्चर डिजाइन (Architectural Design):

Objective:

- Dividing the software into smaller modules.
- Deciding how the modules will be connected to each other.
- सॉफ्टवेयर को छोटे मॉड्यूल्स में विभाजित करना।
- यह तय करना कि मॉड्यूल्स आपस में कैसे जुड़ेंगे।

Integration Testing:

- Modules are combined to ensure that they work together properly.
- मॉड्यूल्स को जोड़कर यह सुनिश्चित किया जाता है कि वे आपस में सही तरीके से काम कर रहे हैं।



4. मॉड्यूल डिज़ाइन (Module Design):

Objective:

- Defining the structure and functions of each module.
- Deciding what each module will do.
- हर मॉड्यूल की संरचना और कार्यों को परिभाषित करना।
- यह तय करना कि हर मॉड्यूल क्या करेगा।

Unit Testing:

- Each module is tested separately to ensure it is working correctly.
- हर मॉड्यूल को अलग से टेस्ट किया जाता है ताकि यह सुनिश्चित किया जा सके कि वह सही तरीके से काम कर रहा है।



5. कोडिंग (Coding):

Objective:

- Writing software code according to design documents.
- डिजाइन दस्तावेज़ों के अनुसार सॉफ्टवेयर कोड लिखना।

Related Testing:

- All types of testing (Unit, Integration, System and Acceptance) start after coding is completed.
- सभी प्रकार के परीक्षण (यूनिट, एकीकरण, सिस्टम और स्वीकृति) कोडिंग पूरी होने के बाद शुरू होते हैं।



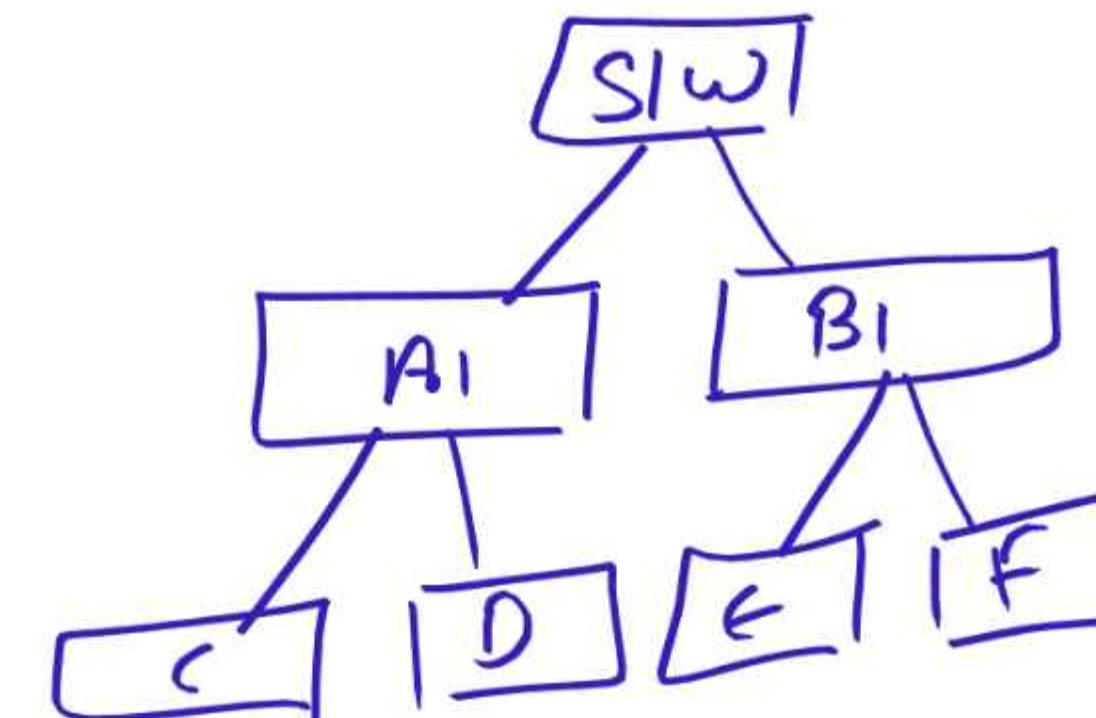
→ V मॉडल के फायदे (Advantages of V model):

