UNIT – III
SOFTWARE DESIGN

Software Design

1. What is Software design? Explain.

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

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Software Design Levels: Software design yields three levels of results:

- **Architectural Design** - The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

- **High - level Design** - The high - level design breaks the 'single entity - multiple components' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High - level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.

- **Detailed Design** - Detailed design deals with the implementation part of what is seen as a system and its sub - systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

2. Explain Design concepts.

A. A set of fundamental software design concepts has evolved over the past four decades. Each provides the software designer with a foundation from which more sophisticated design methods can be applied.

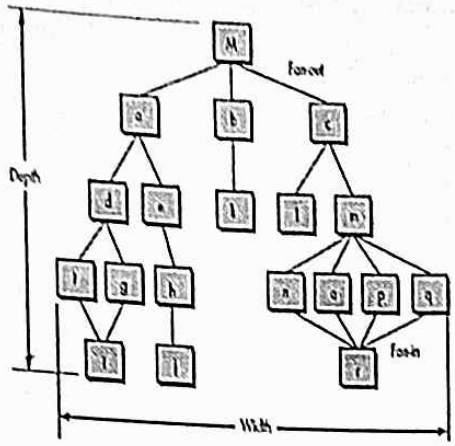
- **Abstraction:** Many levels of abstraction can be posed at modular solution to the problem. At the highest level, a solution is stated in broad terms, and at the lowest level of abstraction a solution is stated in procedural manner.
  There are three levels of abstraction – *procedural abstraction* is a sequence of instructions that has a specific and limited functions. *Data abstraction* is a collection of data that describes a data object. *Control abstraction* implies a program control mechanism without specifying internal details.

- **Refinement:** *Refinement* is a top – down design strategy, is actually a process of elaboration. Refinement causes the designer to elaborate on the original statement providing more and more details. Abstraction enables a designer to suppress low level details, whereas refinement reveals the low level details.

- **Modularity:** Software divided into separately named and addressable components called *modules* that are integrated to satisfy problem requirements. There are five criteria that enables the designers to define effective modular system:
  Modular decomposability
  Modular composability
  Modular understandability
  Modular continuity
  Modular protection

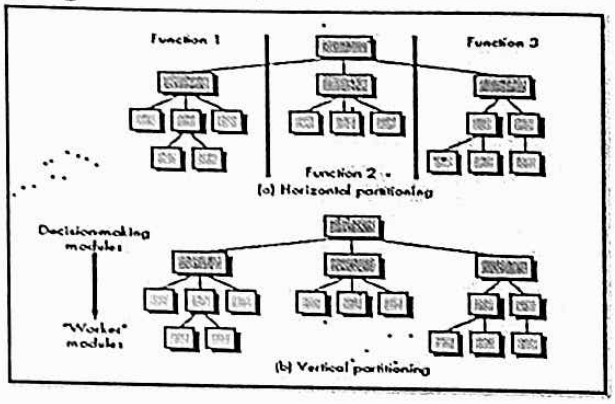
- **Software Architecture:** *Software Architecture* is the hierarchical structure of program components in which these components interact. There are five different models of architectural models available:
  Structural Models
  Framework Models
  Dynamic Models
  Process Models
  Functional Models

- **Control Hierarchy:** *Control hierarchy* is also called as *program structure* which organizes the program components and implies a hierarchy of control. Different notations are used to represent control hierarchy for architectural styles. The most common is *treelike* diagrams.



Depth and *width* provides the indication of the levels of control. *Fan – out* is a measure of number of modules controlled by another module. *Fan – in* is a measure of number of modules control a given module. A module controls other module is *super ordinate*, a module control by other is *sub ordinate*.
The control hierarchy has two characteristics: *Visibility* – the set of components that may be used as data by given component. *Connectivity* – the set of components that are directly used as data by given component.

- **Structural partitioning:** The program structure can be partitioned both horizontally and vertically. *Horizontal partitioning* defines separate branches of program function. *Vertical portioning* called *factoring* should be distributed top – down in the program structure.



- **Data Structure:** *Data structure* is a representation of logical relationship among individual elements of data. *Scalar item* is the simplest of all data structures. It represents a single item. *Sequential vector* is the collection of scalar items.

- **Software Procedure:** Software procedure focuses on the processing details of each module individually.

