Unit - 2

Software Design :-

Software Design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

It deals with representing the client requirement as described in SRS (Software Requirement Specification) document, into a form that is easily implentable using programming language.

The software design is the first step in SDLC, which moves the concentration from the problem domain to solution domain.

Software Design levels :-

i) External design :- Focus on deciding which modules are needed, their specification, relationship and interface among them.

ii) Internal design :- Focus on planning and specifying the internal structure and processing detail.

Software Duign Document (SDD)

→ Conceptual Design (What?)

→ Techaical Design (How?)

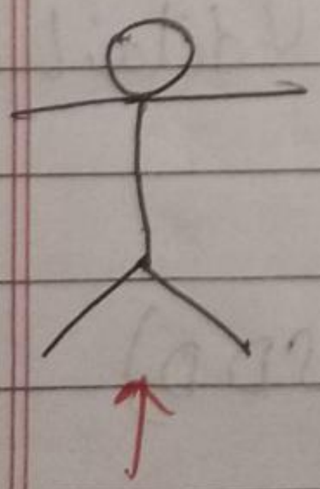
Designer

Conceptual Design

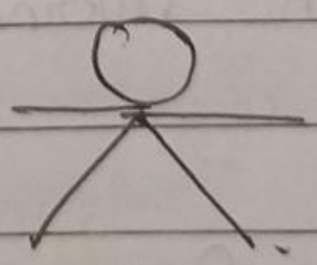
- → Customer Understandable Language
- → Detailed System characteristics explanation
- → System functionality described
- → Independent of Implementation
- → Linked to SRS.

Technical Design

- → H/W Component & Design
- → Functionality & hierarchy of S/W component
- → S/W Architecture
- → H/W
- → Data Structure & flow of data
- → I/O component of the system.



Customer



System Designers

1. What are the objective of a software design?

Ans→ Following are the objective of software design:

i) Correctness :- Software design should be correct as per requirement.

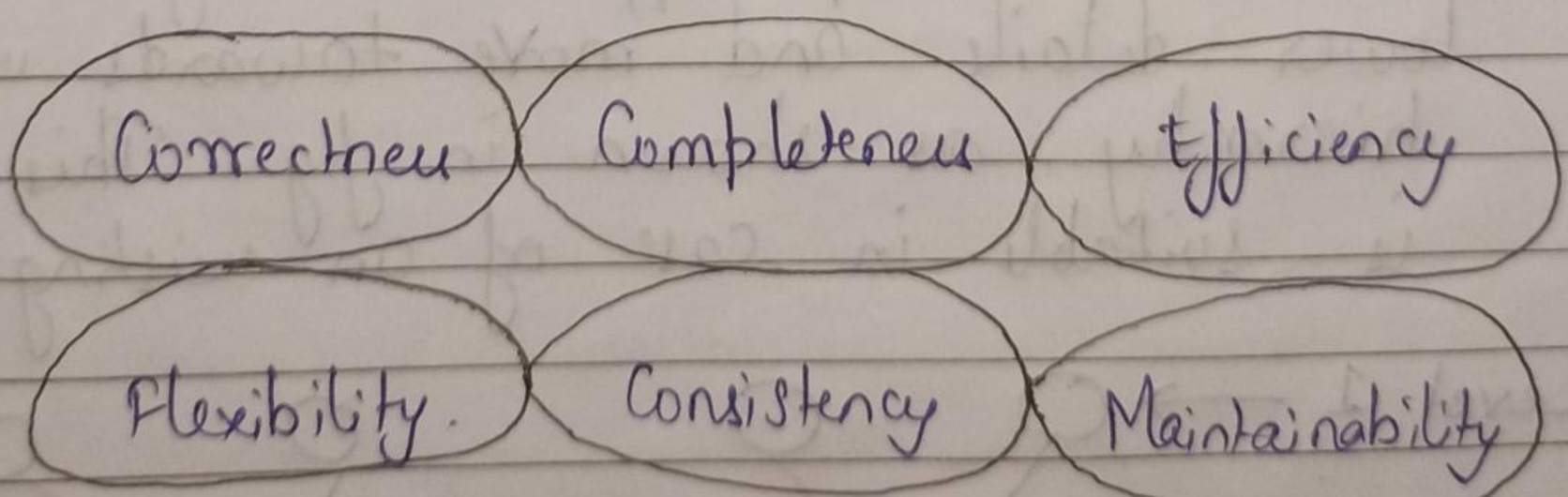
ii) Completeness :- The design should have all components like data structures, modules and external interfaces.

iii) Efficiency :- Resources should be used efficiently by the program.

iv) Flexibility :- Resources or Software design should be able to modify on changing needs.

v) Consistency :- There should not be any inconsistency in the design.

vi) Maintainability :- The design should be so simple so that it can be easily maintainable by other designers.



objective of Software Design.

