UNIT-5
Software Design

1) Explain design concepts.

A set of fundamental software design concepts has evolved over the past four decades. Although the degree of interest in each concept has varied over the years, each has stood the test of time.

1.1 Abstraction

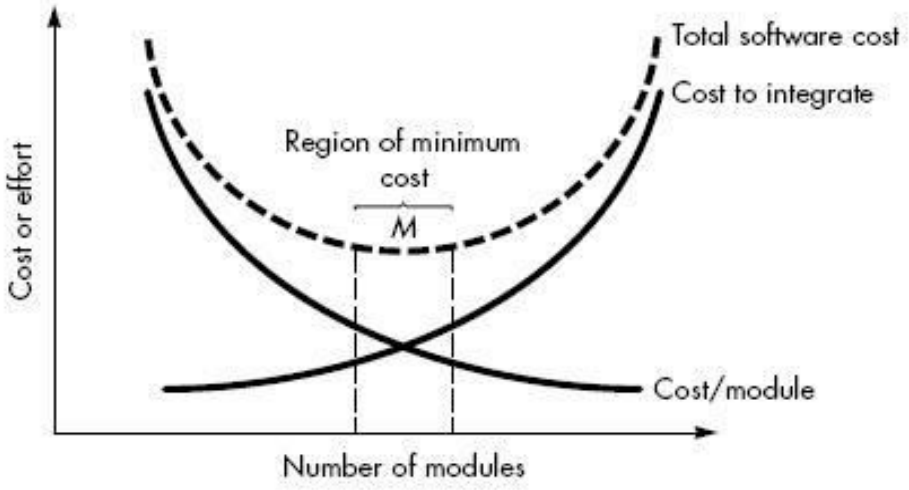
- Abstraction basically gets divided into two parts procedural abstraction and data abstraction.
- When we consider a modular solution to any problem many levels of abstraction can be posed.
- At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment.
- At lower levels of abstraction, a more procedural orientation is taken.
- Problem-oriented terminology is coupled with implementation- oriented terminology in an effort to state a solution.

1.2 Refinement

- Refinement is actually a process of elaboration.
- We begin with a statement of function that is defined at a high level of abstraction.
- That is, the statement describes function or information conceptually but provides no information about the internal workings of the function or the internal structure of the information.
- We need to elaborate on the original statement.
- Stepwise refinement is a top-down design strategy.
- Abstraction and refinement are complimentary concepts.

1.3 Modularity

- Software architecture has the concept of modularity basically modularity is nothing but dividing software into separately named and addressable components which often called modules that are integrated to satisfy problem requirements.
- As we increase number of modules apparently cost of integration will increase apparently. So number of module in our software should be in region of minimum cost.



1.4 Software Architecture

- Software architecture suggests the overall structure of the software and the ways in which that structure

  provides conceptual integrity for a system.
- Number of different models can use to represent architecture which are described over here.
- Structural Model which represents architecture as an organized collection of components
- Framework model Increase level of design abstraction by identifying repeatable architectural design framework.
- Dynamic model address behavior of the program architecture and indicating how system or structure configuration may change as a function.
- Process Model focus on design of the business or technical process that the system must accommodate.
- Functional models used to represent the functional hierarchy of a system.

1.5 Control Hierarchy

- Control hierarchy, also called program structure, represents the organization of program components and implies a hierarchy of control.
- It does not represent procedural aspects of software such as sequence of processes, occurrence or order of decisions, or repetition of operations; nor is it necessarily applicable to all architectural styles.

1.6 Structural Partitioning

- If the architectural style of a system is hierarchical, the program structure can be partitioned both horizontally and vertically.
- Here, horizontal partitioning defines separate branches of the modular hierarchy for each major program function. Control modules, represented in a darker shade are used to coordinate communication between and execution of the functions.
- Vertical partitioning (Figure 13.4b), often called factoring, suggests that control (decision making) and work should be distributed top-down in the program structure.

1.7 Data Structure

- Data structure is a representation of the logical relationship among individual elements of data.
- Because the structure of information will invariably affect the final procedural design, data structure is as important as program structure to the representation of software architecture.

1.8 Information Hiding

- In information Hiding modules should be specified and designed so that information (algorithm and data) contained within a module is inaccessible to other modules that have no need for such information.
- The intent of information hiding is to hide the details of data structure and procedural processing behind a module interface.

1.9 Software Procedure

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

- Procedure must provide a precise specification of processing, including sequence of events, exact decision points, repetitive operations, and even data organization and structure.

2) Explain coupling and cohesion.

