Design:

S/w Design:

Is the process to transform the user requirement into some suitable form. Which helps the programmer in s/w coding and implementation.

->During the s/w design phase the design document is produced, based on the customer requirement as documented in the srs (software requirement specification) document.

->hence the aim of this phase is to transform the srs document into design document.

The following item are designed and documented during the design phase:

1)Different modules required

2)control relationship among the modules

3)interface among different modules.

4)Data structure among the different modules

5)Algorithms required to implemented among the individual modules.

Objectives of S/W Design:

1)correctness:

A Good Design should be correct. It should implement all the functionality of the system.

2)Efficiency:

A Good Desgin should address the resource , time and cost and optimization issues.

3) Understandability:

It should be easily understandable for which it should be modular and all the module are arranged in the layer.

4)Completeness:

It should have all the components like data structure , modules and interface.

5)Maintainability:

It should be easily amenable to change whenever a change request is made from the customer side.

SOFTWARE DESIGN CONCEPTS:

Concepts are defined as principal idea or invention that comes in our mind or in thought to understand something.

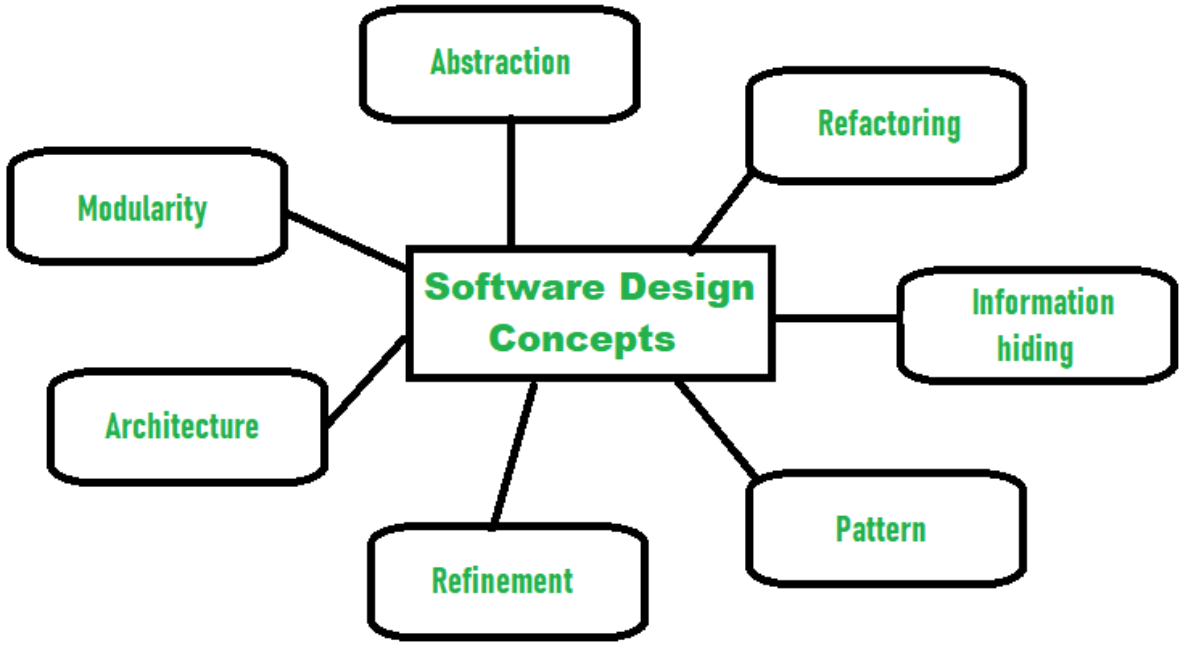
The s/w design concepts simply means the idea or principle behind the design.

->it describes how you plan to solve the problem of designing s/w, the logic, or thinking behind what will to design s/w

->it allows the s/w engineer to create the model of the s/w or system or product that is to be developed or build.

->the s/w design concepts provides a supporting and essential structure or model for developing the right s/w.

They are many concepts of s/w design some of them are



1)Abstraction: (hide relevant data)

Simply means to hide the details to reduce complexity and increase efficiency or quality.

i)Procedural abstraction:

A name sequence of instruction that has a specific and limited function:

Eg: word open for Door.

ii)Data abstraction:

A name collection of data that describe the data object.

Data abstraction for door would be set of attribute that describe the door.

Eg: door type, swing direction, weight.

->the lower level of abstraction provides a more details description of the solution.