Software Design Principles :-

i) Problem partitioning :-

It means dividing the problem into smaller pieces so that each piece can be captured separately. For software design, problem partitioning goal is to divide the problem into manageable pieces.

Benifits :-

→ Software is easy to understand
→ Software become simple
→ Software is easy to modify
→ Software is easy to maintain
→ Software is easy to expand & test.

ii) Abstraction :- Abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation.

It can be used for existing element as well as component being designed.

Two Abstraction Mechanism :-

(a) Functional abstraction :- In functional abstraction, A module is specified by method it performs Details of algorithm to accomplish the function is not visible to the user of the function. It forms basis for Functional oriented design approach.

(b) Data Abstraction :- In this, details of the data element is not visible to the user. It forms the basis for Object-oriented design approach.

(iii) Modularity :-

It is a process of dividing a software system into multiple independent modules where each module works independently.

It is the only property that allows a program to be intellectually manageable.

Desirable property of modular system :-

* Each module is a well defined system that can be used with other application.
* Each module has a single specified objectives.
* Modules can be separately compiled and saved in library.
* Module should be easier to use than to build.
* Modules are simpler from outside than inside.

Disadvantages :-

there are not certainly

* Execution time is maybe longer.
* Storarage size perhaps increased.
* Compilation & loading time may be longer.
* Inter-module communication problem may be increased.
* More linkage required, run-time may be longer and more documentation has to be done.

3. List the advantages of Modularization.

Ans→ Following are the advantages of modularization :-

i) It allows large programs to be written by several or different people.
ii) It encourages the creation of commonly used routines to be placed in the library and used by other programs.
iii) It simplifies the overlay procedure of loading a large program into main storage.
iv) It provides more checkpoints to measure progress.
v) It provides a framework for complete testing more accessible to test.

4. Write short note on Cohesion & Coupling.

Ans→ Cohesion :- It is the measure of degree of to which the element of module are functionally related. It is also the degree to which all elements are directed towards performing the single task.

Following are the various types of cohesion :-

i) Functional :- Performs the task and function

ii) Sequential :- Only take care of the sequence of task being performed i.e. input of one module will be output of next upcoming module.

iii) Communicational :- Two or more elements operates on same input data or contribute towards same output.

iv) Procedural :- It deals with whether or not procedures is being followed.

v) Temporal :- It deals with whether or not time stamps is being followed.

vi) Logical :- If all the element of module performs similar operation, then module is said to be logically cohesive.

vii) Coincidental :- A module is coincidental cohesive, if it peform task that are associated with each other very loosely.

Coupling :- It is the measure of the degree of interdependence between the module. A good software will have low coupling.

Following are the types of coupling :-

i) Data Coupling :- If the dependency between the module is based on the fact that they communicate by passing data only, then the modules are said to be data coupled.

ii) Stamp Coupling :- In this, complete data structure is passed from one module to another.

iii) Control Coupling :- If the module communicate by passing control information then they are controlled coupled.

iv) External Coupling :- Here, module depends on other modules that are external to the software being developed or to the particular type of hardware.

Date
Page

v) Common :- Here, the modules have shared some common parameter or data or data structure which is global in nature.

vi) Content Coupling :- One module can modify the data of another module or control flow is passed from one module to the other.

~~Strategy~~ Modular Design :-
Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system.

Different sections :-

(a) Functional Independence :- It is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. Independent module is easier to maintain, test and reduce error propagation and can be reused in other programs as well.

(b) Information hiding :- Fundamental of Information hiding suggest that modules can be characterized by the design decision that protect from the others. i.e in other words, module should be specified that data included in module is inaccessable to other modules.