Coupling

Coupling is a measure of interconnection among modules in a software structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.

- In software design, we strive for lowest possible coupling. Simple connectivity among modules results in software that is easier to understand and less prone to a "ripple effect", caused when errors occur at one location and propagates through a system.

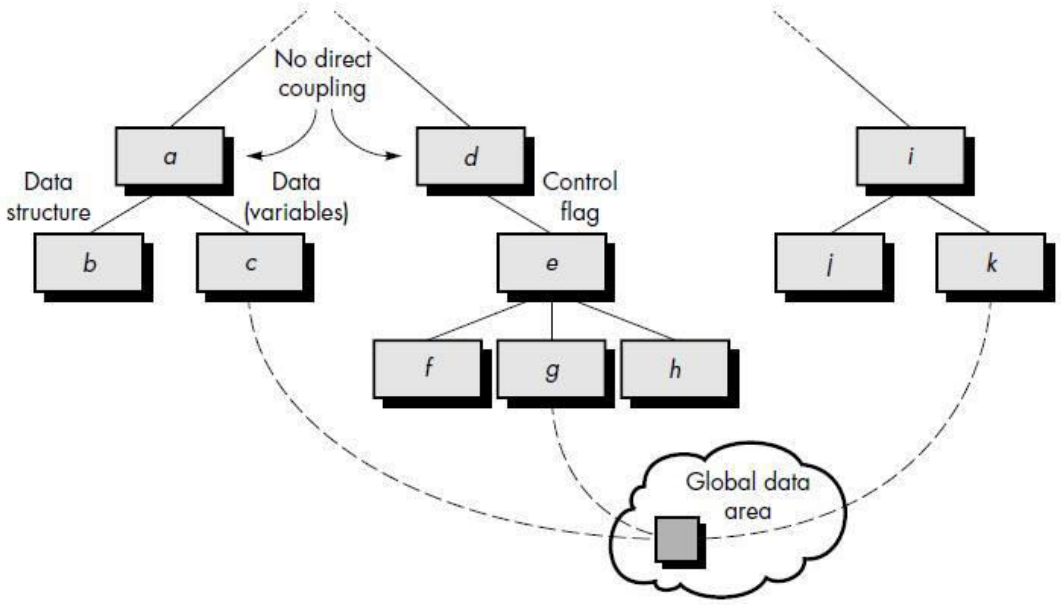


Figure provides examples of different types of module coupling. Modules A and D are subordinate to different modules. Each is unrelated and therefore no direct coupling occurs.

Types of coupling

Data coupling:

Module C is subordinate to module A and is accessed via a conventional argument list, through which data are passed. As long as a simple argument list is present, low coupling (called data coupling) is exhibited in this portion of structure.

Stamp coupling:

A variation of data coupling, called stamp coupling is found when a portion of a data structure (rather than simple arguments) is passed via a module interface. This occurs between modules B and A.

Control coupling:

At moderate levels, coupling is characterized by passage of control between modules. Control coupling is very common in most software designs and is shown in Figure where a “control flag” is passed between modules D and E.

External coupling:

Relatively high levels of coupling occur when modules are tied to an environment external to
software. For example, I/O couples a module to specific devices, formats, and communication protocols. External coupling is essential, but should be limited to a small number of modules with a structure.

Common coupling:

High coupling also occurs when a number of modules reference a global data area. Common coupling, as this mode is called, is shown in Figure. Modules C, G, and K each access a data item in a global data area. Module C initializes the item. Later module G recomputed and updates the item. Let's assume that an error occurs and G updates the item incorrectly.

Content coupling:

The highest degree of coupling, content coupling, occurs when one module makes use of data or control information maintained within the boundary of another module. Secondarily, content coupling occurs when branches are made into the middle of a module. This mode of coupling can and should be avoided.

Stamp Coupling:

Occurs when class B is declared as a type for an argument of an operation of class A. Because Class B is now a part of the definition of Class A,, modifying the system becomes more complex.

Routine Call coupling:

Occurs when one operation invokes another. This level of coupling is common and is necessary. However, it does increase the connectedness of a system.

Inclusion or Import coupling:

Occurs when component A imports or includes a package or the content of component B.

Cohesion:

- Cohesion is a natural extension of the information hiding concept.
- A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.
- We always look for high cohesion, although the midrange of the spectrum is often acceptable.
- Low-end cohesiveness is much "worse" than middle range, which is nearly as "good" as high-

end cohesion.