2) Modularity: (subdivide the system)

->it divide the system or project into smaller part to reduce the complexity of the system or project.

->so that these parts created independently and then use these part in different function.

->it is necessary to divide the s/w into components known as modules because

nowdays there are different s/w available like monolithic s/w that is hard to grasp for s/w engineer.

->so modularity is design has now become a trend and important.

3)Architecture: (design structure of something)

->is concept that focus on various elements and the data of the structure.

->the components interact with each other and use the data of the structure in architecture.

4)Refinement : (remove the impurities)

->to refine something to remove any impurities if present and increase the quality.

->is concepts of s/w design is actually a process of developing or presenting the s/w or system in detailed manner that means to elaborate a system or s/w.

->its very necessary to find out any error if present and then reduce it,

5)Pattern: (repeated form)

-> it means repeated form or design in which the same shape is repeated several time to form pattern.

->the pattern in the design process means repetition of solution to common recurring problem with a certain context.

6)Information Hiding: (hiding the information)

-> to hide the information so that it cannot be accesses by an unwanted party

->in s/w design, the information hiding is achieved by designing the modules in manner that the information gathered or contained in one module is hidden and it cannot be accessed by any other modules.

7)Refactoring: (reconstruct something)

->is means to reconstruct something in such way that it does not affect the behaviour or any other feature.

->to reconstruct the design to reduce and complexity and simply it without affecting the behaviour or its functions.

->”the process of changing a s/w system in way that it wont affect the behaviour of design and improve the structure.

Architectural Design:

->the s/w need the architectural design to represent the design of s/w.

->IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interface to establish the framework for the development of computer system”

-> the s/w that is built for computer based can exhibit one of the many architectural style.

->Each style will describe a system category that consists of:

I) A set of component (eg database , computational model) that will perform a function required by system.

2)A set of connector will help in coordination, communication and cooperation b/n the components.

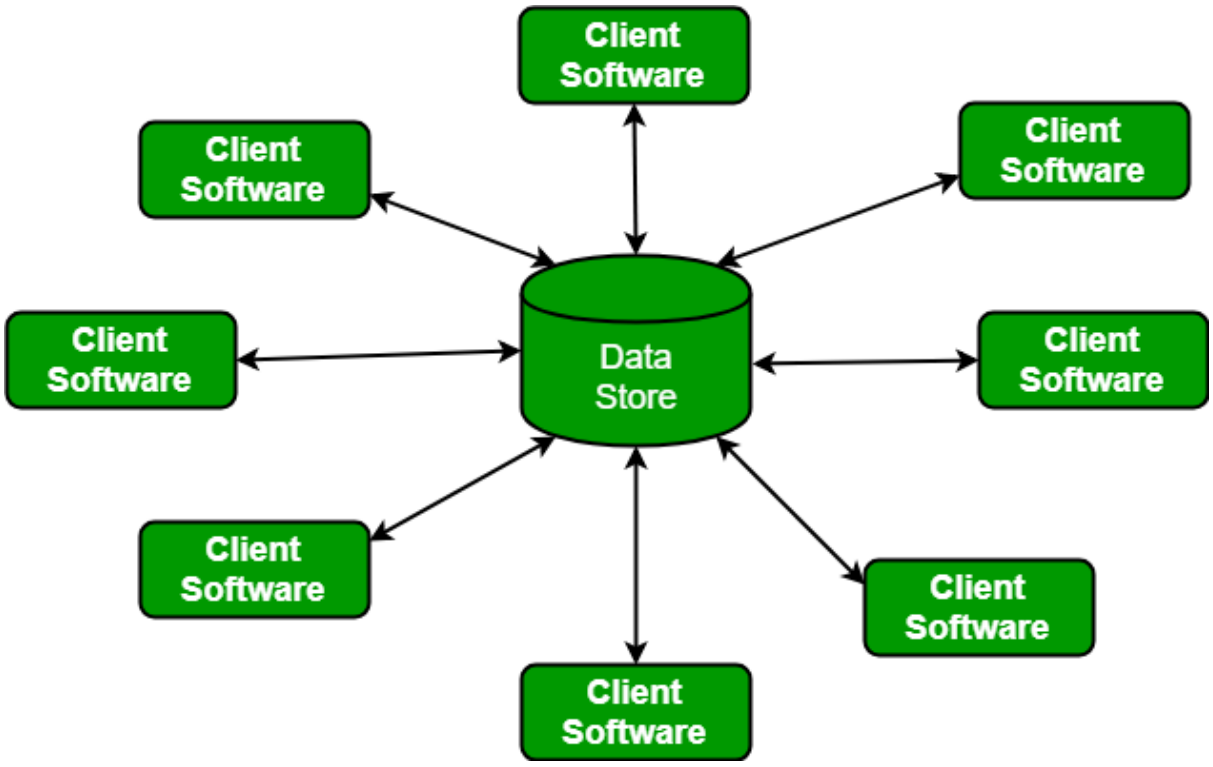
3)conditions that how components can be integrated to form the systems

4)Semantic model that help the designer to understand the overall properties of the system.