- Testing is done at every stage, so errors are caught early.
 - Customer requirements are better understood and fulfilled.
 - Project time and cost are saved.
-
- हर स्टेज पर टेस्टिंग होती है, जिससे ऐरर जल्दी पकड़ी जाती हैं।
 - ग्राहकों की आवश्यकताओं को बेहतर तरीके से समझा और पूरा किया जाता है।
 - प्रोजेक्ट का समय और लागत बचती है।



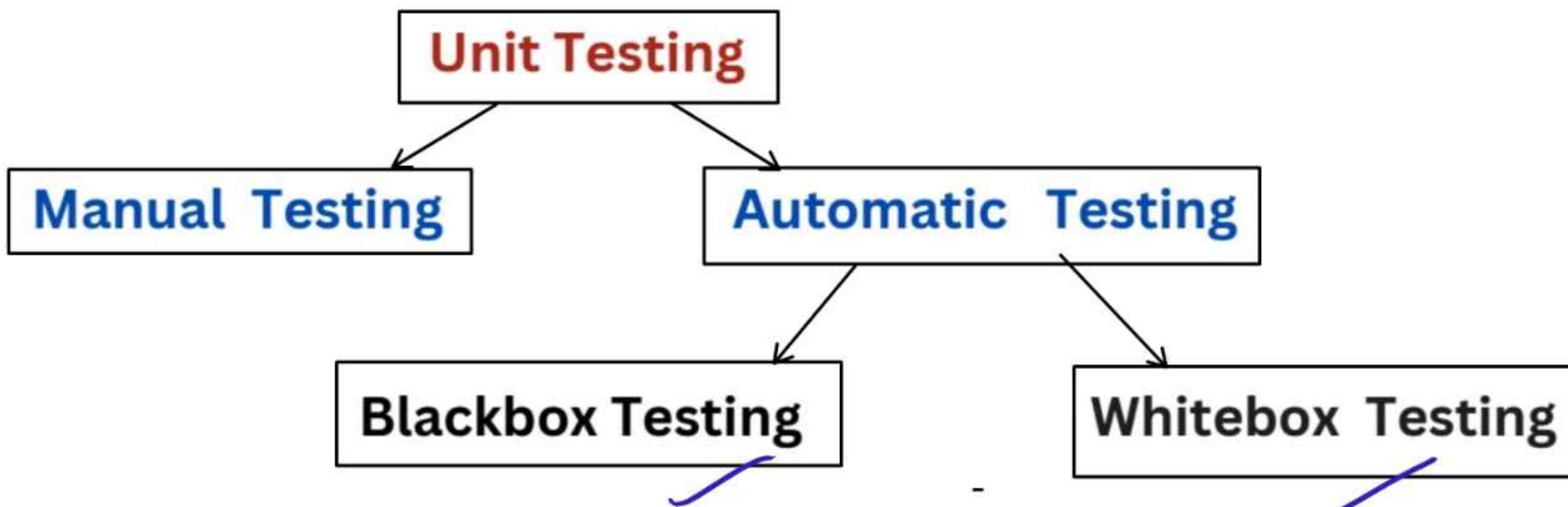
Unit Testing:

- Unit Testing means testing small parts (units) of a software program separately to ensure that they are working correctly. This is an important process done at the beginning of software development.
- Unit Testing (यूनिट परीक्षण) का मतलब है सॉफ्टवेयर प्रोग्राम के छोटे-छोटे हिस्सों (यूनिट्स) को अलग-अलग जांचना ताकि यह सुनिश्चित हो सके कि वे सही तरीके से काम कर रहे हैं। यह सॉफ्टवेयर डेवलपमेंट की शुरुआत में की जाने वाली एक महत्वपूर्ण प्रक्रिया है।





Unit Testing के प्रकार (Types of Unit Testing):





(1): Manual Testing:

- In this, the developer himself tests the units.
 - This takes more time and the chances of errors increase.
 - Example: Running a program manually to see if the output is correct or not.
-
- इसमें डेवलपर खुद यूनिट्स को टेस्ट करता है।
 - इसे करने में समय अधिक लगता है और त्रुटियों की संभावना बढ़ जाती है।
 - उदाहरण: किसी प्रोग्राम को मैन्युअली रन करके देखना कि आउटपुट सही है या नहीं।