- So designer should avoid low end cohesion.
- When processing elements of a module are related and must be executed in a specific order, procedural cohesion exists.
- When all processing elements concentrate on one area of a data structure, communicational cohesion is present.
- High cohesion is characterized by a module that performs one distinct procedural task.

Types of Cohesion

Logical cohesion

When module performs tasks that are related logically at that time logical cohesion is exist.

Temporal cohesion.

When a module contains tasks that are related by the fact that all must be executed with the same span of time at that time temporal cohesion exist.

Coincidentally cohesion

At the low-end of the spectrum, a module that performs a set of tasks that relate to each other loosely at that time coincidental cohesion exists.

3) Compare Coupling and Cohesion

Cohesion	Coupling
• Cohesion of a single module/component is the degree to which its responsibilities form a meaningful unit	• Coupling between modules/components is their degree of mutual interdependence
• A highly-cohesive system is one in which all procedures in a given module work together towards some end goal.	• A highly coupled system in one in which the procedures in one module can directly access elements of another module.
• High cohesion is often characterized by high readability and maintainability.	• Highly coupled systems are often characterized by code that is difficult to read and maintain.
• Higher cohesion is better.	• Lower coupling is better.
• Types of Cohesion may be Logical, temporal, procedural etc.	• Types of Coupling may be data, content, common, stamp etc.

4) Explain design model.

In the figure, the design model is represented as a pyramid.

- The symbolism of this shape is important. A pyramid is an extremely stable object with a wide base and a low center of gravity. Like the pyramid, we want to create a software design that is stable.
- **Data design.**

  Establishment of broad foundation

can be complited by data design

- **Architectural Design.**

  Architectural Design is next layer it provides stability to midpoint region.

- **Interface Design.**

  Interface Design also provides stability which architectural design does however functionality of two layer is totally different.

- **Component level Design.**

  Component level design is also shown here using

  sharp point a sharp point we create a design model that is not easily “tipped over” by the winds of change.

- It is interesting to note that some programmers continue to design implicitly, conducting component-level design as they code.
- This is akin to taking the design pyramid and standing it on its point—an extremely unstable design results. The smallest change may cause the pyramid (and the program) to topple.

5) Explain data design in brief. Data Design

- Data design creates a model of data and information that is represented at a high level of abstraction. This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system.
- The structure of data has always been an important part of software design.
- Data design plays an important role at component level, application level and business level.

Data Design at Architectural Level

- In Data Design at Architectural Level the challenge for a business has been to extract useful information from this data environment, particularly when the information desired is cross-functional (e.g., information that can be obtained only if specific marketing data are cross-correlated with product engineering data).
- To solve this challenge, the business IT community has developed data mining techniques, also called knowledge discovery in databases(KDD), that navigate through existing databases in an attempt to extract appropriate business-level information.
- However, the existence of multiple databases, their different structures, and the degree of detail contained with the databases, and many other factors make data mining difficult within an existing database environment.

Data Design at Component Level

- Data design at the component level focuses on the representation of data structures that are directly accessed by one or more software components.
- Also, a set of proposed principles that may be used to specify and design such data structures.
- In actuality, the design of data begins during the creation of the analysis model.
- Recalling that requirements analysis and design often overlap, we consider the following set of principles for data specification:

- The systematic analysis principles applied to function and behavior should also be applied to data.
- All data structures and the operations to be performed on each should be identified.
- A data dictionary should be established and used to define both data and pro-gram design.
- Low-level data design decisions should be deferred until late in the design process.
- The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.
- A library of useful data structures and the operations that may be applied to them should be developed.
- A software design and programming language should support the specification and realization of abstract data types.

6) Explain its architectural styles and patterns.