5)the use of architecture style is to establish a structure for all the components of systems.

Types of architecture Styles:

1)Data Centred Architecture:

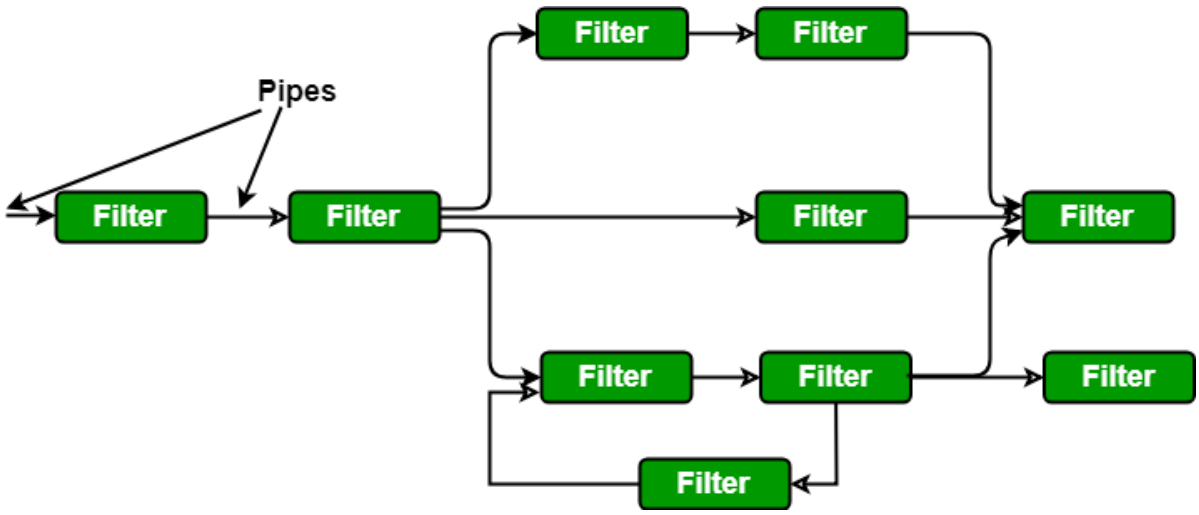


→A data store will reside at the centre of this architecture and is accessed frequently by other components. That update add delete or modify that data present within store.

→when data related to client or data of interest for the client change the notification to client s/w

->this data centred architecture will promote integrability. This means that the existing component can be changed and new client component can be added the architecture without the permission or concern of other client.

2)Data Flow Architecture:



->this kind of architecture is used when input data to be transformed into o/p data through series of computational manipulative components.

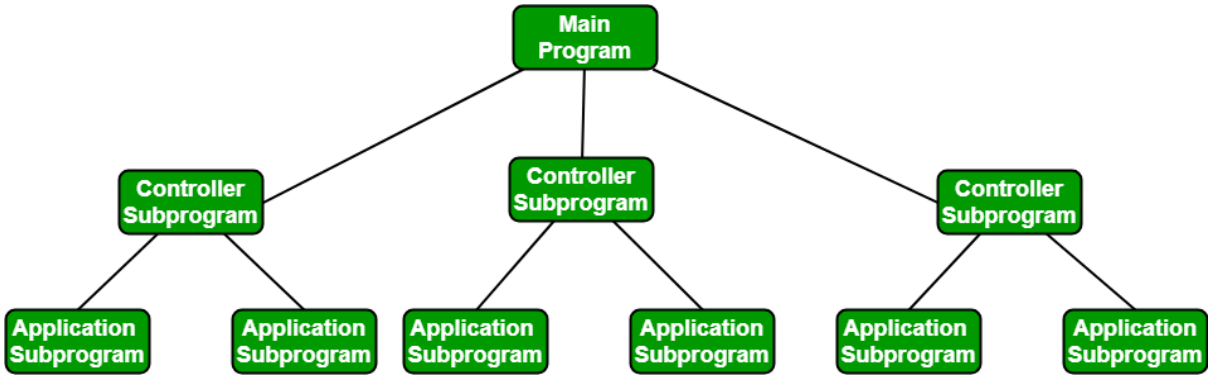
->diagram represent pipe and filter architecture since it uses both pipe and filter and it has a set of components called filters connected by pipes.