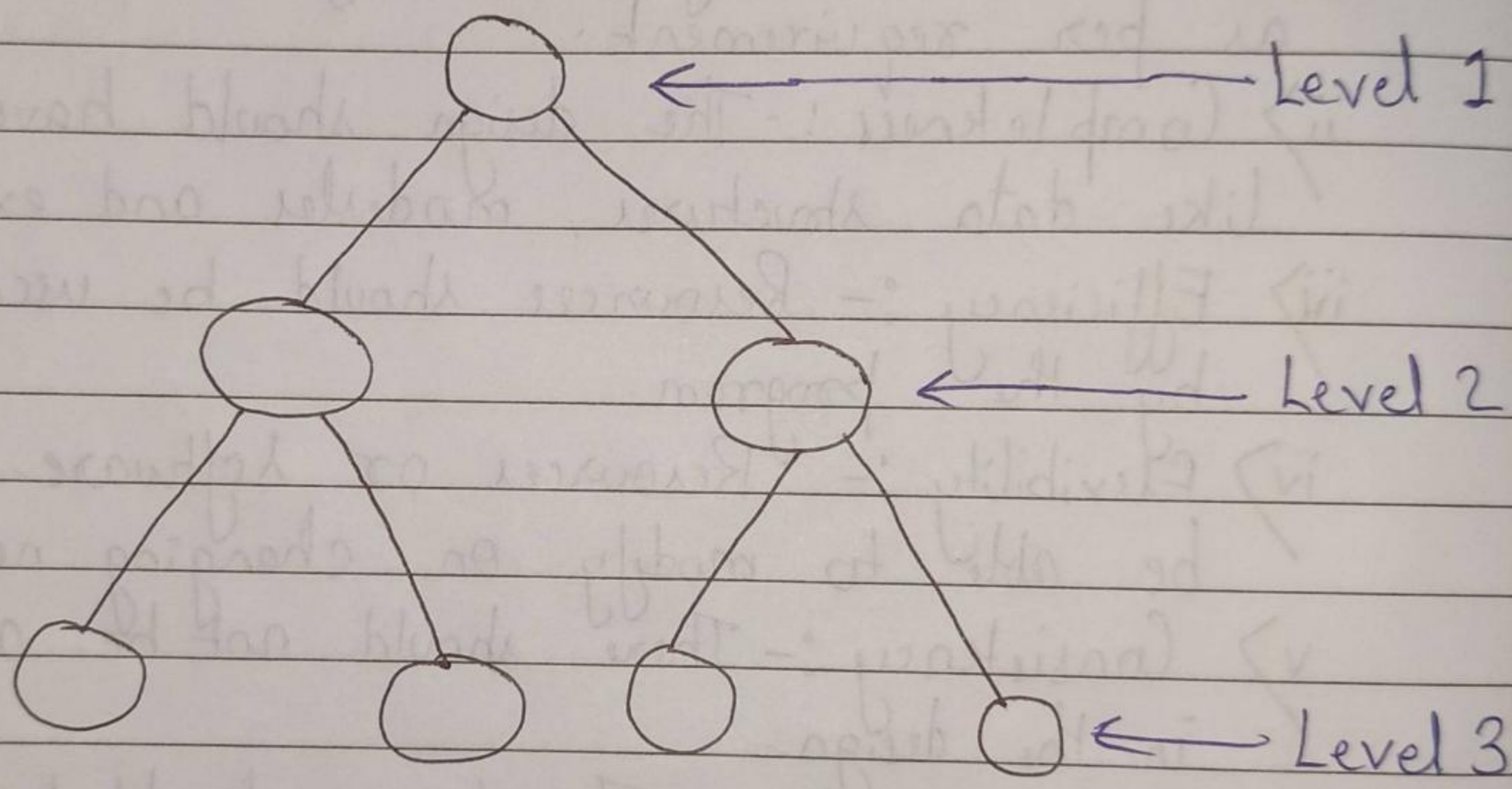
(iv) Strategy of design :- A good system design strategy is to organise the program modules in such a method that are easy to develop and later too change.