(2): Automated Testing:

- In this, testing is done through automated tools.
- It is fast and accurate.
- Example : Use of tools like JUnit, NUnit, or PyTest.
- इसमें टेस्ट को ऑटोमेटेड टूल्स के ज़रिए किया जाता है।
- यह तेज़ और सटीक होता है।
- उदाहरण: JUnit, NUnit, या PyTest जैसे टूल्स का इस्तेमाल।



→ (a): Black box testing:

- Black Box Testing is a technique of software testing in which only the input and output of the system are tested without looking at its internal workings (internal logic).
- It is also called **Behavioral Testing** because in this the behavior of the software is checked.
- Black Box Testing (ब्लैक बॉक्स टेस्टिंग) सॉफ्टवेयर टेस्टिंग की एक ऐसी तकनीक है जिसमें सिस्टम की आंतरिक कार्यप्रणाली (Internal Logic) को बिना देखे केवल उसके इनपुट और आउटपुट का परीक्षण किया जाता है।
- इसे **Behavioral Testing** (व्यवहार परीक्षण) भी कहा जाता है क्योंकि इसमें सॉफ्टवेयर का व्यवहार चेक किया जाता है।



Black Box Testing के प्रकार (Types of Black Box Testing):

1. Functional Testing (कार्यात्मक परीक्षण):

- All the functionalities of the software are tested.
- It is observed whether the software is working as per the user requirements or not.
- Example: Whether the login is happening or not when the correct username and password is given in the login system.
- सॉफ्टवेयर के सभी फंक्शनलिटी की जांच की जाती है।
- यह देखा जाता है कि सॉफ्टवेयर उपयोगकर्ता की जरूरतों के अनुसार काम कर रहा है या नहीं।
- उदाहरण: लॉगिन सिस्टम में सही यूजरनेम और पासवर्ड देने पर लॉगिन हो रहा है या नहीं।



2. Non-Functional Testing (गैर-कार्यात्मक परीक्षण):

- The software's performance, security, usability, and stability are tested.
- Example: How fast a website loads
- सॉफ्टवेयर की प्रदर्शन क्षमता, सुरक्षा, उपयोगिता, और स्थिरता की जांच की जाती है।
- उदाहरण: एक वेबसाइट कितनी तेज़ी से लोड हो रही है।

3. Regression Testing (रीग्रेशन टेस्टिंग):

- When a change is made to the software, it ensures that the rest of the parts are working properly
- जब सॉफ्टवेयर में कोई बदलाव किया जाता है, तो यह सुनिश्चित किया जाता है कि बाकी हिस्से सही से काम कर रहे हैं



Black Box Testing के फायदे(Advantages of Black Box Testing):

- It can be done even if you don't know the code.
 - Helps in understanding the working of the software from the user's point of view.
 - Supports both functional and non-functional testing.
-
- कोड की जानकारी न होने पर भी इसे किया जा सकता है।
 - उपयोगकर्ता की दृष्टि से सॉफ्टवेयर की कार्यप्रणाली को समझने में मदद करता है।
 - फंक्शनल और नॉन-फंक्शनल दोनों प्रकार की टेस्टिंग को सपोर्ट करता है।



Black Box Testing की सीमाएँ (Limitations of Black Box Testing):

- Cannot catch bugs hidden in internal code.
 - Test coverage may be low as it is limited to inputs and outputs only.
 - It is difficult to find potential bugs in more complex software.
-
- आंतरिक कोड में छुपे बग्स को नहीं पकड़ सकता।
 - टेस्ट कवरेज (Coverage) कम हो सकता है क्योंकि यह केवल इनपुट और आउटपुट तक सीमित है।
 - अधिक जटिल सॉफ्टवेयर में संभावित बग्स को खोजने में कठिनाई होती है।



Whitebox testing:

- White Box Testing is a technique of software testing in which the internal structure, code, and logic of the software are tested.
- It is also called Clear Box Testing, Open Box Testing, or Structural Testing. In this testing, the person doing the testing has complete knowledge about the code and its working.
- White Box Testing (व्हाइट बॉक्स टेस्टिंग) सॉफ्टवेयर टेस्टिंग की एक तकनीक है जिसमें सॉफ्टवेयर के आंतरिक स्ट्रक्चर, कोड, और लॉजिक को टेस्ट किया जाता है।
- इसे Clear Box Testing, Open Box Testing, या Structural Testing भी कहा जाता है। इस टेस्टिंग में, टेस्ट करने वाले व्यक्ति को कोड और उसके वर्किंग के बारे में पूरी जानकारी होती है।