->Each filter will work independently and is designed to take data i/p of certain form and produce data o/p to the next filter of specified form. The filter don’t require any knowledge of working of neighboring filter.

->if the data flow degenerates into a single line of transform , then it is termed as batch sequential, the structure accepts the batch of data and then applies a series of sequential component to transform it.

3)call or return architecture:

->used to create a program that is easy to scale and modify. Many sub styles exist within this category.



->MAIN PROGRAM OR SUB PROGRAM:

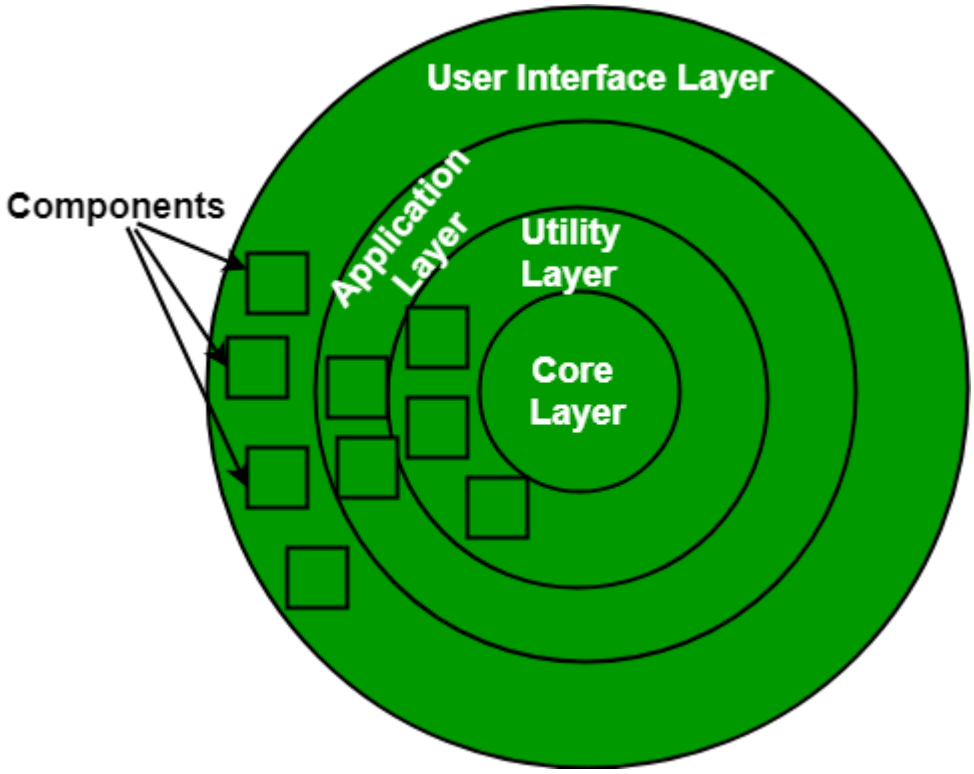
The main program structure decompose into number of subprograms or function into control hierarchy

->main program contains number of subprogram that can invoke other components.

4)Object oriented architecture:

The components of the system encapsulate data and the operation that must be applied to manipulate the data . the coordination and communication are established via the message passing.

5)Layerd architecture:



->A number of different layers are defined with each layer performing well defined set of operation each layer will do some

operation that becomes closer to machine instruction set progressively.

->At the outer layer component will receive the user interface operations and at the inner layer components will perform the operating system interface(communication and coordination with os)

->intermediate layer to utility services and applications s/w functions.

Coupling and Cohesion:

➔ The purpose of design phase in s/w development life cycle is to produce a solution to a problem given in SRS(Software Requirement specification) document.