To design a system there are two possible approach :-
(a) Top down
(b) bottom up

2. Describe top down and bottom up strategy.

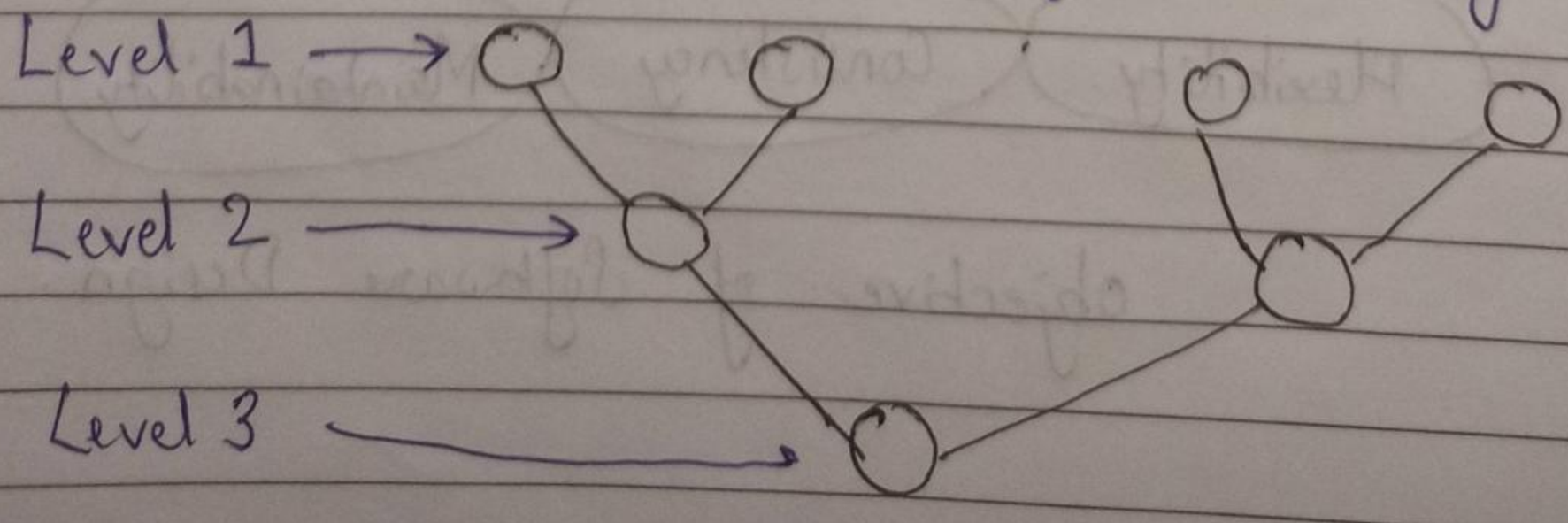
Ans→ Top-down Approach :-

This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



Bottom-up Approach :-

A bottom-up approach begins with the lower details and move towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



Difference between Coupling & Cohesion :-

Coupling	Cohesion
→ Also known as inter-module binding	→ Also known as intra-module binding.
→ It shows relationship between modules	→ It shows relationship within modules.
→ It shows relative independence between the module.	→ It shows module's relative functional strength.
→ While creating we should aim for low coupling.	→ While creating we should aim for high cohesion.
→ In this, modules are linked to other modules	→ In this modules focuses on single thing.

* Difference between top-down Approach & Bottom-up approach :-

Top down design Approach	Bottom up design Approach
→ In this, problem is divided into small part.	→ In this, small solutions are integrated to make a whole solution
→ It is mainly used by structured programming like C, COBOL, FORTAIN, etc.	→ It is mainly used by object oriented programming like C++, Java, etc.

→ Often Redundancy can be encounted due to breaking of task into smaller pieces → It is used in debugging module documentation → In this, decomposition is done.	→ Redundancy is minimized by using encapsulation & data-hiding, Abstraction, etc. → It is used in testing → In this composition is done.

* Design Model - Architecture Design :-

A design model in software development is an object-based picture or pictures that represent the use cases for a system.

The software needs architecture design to represent the design of software. IEEE define it as "the process of defining a collection of hardware and software component and their interfaces to establish the framework for the development of a computer system".

The software that is built for computer-based systems can exhibits one of the many architecture styles. Each style will describe a system category that consist of :-

(a) Set of components (i.e. database, modules) that will perform a function required by system

(b) Set of connectors that help in coordination, communication and cooperation between components.

(c) Conditions that how components can be integrated to form a system.

Components :-

A component is a modular, portable, replaceable and reusable set of well-defined functionality that encapsulate its implentation and exporting it as a high level interface.

A software component can be defined as a unit of composition with a contractually specified interface and explicit, context dependencies only (i.e. software component can be deployed independently and is subject to composition by third parties.

Characteristics of Components :-

* Reusability :- Components are usually designed to be reused in different situations in different applications.
* Replaceable :- Components may be freely substituted with other similar components.
* Independent :- Designed to have minimal dependencies on other components.
* Encapsulated :- A component defines the interface, which allow the caller to use its functionality and do not expose details of internal process or any internal variable.
* Extensible :- A components can be extented from existing components to provide new behavior.

5. Discuss Component Level design.

Ans → Component based architecture design focuses on the decomposition of the design into individual functional or logical components that represents well defined communication interface containing methods, events and properties.

It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions.

The primary objective of component level design is to ensure the component reusability. A component encapsulates functionalities and behaviour of a software element into a reuseable and self-deployable binary unit.