- **Information Hiding:** The principle of information hiding is the inaccessibility of procedures and data within a module.

****

Effective Modular Design

3. Explain Effective Modular Design.

A. A modular design reduces the complexity, facilitates change, and results in easier implementation.

1] **Functional Independence:** The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding. If we want to design software so that each module addresses a specific sub-function of requirements and has a simple interface, when viewed from other parts of the program structure. Functional independence is a key to good design, and design is the key to software quality.

2] **Cohesion:** Cohesion is a measure that defines the degree of intra dependability within elements of a module. The greater the cohesion the better is the program design. There are 7 types of cohesions:

a. **Co – incidental Cohesion:** It is unplanned and random cohesion, which breaks the program into smaller module. Because of unplanning, it may confuse the programmer.

b. **Logical Cohesion:** When logically categorized elements are put together into a module, is called logical cohesion.

c. **Temporal Cohesion:** When elements of module are organized such that they are processed at a similar point in time. It is called temporal cohesion.

d. **Procedural Cohesion:** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.

e. **Communicational Cohesion:** When elements of module are grouped together, which are executed sequentially and work on same data, is called communicational cohesion.

f. **Sequential Cohesion:** When elements of module are grouped because the output of one element serves as input to another and so on. It is called sequential cohesion.

g. **Functional cohesion:** It is considered to be highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well defined function. It can also be reused.

3] **Coupling:** *Coupling* is measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and

interact with each other. The lower the coupling, the better the program. There are 5 levels of coupling:

a. **Content coupling:** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
b. **Common coupling:** When multiple modules have read and write access to some global data, it is called global or common coupling.
c. **Control coupling:** Two modules are called control couple if one of them decides the function of the other module or changes its flow of execution.
d. **Stamp coupling:** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
e. **Data coupling:** Data coupling is when two modules interact with each other by means of passing data. If a module passes data structure as parameter, then the receiving module should use all its components.

****

Architectural Design and Procedural Design

4. Explain Software Architecture.

A. **Software Architecture:** The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Importance of Software Architecture: There are three key reasons that software architecture is important:

- Representation of software architecture is the communication between all parties who are interested in the development of a computer-based system.
- The early architecture design decisions that will have an impact on all software engineering work that follows.
- Architecture "constitutes a relatively small, understandable model of how the system is structured and how its components work together"

Architecture Description: The IEEE standard defines an *architectural description* (AD) as "a collection of products to document an architecture." The description itself is represented using multiple *views*, where each *view* is "a representation of a whole system from the perspective of a related set of [stakeholder] concerns." A view is created according to rules and conventions defined in a *viewpoint* —"a specification of the conventions for constructing and using a view".

Architectural Decisions: The process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

****

5. What are the Architectural Genres? (Or) Explain Architectural classifications.

A. In architectural design, *genre* implies a specific category within the overall software domain. Within each category, you encounter a number of subcategories. For example, within the genre of buildings, you would encounter the following general styles: houses, condos, apartment buildings, office buildings, industrial building, warehouses, and so on.

Grady Booch suggests the following architectural genres for software-based systems:

- **Artificial intelligence** —Systems that simulate or augment human cognition, locomotion, or other organic processes.

- **Commercial and nonprofit** —Systems that are fundamental to the operation of a business enterprise.

- **Communications** —Systems that provide the infrastructure for transferring and managing data, for connecting users of that data, or for presenting data at the edge of an infrastructure.

- **Content authoring** —Systems that are used to create or manipulate textual or multimedia artifacts.

- **Devices** —Systems that interact with the physical world to provide some point service for an individual.

- **Entertainment and sports** —Systems that manage public events or that provide a large group entertainment experience.

- **Financial** —Systems that provide the infrastructure for transferring and managing money and other securities.

- **Games** —Systems that provide an entertainment experience for individuals or groups.

- **Government** —Systems that support the conduct and operations of a local, state, federal, global, or other political entity.

- **Industrial** —Systems that simulate or control physical processes.

- **Legal** —Systems that support the legal industry.

- **Medical** —Systems that diagnose or heal or that contribute to medical research.
- **Military** —Systems for consultation, communications, command, control, and intelligence as well as offensive and defensive weapons.

- **Operating systems** —Systems that sit just above hardware to provide basic software services.

- **Platforms** —Systems that sit just above operating systems to provide advanced services.