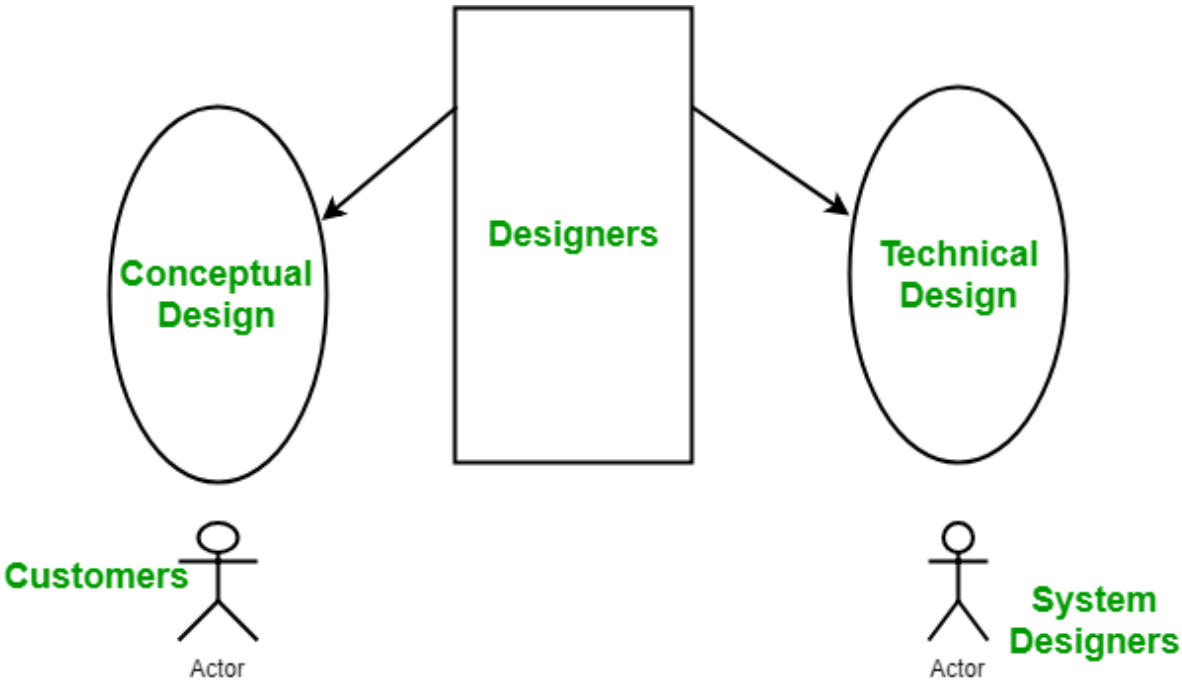
➔ The o/p of design phase is s/w design document(SDD)

Basically design is two part iterative process.

First part is conceptual design that tells the customer what the system will do.

Second part is technical Design that allows the system builder to understand the actual hardware and

s/w needed to solve customer problem.



Conceptual Design of system:

➔ Written in simple language (customer understandable language)

➔ Detail explanation about system characteristics.

➔ Describe the functionality

➔ It is implementation linked with document.

Technical Design of system:

->functionality and hierarchy of s/w component.

->S/w architecture

->network architecture

->data structure and flow of data.

->show interface

Modularization:

The process of dividing a s/w system into multiple independent modules where each modules work independently.

Advantage of modularization:

->easy to understand the system

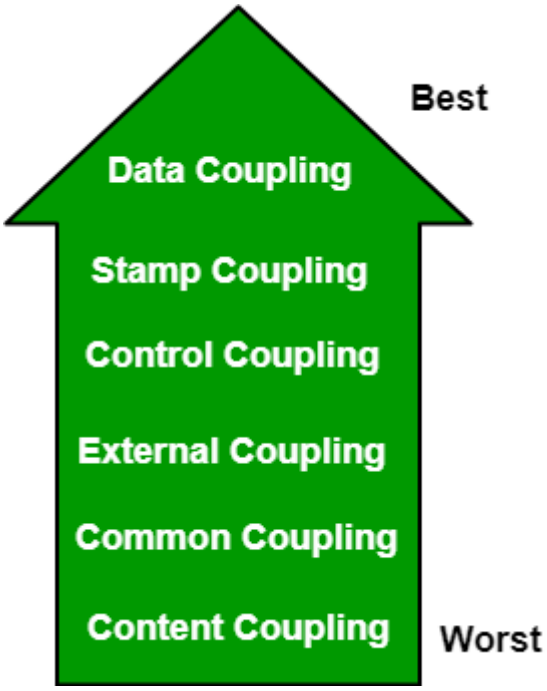
->system maintenance is easy

-> modules can be used many time as the requirement. No need to write it again and again.