Component-oriented software design has many

Date
Page

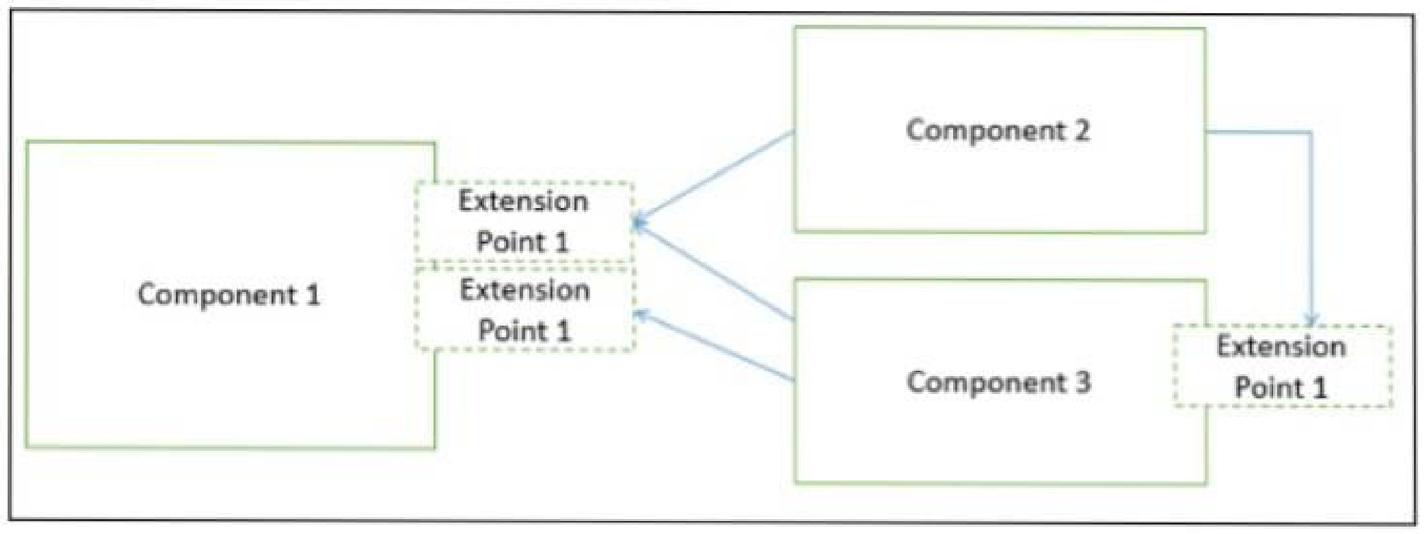
advantages over the traditional object-oriented approaches such as :-

- Reduced time in market and development cost by reusing existing components.
- Increased reliability with the reuse of existing components.

- **Principles of Component–Based Design**

A component-level design can be represented by using some intermediary representation (e.g. graphical, tabular, or text-based) that can be translated into source code. The design of data structures, interfaces, and algorithms should conform to well-established guidelines to help us avoid the introduction of errors.

- The software system is decomposed into reusable, cohesive, and encapsulated component units.
- Each component has its own interface that specifies required ports and provided ports; each component hides its detailed implementation.
- A component should be extended without the need to make internal code or design modifications to the existing parts of the component.
- Depend on abstractions component do not depend on other concrete components, which increase difficulty in expendability.
- Connectors connected components, specifying and ruling the interaction among components. The interaction type is specified by the interfaces of the components.
- Components interaction can take the form of method invocations, asynchronous invocations, broadcasting, message driven interactions, data stream communications, and other protocol specific interactions.
- For a server class, specialized interfaces should be created to serve major categories of clients. Only those operations that are relevant to a particular category of clients should be specified in the interface.
- A component can extend to other components and still offer its own extension points. It is the concept of plug-in based architecture. This allows a plugin to offer another plugin API.



- **Component-Level Design Guidelines**

Creates a naming convention for components that are specified as part of the architectural model and then refines or elaborates as part of the component-level model.

- Attains architectural component names from the problem domain and ensures that they have meaning to all stakeholders who view the architectural model.
- Extracts the business process entities that can exist independently without any associated dependency on other entities.
- Recognizes and discover these independent entities as new components.
- Uses infrastructure component names that reflect their implementation-specific meaning.
- Models any dependencies from left to right and inheritance from top (base class) to bottom (derived classes).
- Model any component dependencies as interfaces rather than representing them as a direct component-to-component dependency.

- **Conducting Component-Level Design**

Recognizes all design classes that correspond to the problem domain as defined in the analysis model and architectural model.

- o Recognizes all design classes that correspond to the infrastructure domain.
- o Describes all design classes that are not acquired as reusable components, and specifies message details.
- o Identifies appropriate interfaces for each component and elaborates attributes and defines data types and data structures required to implement them.
- o Describes processing flow within each operation in detail by means of pseudo code or UML activity diagrams.

- o Describes persistent data sources (databases and files) and identifies the classes required to manage them.
- o Develop and elaborates behavioral representations for a class or component. This can be done by elaborating the UML state diagrams created for the analysis model and by examining all use cases that are relevant to the design class.
- o Elaborates deployment diagrams to provide additional implementation detail.
- o Demonstrates the location of key packages or classes of components in a system by using class instances and designating specific hardware and operating system environment.
- o The final decision can be made by using established design principles and guidelines. Experienced designers consider all (or most) of the alternative design solutions before settling on the final design model.