Architectural Styles and Patterns It can be presented in the following ways:-

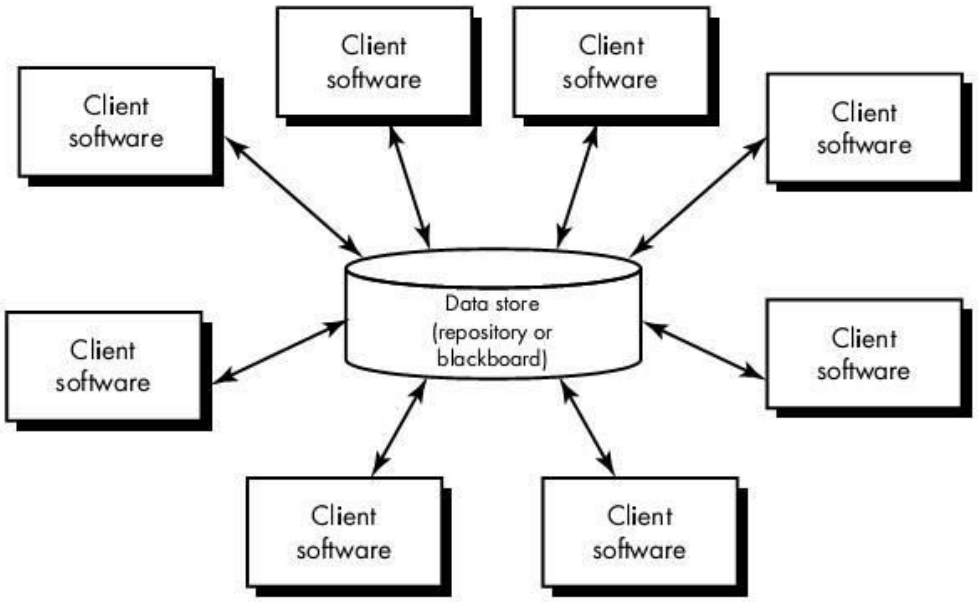
1. Data-centered Architecture.
2. Data flow architecture
3. Call and return architecture
4. Object oriented architecture.
5. Layered architecture.

Data Centered Architecture.

A data store resides at the corner of this architecture and is accessed frequently by other components that update, add, delete or otherwise modify data within the store.

Client software accesses a central repository which is in passive state Client software accesses the data independent of any changes to the data or the actions of other client software.

- So, in this case transform the repository into a “Blackboard”



Data Centered Architecture

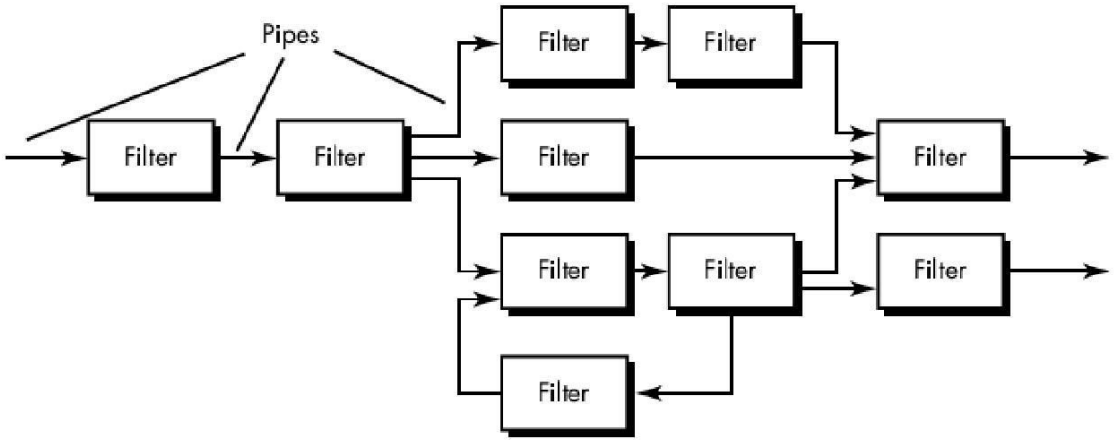
- A blackboard sends notification of subscribers when data of interest changes, and is thus active. Data-centered architectures promote inerrability.

- Existing components can be changed and new client components can be added to the architecture without concern about other clients.
- Data can be passed among the clients using the blackboard mechanism. So client components independently execute processes.

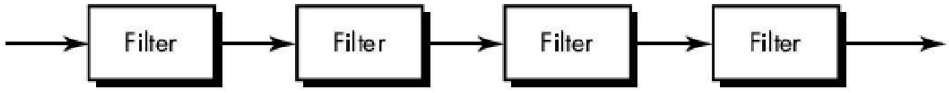
Data Flow Architecture

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 (as in figure.) has a set of components, called filters, connected by pipes that transmit data from one component to the next.
- Each filter works independently (i.e. upstream, downstream) and is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form.
- The filter does not require knowledge of the working of its neighboring filters.
- If the data flow degenerates into a single line of transforms, it is termed batch sequential.



(a) Pipes and filters



(b) Batch sequential

Call and Return Architecture

Architecture styles enable a software designer (system architect) to achieve a program structure that is relatively easy to modify and scale.