Coupling: (relationship between module)

Coupling is measure of the degree of interdependence (connection) b/n the modules. A good s/w will have low coupling.

Types coupling:



1)data coupling:

If the dependency b/n two models is based on the fact that they communicate by passing only data, then the model are said to data coupled.

->in data coupling the component are independent to each other and communicating through the data module communication don’t tramp data(data which is passed via one function to another function and not otherwise used by first)
EX: customer billing system

2)Stamp Coupling:

The complete data structure is passed from one module to another module therefore it involve tramp data. It may be

necessary due to efficiency factor the choice made by insightful designer.

3)control coupling:

If the module communicate by passing control information then they are said to be control coupled. It can be bad if parameter indicate completely different behaviour and good if parameter allow factoring and reuse functionality.

->External Coupling:

The modules depend on other module external to s/w beign developed or particular type of software,

->common coupling:

The modules have shared data such global data structure. The changes in global data MEAN TRACING BACK TO ALL MODULES

WHICH access that data to evaluate the effect of the changes so it has got disadvantage like difficulty in reuse model

->Content coupling:

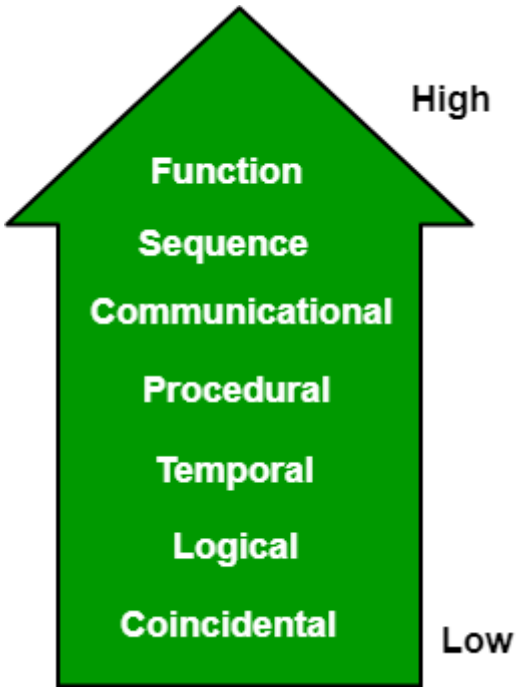
One module can modify the data of another module or control flow is passed from one module to the other . the worst form of coupling and should be avoided.

Cohesion : (relationship within module)

Is defined to the degree to which the elements of the module are functionally related .

- ➔ It measure the strength of relationship b/n pieces of functionality within given module.
- ➔ A good s/w design will have high cohesion.

Types of Cohesion : (relationship within module)



1)Function cohesion:

Is said to exist if the different elements of module , cooperate to achieve single function.

2)Sequential cohesion:

Is said to possess sequential cohesion if the element of module from the component of the sequence where the

output from one component of sequence is input to the next.

3)communication cohesion:

Is said to have communication cohesion, two element operate on the same i/p data or contribute toward the same o/p data.

EX:update record into database and sent it to printer.

4)Procedural cohesion: element of procedural cohesion ensure that order execution action are still weakly connected and unlikely to be reusable.

EX:calculate the student gpa, print student record.

5)Temporal cohesion:

The element are related by their timing involved . a model connected with

temporal cohesion all the task must be executed in same time span. This cohesion contains the code for initializing all the part of the system

6) Logical cohesion:

The element are logically related and not functionally.

Ex: all the component read input from tape disk and network. All the code for the functions is in the same component operations are related by the function are significantly different.

8)Coincidental cohesion:

The element are not related . the element have no conceptual relationship other than location in source code.

It is accidental and worst form of cohesion.

Ex: Print next line and reverse the character of a string in single components.

User Interface:

Is the front end application view to which user interact in order to use the s/w . the s/w becomes more popular if its user interface is attractive, simple to use, responsive in short time, clear to understand.

Two types of user interface:

1)command line interface:

It provides an command prompt, where the user types the command and feed to the system. The user need to remember the syntax of the command and its use.

2)Graphical user interface:

It provide the simple interactive interface to interact with system. GUI can be combination of both hardware and software. Using GuI user can interprets the s/w.

The golden rules of user interface design:

1)place the user in control.

2)reduce the users memory load

3)make the interface consistent.