- **Advantages**

- o **Ease of deployment** – as new compatible versions become available, it is easier to replace existing versions with no impact on the other components or the system as a whole.
- o **Reduced cost** – the use of third-party components allows you to spread the cost of development and maintenance.
- o **Ease of development** – Components implement well-known interfaces to provide defined functionality, allowing development without impacting other parts of the system.
- o **Reusable** – the use of reusable components means that they can be used to spread the development and maintenance cost across several applications or systems.
- o **Modification of technical complexity** – A component modifies the complexity through the use of a component container and its services.
- o **Reliability** – the overall system reliability increases since the reliability of each individual component enhances the reliability of the whole system via reuse.
- o **System maintenance and evolution** – Easy to change and update the implementation without affecting the rest of the system.
- o **Independent** – Independency and flexible connectivity of components. Independent development of components by different group in parallel. Productivity for the software development and future software development.

9. Pattern oriented design

Design patterns are used to represent some of the best practices adapted by experienced object-oriented software developers. A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples.

Usage of Design Pattern

Design Patterns have two main usages in software development.

- **Common platform for developers**

  Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

- **Best Practices**

  Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps inexperienced developers to learn software design in an easy and faster way.

- **Types of Design Patterns**

  As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software**, there are 23 design patterns which can be classified in three categories: Creational, Structural and Behavioral patterns. We'll also discuss another category of design pattern: J2EE design patterns.

S.N.	Pattern & Description
1	**Creational Patterns** These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.
2	**Structural Patterns** These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.
3	**Behavioral Patterns** These design patterns are specifically concerned with communication between objects.

4	**J2EE Patterns** These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

User Interface Design :-

It is the front-end application view to which user interacte in order to use the software.

The software becomes more popular if its user interface is

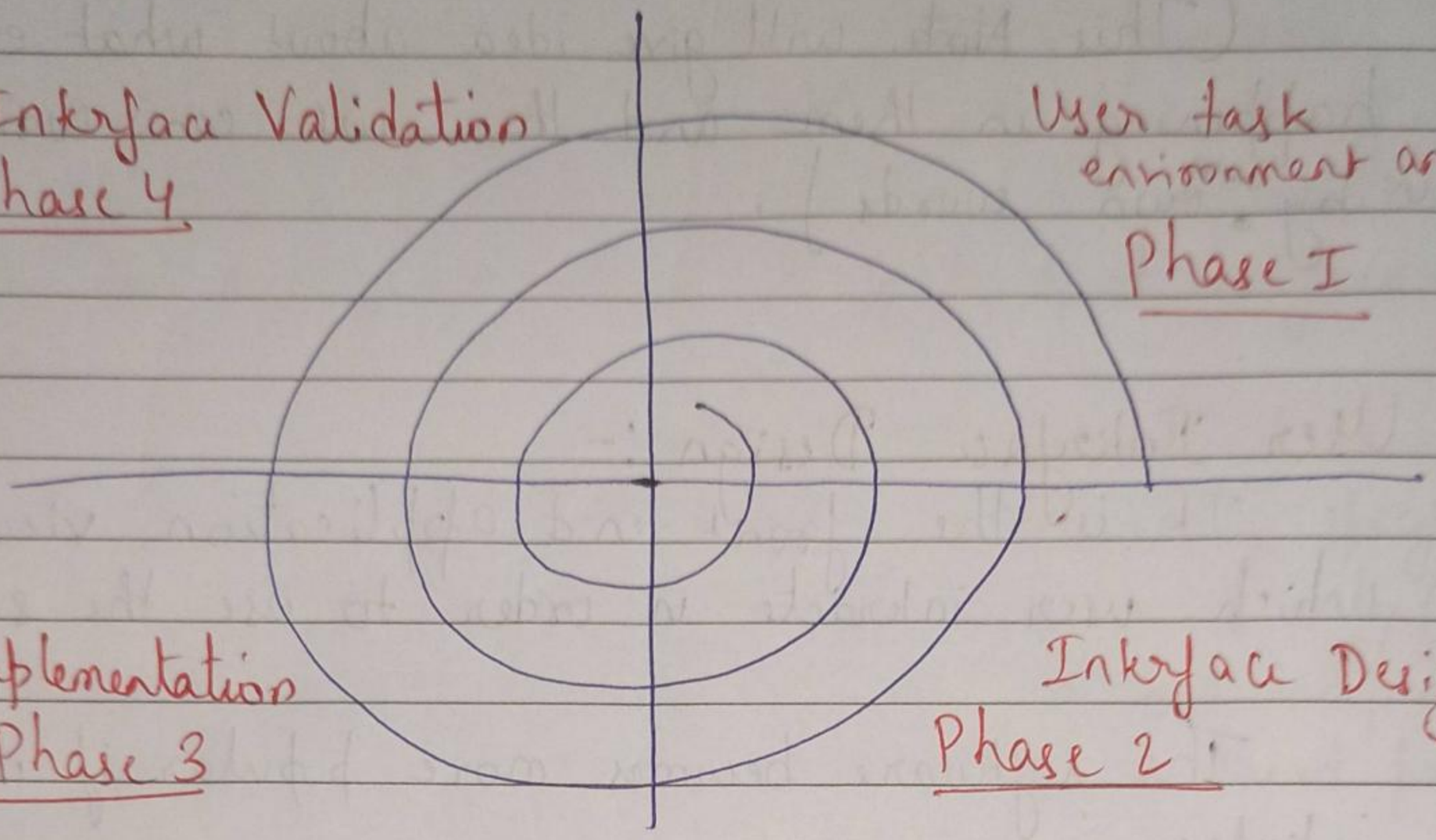
- Attractive
- Simple to use
- Responsive in short-time
- Clear to understand
- Consistent on all interface screens.