- **Scientific** —Systems that are used for scientific research and applications.

•**Tools**—Systems that are used to develop other systems.

• **Transportation**—Systems that control water, ground, air, or space vehicles.

• **Utilities**—Systems that interact with other software to provide some point service.

****

6. Explain Architectural styles.

A. The software that is built for computer-based systems also exhibits one of many architectural styles. An *architectural style* is a transformation that is imposed on the design of an entire system. Each style describes a system category that encompasses (1) A *set of components* that perform a function required by a system;

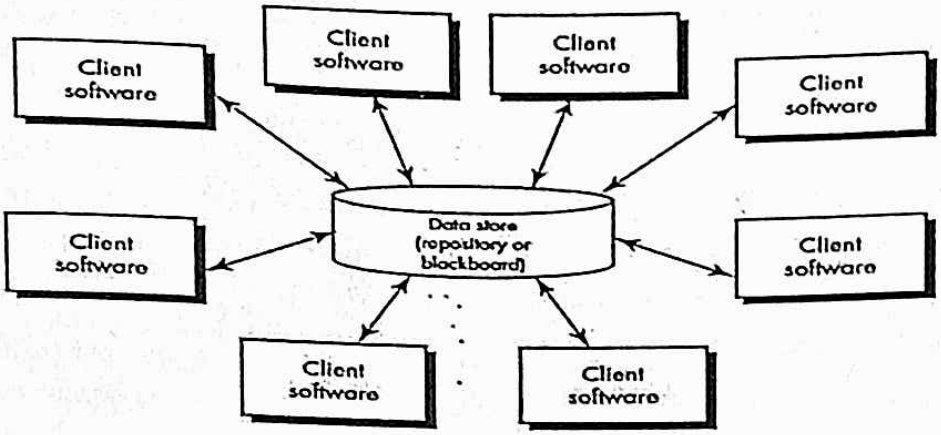
(2) A *set of connectors* that enable "communication, coordination and cooperation" among components;

(3) *Constraints* that define how components can be integrated to form the system;

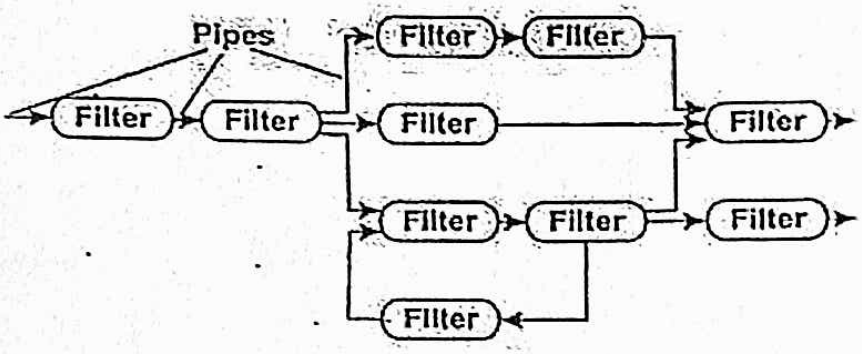
(4) Semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its parts.

Architectural Styles:

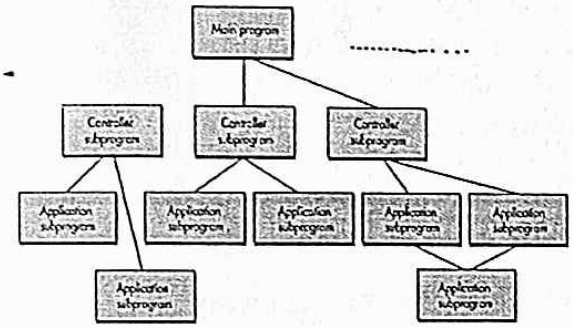
(1) Data-centered architectures: A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.
Data-centered architectures promote integrability, that is, existing components can be changed and new client components added to the architecture without concern about other clients.



(2) **Data-Flow architectures:** This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data. A *pipe-and-filter pattern* has a set of components, called filters, connected by pipes that transmit data from one component to the next.