Object Oriented Analysis and Design

Object-Oriented Analysis and Design (OOAD) is a software engineering methodology that involves using object-oriented concepts to design and implement software systems. OOAD involves a number of techniques and practices, including object-oriented programming, design patterns, UML diagrams, and use cases. Here are some important aspects of OOAD:

1. Object-Oriented Programming: Object-oriented programming involves modeling real-world objects as software objects, with properties and methods that represent the behavior of those objects. OOAD uses this approach to design and implement software systems.
2. Design Patterns: Design patterns are reusable solutions to common problems in software design. OOAD uses design patterns to help developers create more maintainable and efficient software systems.

3. UML Diagrams: Unified Modeling Language (UML) is a standardized notation for creating diagrams that represent different aspects of a software system. OOAD uses UML diagrams to represent the different components and interactions of a software system.
4. Use Cases: Use cases are a way of describing the different ways in which users interact with a software system. OOAD uses use cases to help developers understand the requirements of a system and to design software systems that meet those requirements.

There are several advantages to using OOAD in software engineering:

1. Reusability: OOAD emphasizes the use of reusable components and design patterns, which can save time and effort in software development.
2. Scalability: OOAD can help developers design software systems that are scalable and can

handle changes in user demand and business requirements over time.

3. Maintainability: OOAD emphasizes modular design and can help developers create software systems that are easier to maintain and update over time.
4. Flexibility: OOAD can help developers design software systems that are flexible and can adapt to changing business requirements over time.
5. However, there are also some potential disadvantages to using OOAD:
6. Complexity: OOAD can be complex and may require significant expertise to implement effectively.
7. Time-consuming: OOAD can be a time-consuming process that involves significant upfront planning and documentation.
8. Rigidity: Once a software system has been designed using OOAD, it can be difficult to

make changes without significant time and expense.

9. Cost: OOAD can be more expensive than other software engineering methodologies due to the upfront planning and documentation required.
10. Overall, OOAD can be an effective approach to designing and implementing software systems, particularly for complex or large-scale projects. However, it's important to weigh the advantages and disadvantages carefully before adopting this approach.

Object-Oriented Analysis (OOA) is the first technical activity performed as part of object-oriented software engineering. OOA introduces new concepts to investigate a problem. It is based on a set of basic principles, which are as follows-

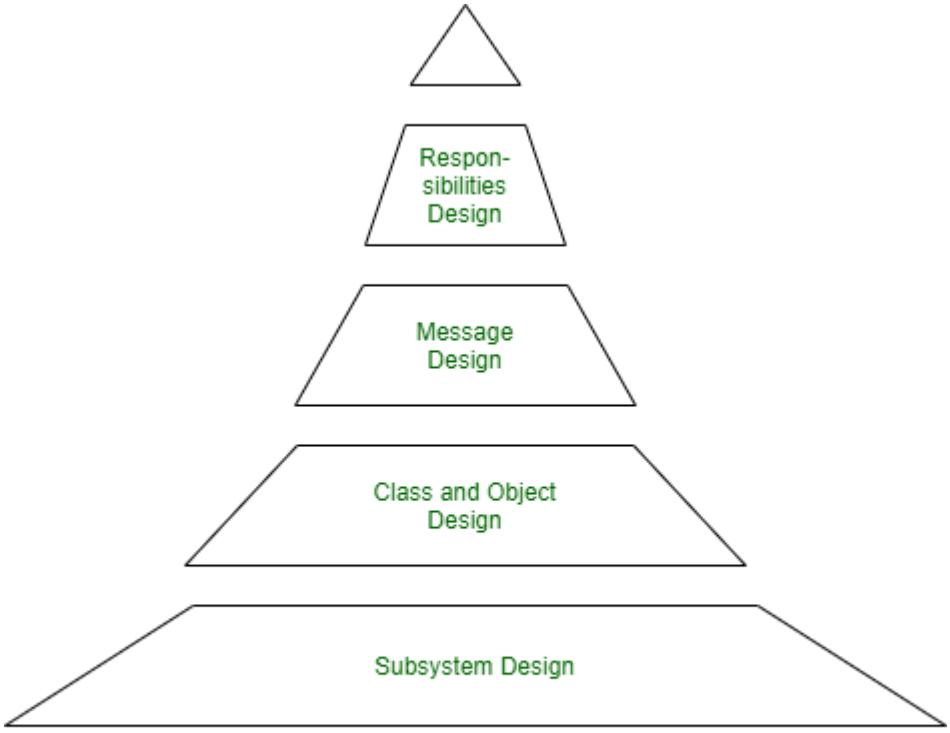
1. The information domain is modeled.
2. Behavior is represented.
3. The function is described.