Types :-

i) Command Line Interface → It provides Command prompt, where the user types the command and feed to system. The user need to remember syntax of command and its use

ii) Graphical User Interface → GUI provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprete the software.

User Interface Design Process :-



We can explain it similar to SDLC

Design Reuse :-

Design Reuse is the process of building new software application and tools by reusing previously developed designs. New features and functionalities may be added by incorporating minor change.

It involves the use of designed modules such as logic and data to build a new and improved product. The reusable components are less expensive. This avoid reinventing existing software.

Data reuse process is as follows :-

(a) Gathering Information → This involves the collection of information, processing and modeling of fetch ~~do~~ related data.

(b) Information reuse :- This involves effective use of data.

* Design reuse process has following challenges/ issues :-

(a) Retrieval of Design :- availability

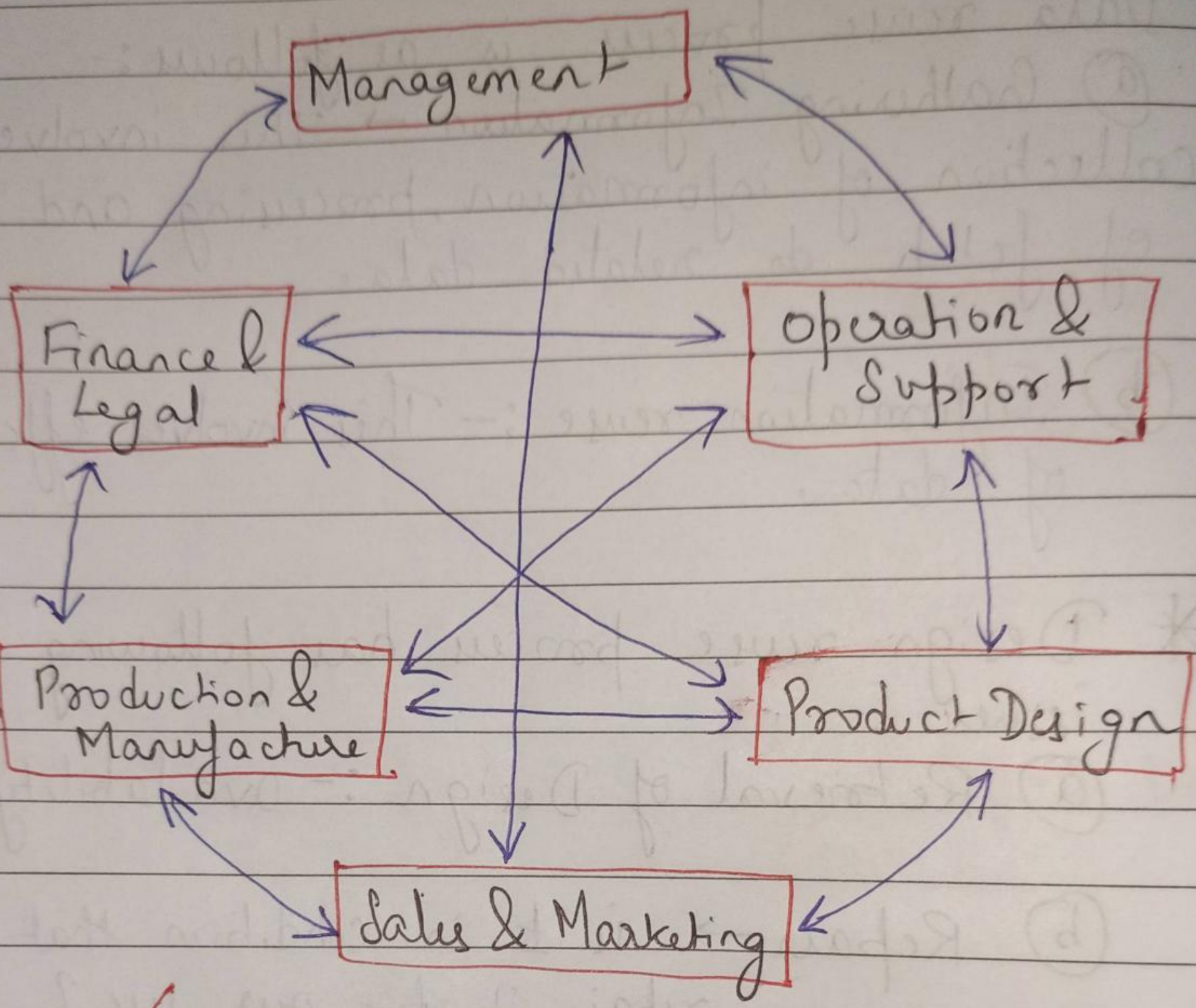
(b) Repair → is it in condition that we can repair it for our use?