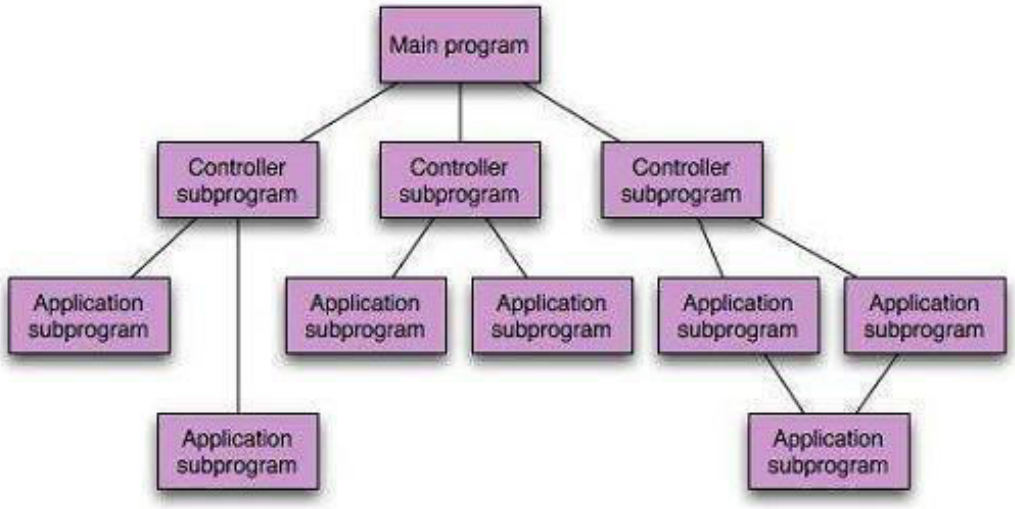
Two sub styles exist within this category:

Main/sub program architecture:

- Program structures decompose function into a control hierarchy where a "main" program invokes a number of program components, which in turn may invoke still other components.

Remote procedure Call architecture:

- The components of main program/subprogram architecture are distributed across multiple computers on a network. Call and Return Architecture shown below



Object oriented Architecture.

- Here the object oriented paradigm, like the abstract data type paradigm from which it evolved, emphasizes the bundling of data and methods to manipulate access that data (Public Interface).
- Components of a system summarize data and the operation that must be applied to manipulate the data.
- Communication and coordination between components is accomplished via message passing.

Layered Architecture

- A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.
- At the outer layer, components examine user interface operations.
- At the inner layer, components

examine operating system interfacing.

- Intermediate layers provide utility services and application software functions.

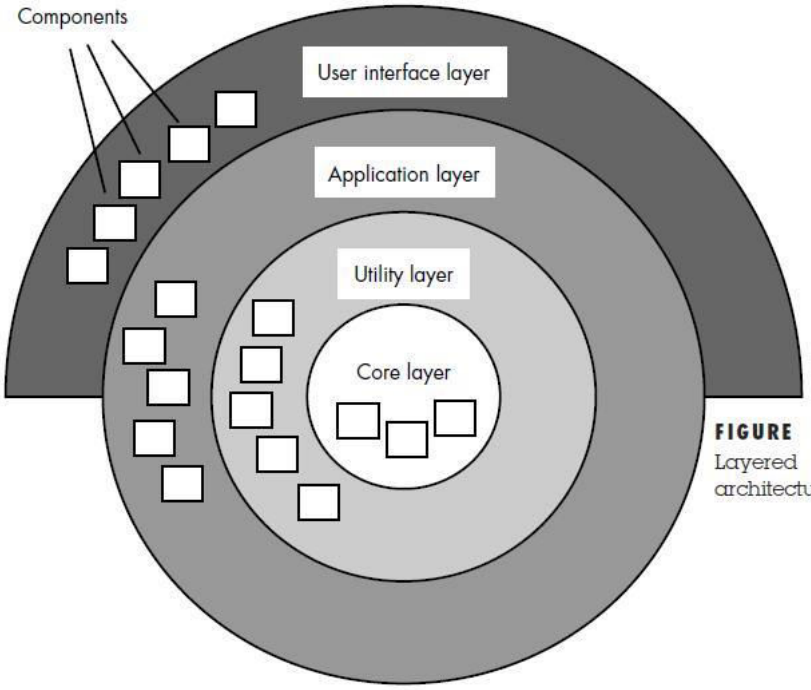


FIGURE Layered architecture

7) Explain structural partitioning.