White Box Testing की विशेषताएँ (Features of White Box Testing):

1. कोड-केंद्रित टेस्टिंग (Code-centric testing):

- In this the code of the software is checked in detail.
- इसमें सॉफ्टवेयर के कोड को डिटेल में चेक किया जाता है।

2. डेवलपर्स द्वारा उपयोगी (Useful by developers):

- Usually this is done by developers or people who have deep knowledge of coding.
- आमतौर पर डेवलपर्स या ऐसे व्यक्ति इसे करते हैं, जिन्हें कोडिंग की गहरी जानकारी होती है।

3. एरर की गहराई तक जाना (Digging deeper into the error):

- It helps in finding bugs, incorrect code logic, and redundant code.
- यह बग्स, गलत कोड लॉजिक, और अनावश्यक कोड को खोजने में मदद करता है।



4. लॉजिक पर ध्यान (. Focus on logic):

- It checks whether all the conditions and loops are working correctly.
- यह चेक करता है कि सभी कंडीशन्स और लूप्स सही तरीके से काम कर रहे हैं या नहीं।

White Box Testing के फायदे (Advantages of White Box Testing):

- It is able to catch hidden errors and vulnerabilities in the code
- It helps in identifying and removing unnecessary code.
- It ensures that every part of the code is tested.
 - यह कोड में छुपे हुए एरर और कमज़ोरियों को पकड़ने में सक्षम है
 - अनावश्यक कोड को पहचानकर उसे हटाने में मदद करता है।
 - यह सुनिश्चित करता है कि कोड का हर हिस्सा टेस्ट किया गया है।



White Box Testing की सीमाएँ (Limitations of White Box Testing):

- This can be difficult and time-consuming to do for large and complex software.
 - If the code changes, the testing also needs to be updated.
 - To do this, the developer must have knowledge of programming and coding.
-
- बड़े और जटिल सॉफ्टवेयर के लिए इसे करना मुश्किल और समय-consuming हो सकता है।
 - अगर कोड में बदलाव होता है, तो टेस्टिंग को भी अपडेट करना पड़ता है।
 - इसे करने के लिए डेवलपर को प्रोग्रामिंग और कोडिंग का ज्ञान होना चाहिए।



Difference between Blackbox & Whitebox Testing.

Black box testing:	White box testing:
<p>Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.</p>	<p>White box testing is the software testing method in which internal structure is being known to tester who is going to test the software</p>
<p>This type of testing is carried out by testers.</p>	<p>Generally, this type of testing is carried out by <u>software developers</u>.</p>
<p>Implementation Knowledge is <u>not</u> required to carry out Black Box Testing.</p>	<p>Implementation Knowledge is required to carry out White Box Testing.</p>



Black box testing:	White box testing:
Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
Black box testing means functional test or external testing	White box testing means structural test or interior testing.



Black box testing:	White box testing:
ब्लैक बॉक्स परीक्षण सॉफ्टवेयर परीक्षण विधि है जिसका उपयोग कोड या प्रोग्राम की आंतरिक संरचना को जाने बिना सॉफ्टवेयर का परीक्षण करने के लिए किया जाता है।	व्हाइट बॉक्स परीक्षण सॉफ्टवेयर परीक्षण पद्धति है जिसमें आंतरिक संरचना परीक्षक को ज्ञात हो जाती है जो सॉफ्टवेयर का परीक्षण करने जा रहा है
इस प्रकार का परीक्षण परीक्षकों द्वारा किया जाता है।	आमतौर पर, इस प्रकार का परीक्षण सॉफ्टवेयर डेवलपर्स द्वारा किया जाता है।
ब्लैक बॉक्स परीक्षण करने के लिए implementation knowledge की आवश्यकता नहीं है।	व्हाइट बॉक्स परीक्षण करने के लिए implementation Knowledge की आवश्यकता होती है।