4. Data, functional, and behavioral models are divided to uncover greater detail.
5. Early models represent the essence of the problem, while later ones provide implementation details.

The above notes principles form the foundation for the OOA approach.

Object-Oriented Design (OOD): An analysis model created using object-oriented analysis is transformed by object-oriented design into a design model that works as a plan for software creation. OOD results in a design having several different levels of modularity i.e., The major system components are partitioned into subsystems (a system-level “modular”), and data manipulation operations are encapsulated into objects (a modular form that is the building block of an OO system.). In addition, OOD must specify some data organization of attributes and a procedural description of each operation. Shows a

design pyramid for object-oriented systems. It is having the following four layers.



The Object Oriented Design Pyramid

1. **The Subsystem Layer :** It represents the subsystem that enables software to achieve user requirements and implement technical frameworks that meet user needs.
2. **The Class and Object Layer :** It represents the class hierarchies that enable the system to develop using generalization and

specialization. This layer also represents each object.

3. **The Message Layer :** It represents the design details that enable each object to communicate with its partners. It establishes internal and external interfaces for the system.
4. **The Responsibilities Layer :** It represents the data structure and algorithmic design for all the attributes and operations for each object.

The Object-Oriented design pyramid specifically emphasizes specific product or system design. Note, however, that another design layer exists, which forms the base on which the pyramid rests. It focuses on the core layer the design of the domain object, which plays an important role in building the infrastructure for the Object-Oriented system by providing support for human/computer interface activities, task management.

Some of the *terminologies* that are often encountered while studying **Object-Oriented Concepts** include:

1. **Attributes**: a collection of data values that describe a class.

2. **Class**: encapsulates the data and procedural abstractions required to describe the content and behavior of some real-world entity. In other words, A class is a generalized description that describes the collection of similar objects.

3. **Objects**: instances of a specific class. Objects inherit a class's attributes and operations.

4. **Operations**: also called methods and services, provide a representation of one of the behaviors of the class.

5. **Subclass**: specialization of the super class. A subclass can inherit both attributes and operations from a super class.

6. **Superclass**: also called a base class, is a generalization of a set of classes that are related to it.

Advantages of OOAD:

1. Improved modularity: OOAD encourages the creation of small, reusable objects that can be combined to create more complex systems, improving the modularity and maintainability of the software.
2. Better abstraction: OOAD provides a high-level, abstract representation of a software system, making it easier to understand and maintain.
3. Improved reuse: OOAD encourages the reuse of objects and object-oriented design patterns, reducing the amount of code that needs to be written and improving the quality and consistency of the software.
4. Improved communication: OOAD provides a common vocabulary and methodology for software developers, improving

communication and collaboration within teams.

5. Reusability: OOAD emphasizes the use of reusable components and design patterns, which can save time and effort in software development by reducing the need to create new code from scratch.
6. Scalability: OOAD can help developers design software systems that are scalable and can handle changes in user demand and business requirements over time.
7. Maintainability: OOAD emphasizes modular design and can help developers create software systems that are easier to maintain and update over time.
8. Flexibility: OOAD can help developers design software systems that are flexible and can adapt to changing business requirements over time.
9. Improved software quality: OOAD emphasizes the use of encapsulation, inheritance, and

polymorphism, which can lead to software systems that are more reliable, secure, and efficient.

Disadvantages of OOAD:

1. Complexity: OOAD can add complexity to a software system, as objects and their relationships must be carefully modeled and managed.
2. Overhead: OOAD can result in additional overhead, as objects must be instantiated, managed, and interacted with, which can slow down the performance of the software.
3. Steep learning curve: OOAD can have a steep learning curve for new software developers, as it requires a strong understanding of OOP concepts and techniques.
4. Complexity: OOAD can be complex and may require significant expertise to implement effectively. It may be difficult for novice developers to understand and apply OOAD principles.

5. Time-consuming: OOAD can be a time-consuming process that involves significant upfront planning and documentation. This can lead to longer development times and higher costs.
6. Rigidity: Once a software system has been designed using OOAD, it can be difficult to make changes without significant time and expense. This can be a disadvantage in rapidly changing environments where new technologies or business requirements may require frequent changes to the system.
7. Cost: OOAD can be more expensive than other software engineering methodologies due to the upfront planning and documentation required.