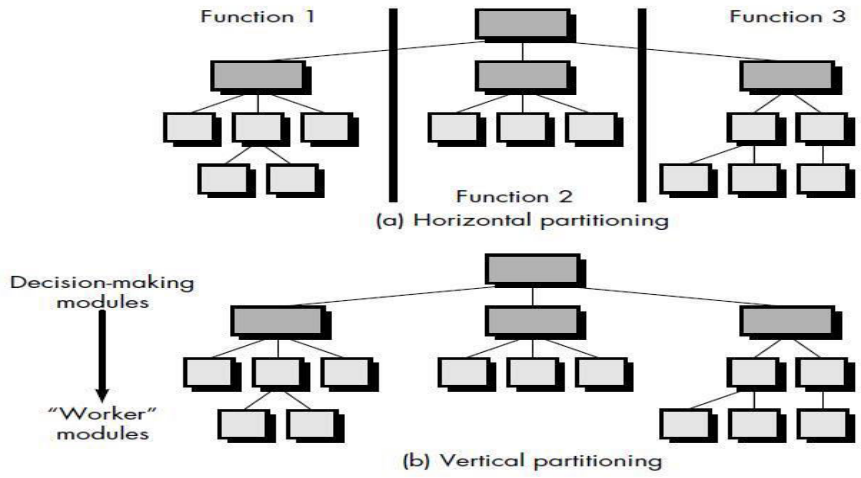
If the architectural style of a system is hierarchical, the program structure can be partitioned both horizontally and vertically.

Horizontal partitioning:

- It defines separate branches of the modular hierarchy for each major program function. Control modules, represented in a darker shade are used to coordinate communication between and execution of the functions. The simplest approach to horizontal partitioning defines three partitions—input, data transformation (often called processing) and output. Partitioning the architecture horizontally provides a number of distinct benefits:

    - software that is easier to test
    - software that is easier to maintain
    - propagation of fewer side effects
    - software that is easier to extend

- Because major functions are decoupled from one another, change tends to be less complex and extensions
  to the system (a common occurrence) tend to be easier to accomplish without side effects. On the negative side, horizontal partitioning often causes more data to be passed across module interfaces and can complicate the overall control of program flow (if processing requires rapid movement from one function to another).



Vertical partitioning:

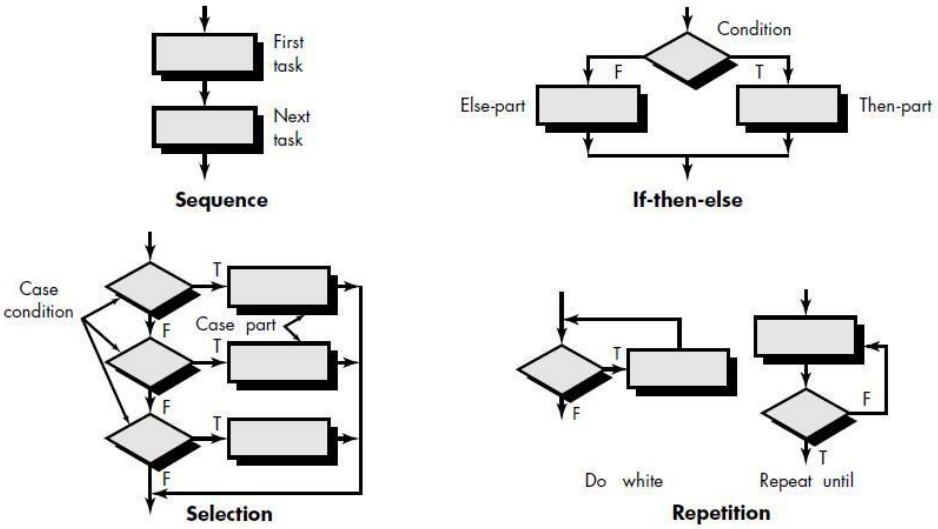
- It often called factoring, suggests that control (decision making) and work should be distributed top-down in the program structure. Top level modules should perform control functions and do little actual processing work.
- Modules that reside low in the structure should be the workers, performing all input, computation, and output tasks.
- The nature of change in program structures justifies the need for vertical partitioning. It can be seen that a change in a control module (high in the structure) will have a higher probability of propagating side effects to modules that are subordinate to it. A change to a worker module, given its low level in the structure, is less likely to cause the propagation of side effects.
- In general, changes to computer programs revolve around changes to input, computation or transformation, and output. The overall control structure of the program. For this reason vertically partitioned structures are less likely to be susceptible to side effects when changes are made and will therefore be more maintainable a key quality factor.

8) Explain procedural design or explain component level design.

- It is possible to represent the component-level design using a programming language.
- In essence, the program is created using the design model as a guide.
  An alternative approach is to represent the procedural design using some intermediate (e.g., graphical, tabular, or text-based) representation that can be translated easily into source code.
- Regardless of the mechanism that is used to represent the component level design, the data structures, interfaces, and algorithms defined should conform to a variety of well-established procedural design guidelines that help us to avoid errors as the procedural design evolves.