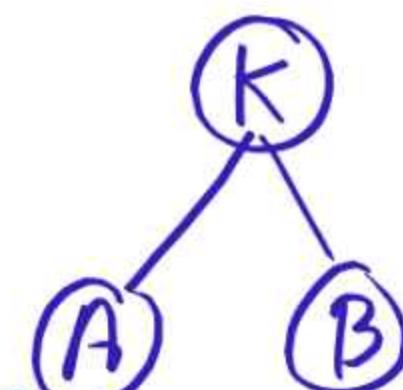


Black box testing:	White box testing:
ब्लैक बॉक्स परीक्षण करने के लिए प्रोग्रामिंग ज्ञान की आवश्यकता नहीं है।	व्हाइट बॉक्स परीक्षण करने के लिए प्रोग्रामिंग ज्ञान की आवश्यकता होती है।
परीक्षण उच्च स्तर के परीक्षणों पर लागू होता है जैसे सिस्टम परीक्षण, स्वीकृति परीक्षण।	परीक्षण निम्न स्तर के परीक्षण जैसे यूनिट परीक्षण, Integration परीक्षण पर लागू होता है।
ब्लैक बॉक्स परीक्षण का अर्थ है कार्यात्मक परीक्षण या बाह्य परीक्षण	व्हाइट बॉक्स परीक्षण का अर्थ है structural परीक्षण या आंतरिक परीक्षण।



Integration Testing:

- Integration testing is an important part of the educational process. It involves assembling different parts or components to check if they work correctly in combination.
- Whenever a project is developed, many small architectures are created in it. After unit testing these modules, integration testing ensures that these modules are working in a correct manner.
- इंटीग्रेशन इलिजेक्शन शैक्षणिक प्रक्रिया का एक महत्वपूर्ण हिस्सा है। अलग-अलग नमूनों या हिस्सों (घटकों) को एकत्र करने में यह जांच की जाती है कि वे संयोजन में सही ढंग से काम कर रहे हैं या नहीं।
- जब भी कोई प्रोजेक्ट डेवलप किया जाता है तो उसमें कई छोटे-छोटे आर्किटेक्चर बनाए जाते हैं। इन मॉड्यूल्स यूनिट की जांच के बाद, इंटीग्रेशन परीक्षण यह सुनिश्चित करता है कि ये मॉड्यूल एक सही तरीके से काम कर रहे हैं।





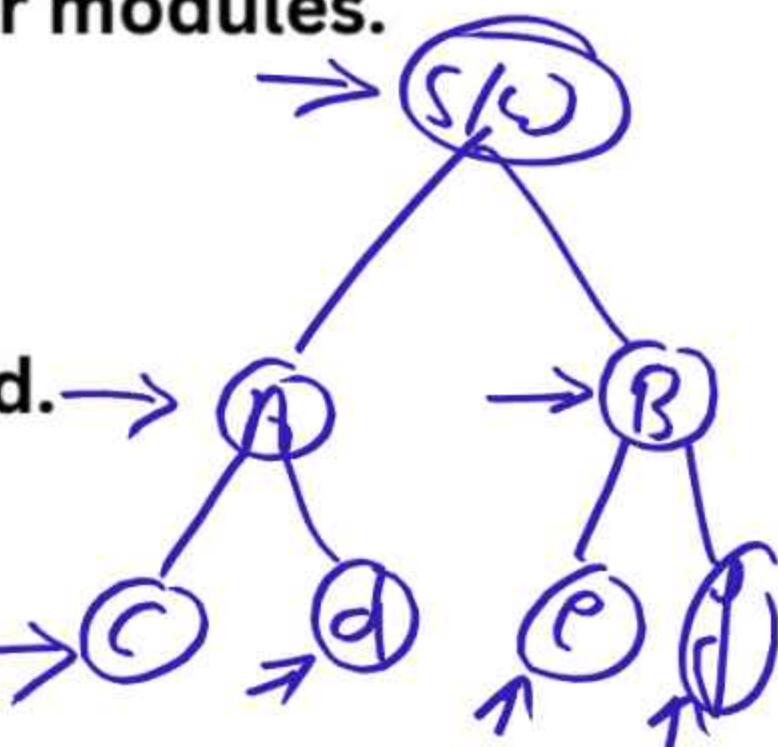
इंटीग्रेशन टेस्टिंग के प्रकार (Types of Integration Testing):