(3) **Call and Return-architectures:** This architectural style enables you to achieve a program structure that is relatively easy to modify and scale. A number of sub styles exist within this category:

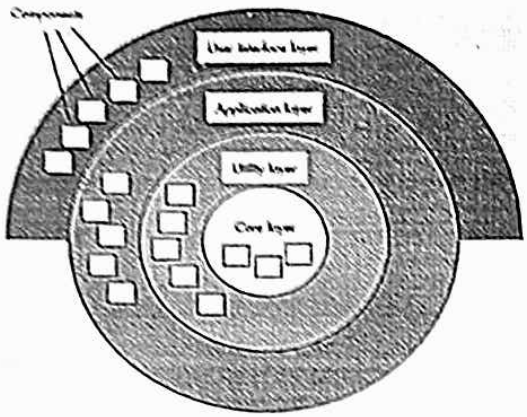


- **Main program/subprogram architectures:** This classic program structure decomposes function into a control hierarchy where a "main" program invokes a number of program components that in turn may invoke still other components.

- **Remote procedure calls architectures:** The components of main program/subprogram architecture are distributed across multiple computers on a network.

(4) Object oriented architectures: The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components are accomplished via message passing.

(5) Layered architectures: A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set. At the outer layer, components service user interface operations. At the inner layer, components perform operating system interfacing. Intermediate layers provide utility services and application software functions.



****

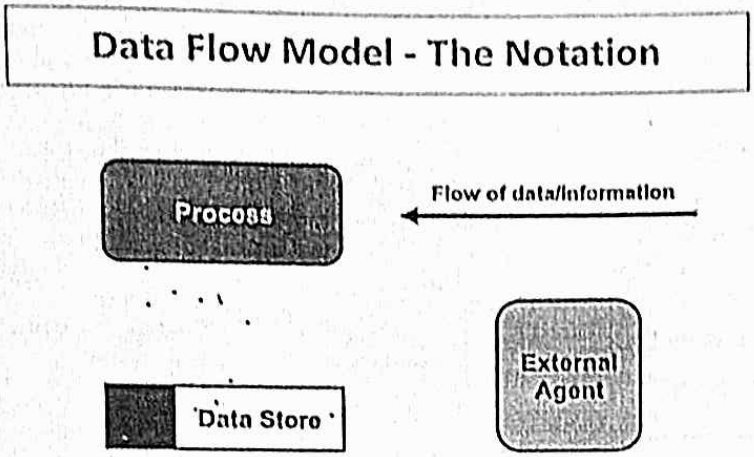
7. Explain Procedural Design.

A. *Procedural design* is best used to model programs that have an obvious flow of data from input to output. It represents the architecture of a program as a set of interacting processes that pass data from one to another. The two major diagramming tools used in procedural design are *data flow diagrams* and *structure charts*.

Data Flow Diagrams: A data flow diagram (or DFD) is a tool to help you discover and document the program's major processes. The following table shows the symbols used and what each represents.

The DFD is a conceptual model – it doesn't represent the computer program, it represents what the program must accomplish. By showing the input and output of

each major task, it shows how data must move through and be transformed by the program.



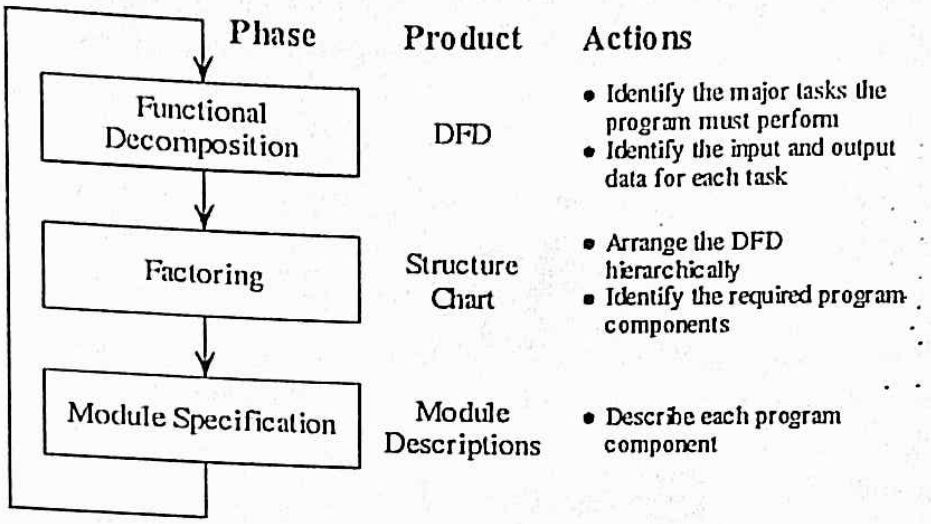
Structure Charts: A structure chart is a tool to help you derive and document the program's architecture. It is similar to an organization chart. When a component is divided into separate pieces, it is called the parent and its pieces are called its children. The structure chart shows the hierarchy between a parent and its children.