Structured programming

- The constructs are sequence, condition, and repetition. Sequence implements processing steps that are essential in the specification of any algorithm.
- Condition provides the facility for selected processing based on some logical occurrence, and repetition allows for looping.
- These three constructs are fundamental to structured programming—an important component-level design technique.



Graphical Design Notation

Here, if graphical tools are misused, the wrong picture may lead to the wrong software.

- A flowchart is quite simple pictorially.
- A box is used to indicate a processing step.
- A diamond represents a logical condition, and arrows show the flow of control. Figure above illustrates three structured constructs.
- The sequence is represented as two processing boxes connected by a line (arrow) of control.
- Condition, also called if then-else, is depicted as a decision diamond that if true, causes then-part processing to occur, and if false, invokes else-part processing.
- Repetition is represented using two slightly different forms.
- The do while tests a condition and executes a loop task repetitively as long as the condition holds true, it repeats until executes the loop task first, then tests a condition and repeats the task until the condition fails.
- The selection (or select-case) construct shown in the figure is actually an extension of the if-then-else.
- A parameter is tested by successive decisions until a true condition occurs and a case part processing path is executed.
- Another graphical design tool, the box diagram, evolved from a desire to develop a procedural design representation that would not allow violation of the structured constructs.

- It is also called, Chapin charts or N-S chart have the following characteristics:

(1) Functional domain (that is, the scope of repetition or if-then-else) is well defined and clearly visible as a pictorial representation.
(2) Arbitrary transfer of control is impossible.
(3) The scope of local and/or global data can be easily determined.
(4) Recursion is easy to represents.

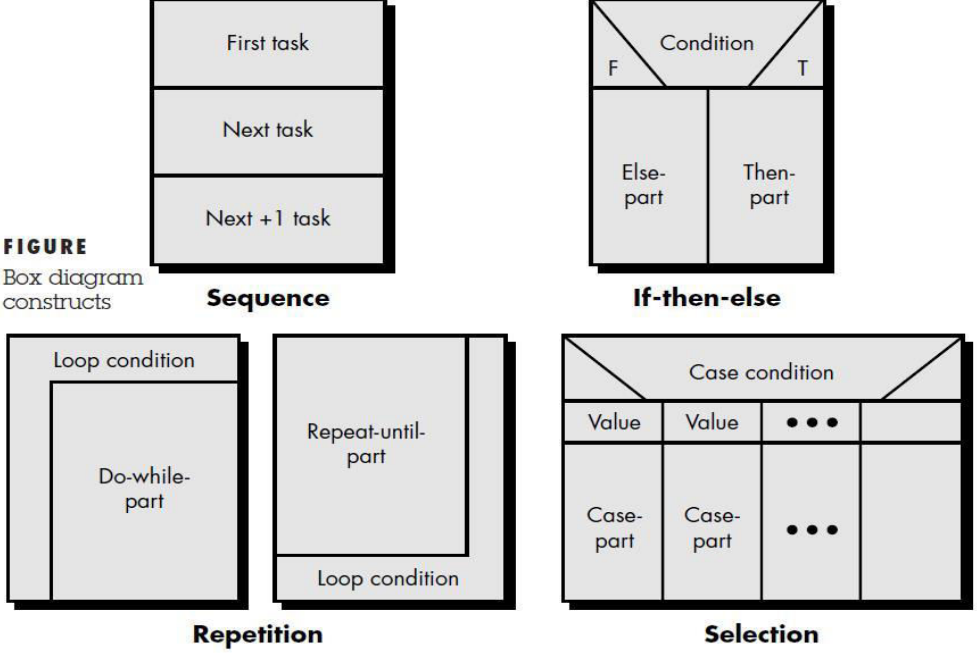


FIGURE
Box diagram constructs

Tabular Design Notation

- Decision table organization is illustrated in Figure above.
- The table is divided into four sections.
- The upper left-hand quadrant contains a list of all conditions.
- The lower left-hand quadrant contains a list of all actions that are possible based on combinations of conditions.
- The right-hand quadrants form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination.
- Therefore, each column of the matrix may be interpreted as a

  processing rule. The following steps are applied to develop a

  decision table:

1. List all actions that can be associated with a specific procedure (or module).
2. List all conditions (or decisions made) during execution of the procedure.
3. Associate specific sets of conditions with specific actions, eliminating impossible combinations of conditions; alternatively, develop every possible permutation of conditions.
4. Define rules by indicating what action(s) occurs for a set of conditions.