(c) Reuse → Can we use it for our system?

Concurrent Engineering :-

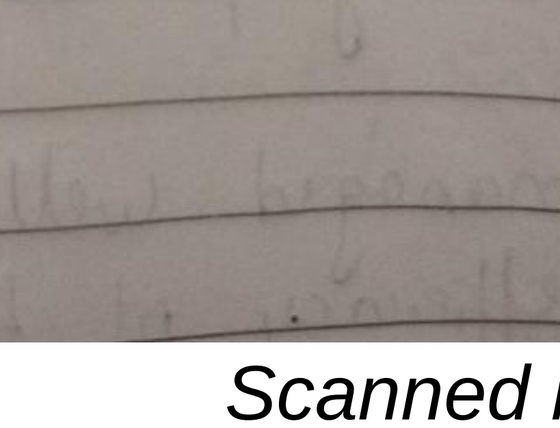
Concurrent Engineering is a method of designing and developing engineering products, in which different department work on different stage of ~~defe~~ engineering product development simultaneously.

If managed well, it helps to ~~manage~~ increase the efficiency of product development and marketing considerably reducing the time and contributing to the reduction of overall development cost while improving final product quality.



↳ Concurrent Engineering interactions

Elements :-
- People
- Process
- Technology



- **People**

Concurrent product development is a multidisciplinary team task and it's necessary that companies utilize the right skilled personnel at the right time to accelerate product development. It is also necessary to find people with the right skills and experience along with the following key aspects;

  - Multidisciplinary team to suit the product at the start of the NPD
  - Teamwork culture at the core of the program

Good communication and collaboration between teams – sharing relevant and up to date information across departments and personnel

The harmonized goal across the company from the top management to the bottom of the organizational structure

- **Process**

A process is a series of product development steps that need to happen to achieve a goal. These can be project planning stages, milestone management, problem-solving methodologies, product development key stages, information sharing workflow, etc., as just people are ineffective without processes in place to support their tasks and decisions. Following are some of the processes that can be adopted in concurrent engineering;

  - Project planning processes and workflow management which include key new product development elements such as key design stages, milestones for cross-departmental interaction, etc.
  - Workflow for product data management such as sharing information, managing engineering change, control specification creep, etc.
  - Product requirement tracking and checkpoints using techniques such as Quality Function Deployment (QFD) across departments
  - Design evaluation workflow processes
  - Design analysis methodologies such as brainstorming
  - Failure Mode and Effects Analysis (FMEA) allows for a systematic investigation of the occurrence and impact of possible flaws in the new product design
  - The use of Design of Experiments (DOE) enables the systematic identification of critical product/process parameters that influence performance

- **Technology**

For concurrent engineering to be successful, the effective introduction of tools, techniques, and technologies to aid a smooth integration of people and processes is vital. Following key aspects should be considered before any implementation.

  - Identifying the correct tools and technologies that suit the company size, number of team members, processes implemented and product type
  - Identifying the training needs and training people to use the tools and technologies identified above

These are just a few of the many supportive tools that can be used in a concurrent engineering environment.

  - Project management software
  - Product data management & product lifecycle management suites
  - Quality Function Deployment (QFD)
  - 3D CAD and rapid prototyping technologies such as additive manufacturing
  - Suitable FEA tools
  - Evaluation tools such as DFM, DFA, DFMA and DOE
  - Failure mode analysis tools such as FMEA