1. Bottom-up approach:

- Starting with smaller modules, they are combined into larger modules.
- मॉड्यूल्स से शुरूआत करके उन्हें बड़े मॉड्यूल्स के साथ जोड़ा जाता है।

2. Top-down approach:

- Starting with the largest module, smaller modules are added.
- सबसे बड़े मॉड्यूल से शुरूआत करके छोटे मॉड्यूल्स को जोड़ा जाता है।



3. Sandwich Approach:

- It uses both bottom-up and top-down techniques.
- इसमें बॉटम-अप और टॉप-डाउन दोनों तकनीकों का उपयोग होता है।

4. Big Bang Approach:

- All modules are combined and tested simultaneously.
- सभी मॉड्यूल्स को एक साथ जोड़कर टेस्ट किया जाता है।



→ इंटीग्रेशन टेस्टिंग के फायदे (Benefits of Integration Testing):

1. Detecting errors:

- During integration testing it is easy to find out if there is any bug when the modules are connected together.
- इंटीग्रेशन टेस्टिंग के दौरान यह आसानी से पता चल जाता है कि जब मॉड्यूल्स को एक साथ जोड़ा गया, तो उनमें किसी तरह की गलती (bug) है या नहीं।

2. Data flow checking:

- This ensures that the data flow from one module to another is happening correctly.
- यह सुनिश्चित करता है कि एक मॉड्यूल से दूसरे मॉड्यूल में डेटा का प्रवाह (data flow) सही तरीके से हो रहा है।



3. Real-world usage testing:

- After connecting different modules, it is determined how the software will work in real situations.
- अलग-अलग मॉड्यूल्स को जोड़ने के बाद यह पता चलता है कि सॉफ्टवेयर असली परिस्थिति में कैसे काम करेगा।

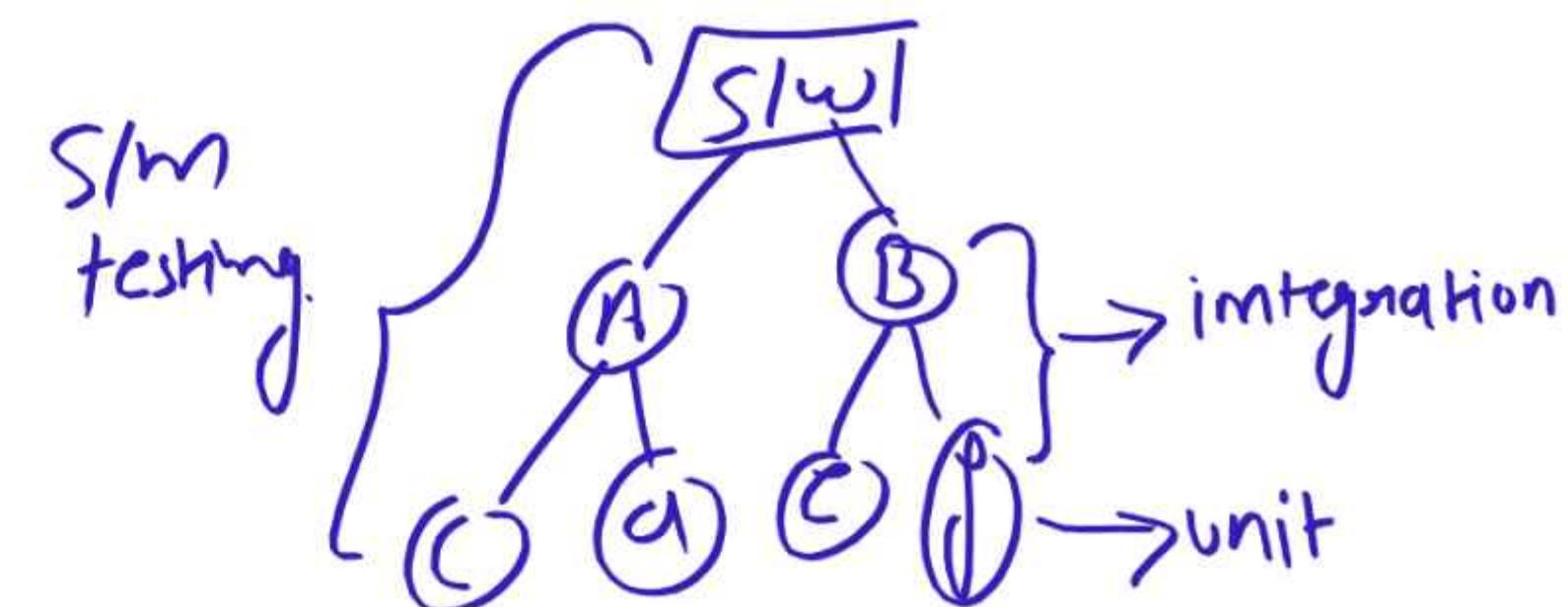
4. Saves time:

- Identifying problems early can save time that would otherwise be needed to correct them later.
- समस्याओं को शुरुआत में ही पहचान लेने से बाद में सुधार के लिए लगने वाले समय को बचाया जा सकता है।



System Testing:

- System testing is a process of software testing in which the entire software is tested as a complete system.
- In this, it is seen whether every part (features, functions) of the software is working properly or not. It is tested from the perspective of the end user.
- System testing checks the functionality of the entire system, including hardware, software, network, and data. It ensures that the software is fully ready and fit for actual use.





System Testing:

- सिस्टम टेस्टिंग सॉफ्टवेयर टेस्टिंग की एक प्रक्रिया है, जिसमें पूरे सॉफ्टवेयर को एक संपूर्ण सिस्टम के रूप में जांचा जाता है।
- इसमें यह देखा जाता है कि सॉफ्टवेयर का हर हिस्सा (फीचर्स, फ़ंक्शंस) सही तरीके से काम कर रहा है या नहीं। इसे अंतिम उपयोगकर्ता (End User) के नजरिए से परखा जाता है।
- सिस्टम टेस्टिंग में हार्डवेयर, सॉफ्टवेयर, नेटवर्क और डेटा को मिलाकर पूरे सिस्टम की कार्यक्षमता की जांच की जाती है। यह सुनिश्चित करता है कि सॉफ्टवेयर पूरी तरह से तैयार है और वास्तविक उपयोग के लिए फिट है।



→ सिस्टम टेस्टिंग के फायदे(Advantages of System Testing):

1. Testing the entire software:

- System testing involves testing the entire software to ensure that all modules and features are working correctly.
- सिस्टम टेस्टिंग में पूरे सॉफ्टवेयर का परीक्षण किया जाता है, जिससे यह सुनिश्चित हो सके कि सभी मॉड्यूल और फीचर्स सही तरीके से काम कर रहे हैं।

2. User Friendly System:

- It is checked that the software is simple and easy to use for the user.
- यह जांचा जाता है कि सॉफ्टवेयर उपयोगकर्ता के लिए सरल और उपयोग में आसान है।

3. Detection of bugs:

- Any type of errors (bugs) can be identified and fixed before production.
- किसी भी प्रकार की त्रुटियों (bugs) को उत्पादन (production) से पहले ही पहचानकर उन्हें ठीक किया जा सकता है।



4. Meeting the customer requirements:

- It is ensured that the software is meeting all the requirements and expectations of the customer.
- यह सुनिश्चित किया जाता है कि सॉफ्टवेयर ग्राहक की सभी आवश्यकताओं और अपेक्षाओं को पूरा कर रहा है।



Configuration Management:

- Configuration management is an important part of the software development process. Its main purpose is to ensure that all files, code, and documents used in a software project are organized and tracked.
 - It can also be called the "Version Control System" of the software.
-
- कॉन्फिगरेशन मैनेजमेंट सॉफ्टवेयर विकास प्रक्रिया का एक महत्वपूर्ण हिस्सा है। इसका मुख्य उद्देश्य यह सुनिश्चित करना है कि सॉफ्टवेयर प्रोजेक्ट में उपयोग होने वाली सभी फाइलें, कोड, और डॉक्युमेंट्स व्यवस्थित और ट्रैक किए जा सकें।
 - इसे सॉफ्टवेयर का “Version Control System” भी कहा जा सकता है।



Main functions:

1. Version Control:

- Tracking every change and ensuring which version of the software is running.
- हर बदलाव को ट्रैक करना और यह सुनिश्चित करना कि सॉफ्टवेयर का कौन-सा वर्जन चल रहा है।