Program Design Language.

A program design language may be a simple transposition of a language such as Ada or C. • Alternatively, it may be a product purchased specifically for procedural design.

FIGURE Decision table nomenclature

Conditions	Rules 1	2	3	4					n
Condition #1	✓			✓	✓				
Condition #2		✓		✓					
Condition #3			✓		✓				
Actions									
Action #1	✓			✓	✓				
Action #2		✓		✓					
Action #3			✓						
Action #4			✓	✓	✓				
Action #5	✓	✓			✓				

- Regardless of origin, a design language should have the following characteristics:

A fixed syntax of keywords that provide for all structured constructs, data declaration, and modularity characteristics.

- A free syntax of natural language that describes processing features.
- Data declaration facilities that should include both simple (scalar, array) and complex (linked list or tree)
  data structures.
- Subprogram definition and calling techniques that support various modes of interface description.

FIGURE Resultant decision table

Conditions	Rules 1	2	3	4	5
Fixed rate acct.	T	T	F	F	F
Variable rate acct.	F	F	T	T	F
Consumption <100 kwh	T	F	T	F	
Consumption ≥100 kwh	F	T	F	T	
Actions					
Min. monthly charge	✓				
Schedule A billing		✓	✓		
Schedule B billing				✓	
Other treatment					✓

9) What is object oriented design of a system?

- Object Oriented Design contains four types of layers included in a system. These four layers in OOD are in pyramid form as shown in figure.

- **The subsystem layer**

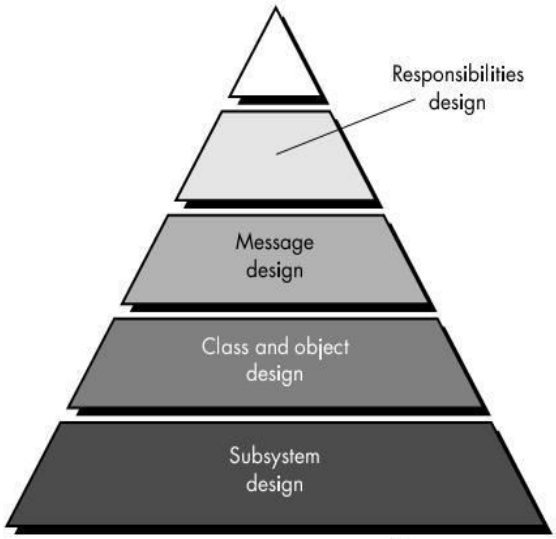


FIGURE The OO design pyramid

It contains a representation of each of the subsystems that enable the software to achieve its customer-defined requirements and to implement the technical infrastructure that supports customer requirements.

- **The class and object layer**

  It contains the class hierarchies that enable the system to be created using generalizations and increasingly more targeted specializations. This layer also contains representations of each object.

- **The message layer**

  It contains the design details that enable each object to communicate with its collaborators. This layer establishes the external and internal interfaces for the system.

- **The responsibilities layer**

  It contains the data structure and algorithmic design for all attributes and operations for each object. The design pyramid focuses exclusively on the design of a specific product or system.

  It should be noted, however, that another "layer" of design exists, and this layer forms the foundation on which the pyramid rests.

- The foundation layer focuses on the design of domain objects.
- Domain objects play a key role in building the infrastructure for the OO system by providing support for human and computer interface activities, task management, and data management.
- Domain objects can also be used to flesh out the design of the application itself.

10) Compare the relative advantages of function oriented and object oriented approaches to software design.

Function Oriented Design	Object Oriented design
• In function-oriented design, functions are grouped together by which a higher-level function is obtained.	• In this design, the functions are grouped together on the basis of the data they operate on, such as in class person, function displays are made member functions to operate on its data members such as the person name, age, etc.
• In this approach, the state information is often represented in a centralized shared memory.	• In this approach, the state information is not represented in a centralized shared memory but is implemented / distributed among the objects of the system.
• The Data Flow Diagram (DFD), Control Flow Diagrams (CFD) and Entity Relationship Diagrams (ERD) can be drawn in structured analysis.	• The use cases, class diagram, sequence diagram state chart diagram, activity diagram, component diagrams can be drawn in this method.
• This is a simple technique of a system analysis.	• In this technique, the overhead of partitioning the structure into modules is involved.
• Finding bugs is difficult because focus of this approach is merely on functionalities. That means the system structure is according to the functionalities.	• Finding bugs is simple because the system structure is modular.