Structure Chart Symbols	
Symbol	Description
Component	A major component within the program
	Connects a parent component to one of its children
Name	Data that is passed between components

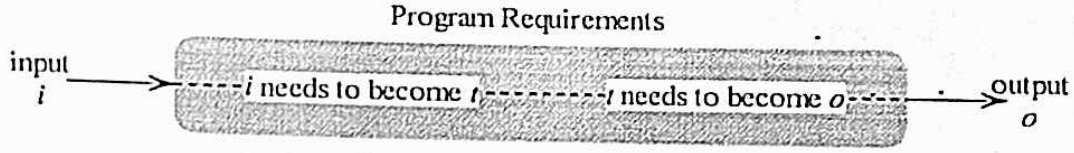
****

8. Explain Procedural Design methodology. (Or) Explain the steps involved in creating procedural design.

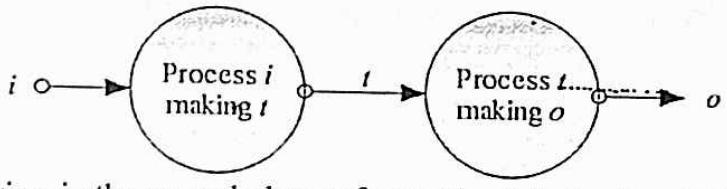
A. Here are the basic steps you follow to create a procedural design.



Functional Decomposition: In computer programming, decomposition is the process of dividing a large entity into more manageable pieces. For a procedural design, this me and dividing tasks into sequences of smaller tasks, which is functional decomposition. One technique for doing this, called *data flow analysis*, involves:
(1) Identifying a major data flow,
(2) Following it from input to output,
(3) Determining where it undergoes a major transformation and
(4) Dividing the processing at that point.
To illustrate, given the following program requirements:

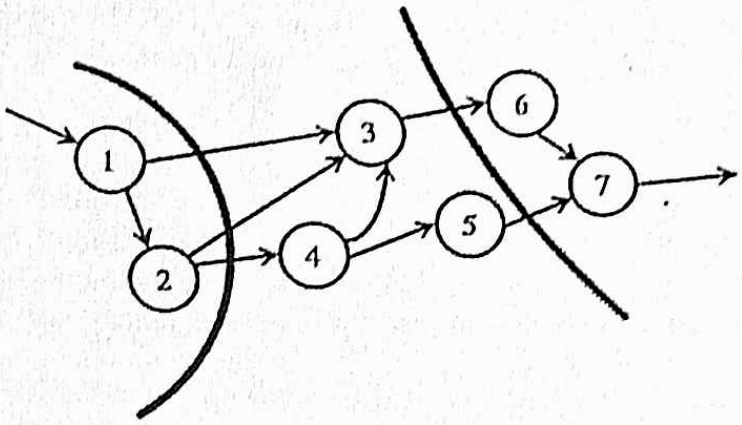


Data flow analysis yields:



Factoring: Factoring is the second phase of procedural design in which you create a structure chart that shows what program components need to be implemented. We do this in two passes. First, arrange your DFD hierarchically. Second, identify exactly which conceptual processes are to be implemented as physical components in the program.

To arrange your DFD hierarchically, cut it into three partitions: (a) processes that prepare input for the main computation, (b) processes that perform the main computation and (c) processes that prepare the output.



Module Specification: Module Specification is the act of documenting your program design by fully describing each of its modules. Module is a general term that can refer to any manner of computer program components, including a single method, a single class, a single object or a collection of related methods.

When the module is a single method, the following facts must be provided. Seasoned programmers generally write them using a combination of computer language and English.

- The method's name
- A description of what it does
- Its number of arguments and the data type and purpose of each
- A description of the return value and its data type
- A description of any exceptions that it may throw.

It is often useful to specify the behavior of a method by stating its preconditions and post conditions. A precondition is a statement of what must be true when the method is called; a post condition is a statement of what must be true when the method returns.

****