2. Backup & Recovery:

- If data gets accidentally deleted, then recover it from the backup.
- यदि गलती से डेटा डिलीट हो जाए, तो बैकअप से उसे पुनः प्राप्त करना।

3. Tracking:

- Knowing who changed the code or document, when, and why.
- यह जानना कि किसने, कब, और क्यों कोड या डॉक्युमेंट में बदलाव किया।

4. Collaboration:

- Team members can work together without conflict.
- टीम के सदस्य बिना किसी संघर्ष के (conflict) एक साथ काम कर सकें।



Advantage of Configuration:

- The chances of making mistakes are reduced.
 - All changes are transparent.
 - It is easier to manage big projects.
 - Coordination among the team improves.
-
- गलती होने की संभावना कम हो जाती है।
 - सभी बदलाव ट्रांसपरेंट (पारदर्शी) होते हैं।
 - बड़े प्रोजेक्ट्स को मैनेज करना आसान होता है।
 - टीम के बीच तालमेल बेहतर होता है।



Overview of test cases:

- A test case is a document or set of steps in software engineering designed to test whether a particular part (feature or function) of a software works correctly.
- टेस्ट केस सॉफ्टवेयर इंजीनियरिंग में एक दस्तावेज़ या सेट ऑफ़ स्टेप्स (चरणों) का वर्णन है, जिसे यह जाँचने के लिए डिज़ाइन किया जाता है कि सॉफ्टवेयर का कोई विशेष हिस्सा (फीचर या फंक्शन) सही तरीके से काम कर रहा है या नहीं।



Verification vs Validation



टेस्ट केस के मुख्य घटक (Components):

1. Test Case ID:

- Every test case is given a unique number or name.
- हर टेस्ट केस के लिए एक यूनिक नंबर या नाम दिया जाता है।

2. Test Case Name:

- It tells what purpose this test case is for.
- यह बताता है कि यह टेस्ट केस किस उद्देश्य के लिए है।

3. Objective:

- It explains why this test case is created.
- यह बताता है कि इस टेस्ट केस को क्यों बनाया गया है।



4. Precondition:

- What are the conditions that must be fulfilled before starting the test
- टेस्ट शुरू करने से पहले कौन-कौन सी शर्तें पूरी होनी चाहिए।

5. Input Data:

- The data used for the test.
- टेस्ट के लिए उपयोग होने वाला डेटा।

6. Testing Steps (Steps to Execute):

- The steps that are to be followed.
- वह चरण जिनका पालन करना है।

7. Expected Output:

- What will be the result if the software works correctly.
- यदि सॉफ्टवेयर सही तरीके से काम करता है, तो क्या परिणाम मिलेगा।



8. Actual Result:

- What the actual result was after the test run.
- टेस्ट रन के बाद वास्तव में क्या परिणाम आया।

9. स्टेटस (Status):

- पास (Pass) या फेल (Fail)।



Example: Testing the login form.

1. Test Case ID:

- TC001

2. Test Case Name:

- User Login Test

3. Objective:

- To verify that login is happening with correct username and password.

4. Pre-requisite:

- Web browser installed.
- Internet connection.



5. Input Data:

- Username: admin
- Password: 1234

6. Steps:

- Open the software.
- Go to login page.
- Enter username and password.
- Click on login button.

7. Expected Result:

- Reached dashboard.

8. Actual Result:

- Reached dashboard.

9. Status: Pass.

Chapter No-6(Software Testing)

Q 1: Define software Testing. Explain various software Techniques in detail.

Q 1: सॉफ्टवेयर testing को परिभाषित करें। विभिन्न सॉफ्टवेयर तकनीकों को विस्तार से समझाएँ।

Q 2: Differentiate between Black-Box and White-Box Testing.

Q 2: ब्लैक-बॉक्स और व्हाइट-बॉक्स परीक्षण के बीच अंतर बताएं।

Q 3: Differentiate between verification and validation.

Q 3: verification और validation के बीच अंतर बताइए।

Q 4: What is unit testing? Why is it important?

Q 4: यूनिट परीक्षण क्या है? यह क्यों महत्वपूर्ण है?

Q 5: Explain integration testing in detail.

Q 5: integration परीक्षण को विस्तार से समझाइये।

Thank
you!

6-unit
complete