4.1 <u>UML (CLASS DIAGRAM, USE CASE):</u>

REFER FROM PPT

4.2 <u>DATA FLOW DIAGRAM (DFD):</u>

- Through a structured analysis technique called data flow diagrams (DFD), the systems analyst can put together a graphical representation of data processes throughout the system.
- The data flow approach emphasizes the logic underlying the system.

- There are two types of DFD:
  1. Physical Data Flow Diagrams
  2. Logical Data Flow Diagrams

1. Physical Data Flow Diagrams:

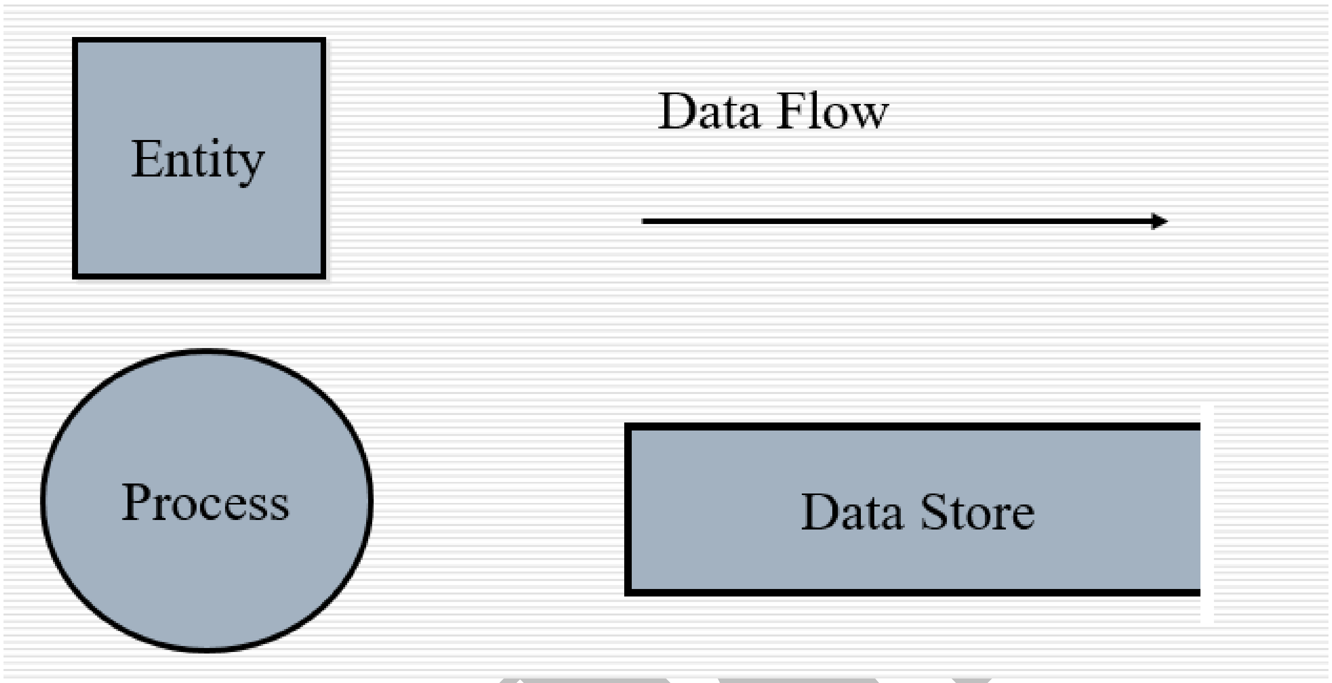
An implementation-dependent view of the current system, showing what tasks are carried out and how they are performed. Physical characteristics can include:

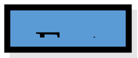
- Names of people
- Form and document names or numbers
- Names of departments
- Master and transaction files
- Equipment and devices used
- Locations
- Names of procedures

2. Logical Data Flow Diagrams

- An implementation-independent view of the a system, focusing on the flow of data between processes without regard for the specific devices, storage locations or people in the system.

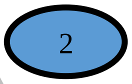
Symbols used in DFD:



- **Entity:** 

An external entity (sink) (e.g. a customer, supplier, department, employee) is a source or destination of a data flow which is outside the area of study. Only those entities which originate or receive data are represented on a data flow diagram. The symbol used is an square containing a meaningful and unique identifier.

- **Data Flow:**

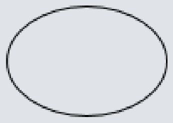
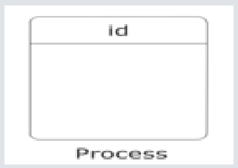
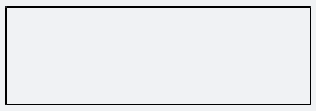
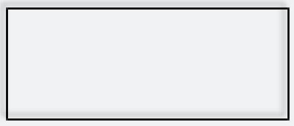
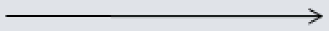
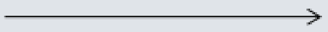
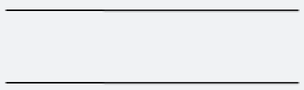
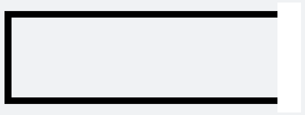
A data flow shows the flow of information from its source to its destination. A data flow is represented by a line, with arrowheads showing the direction of flow. Information always flows to or from a process and may be written, verbal or electronic. 

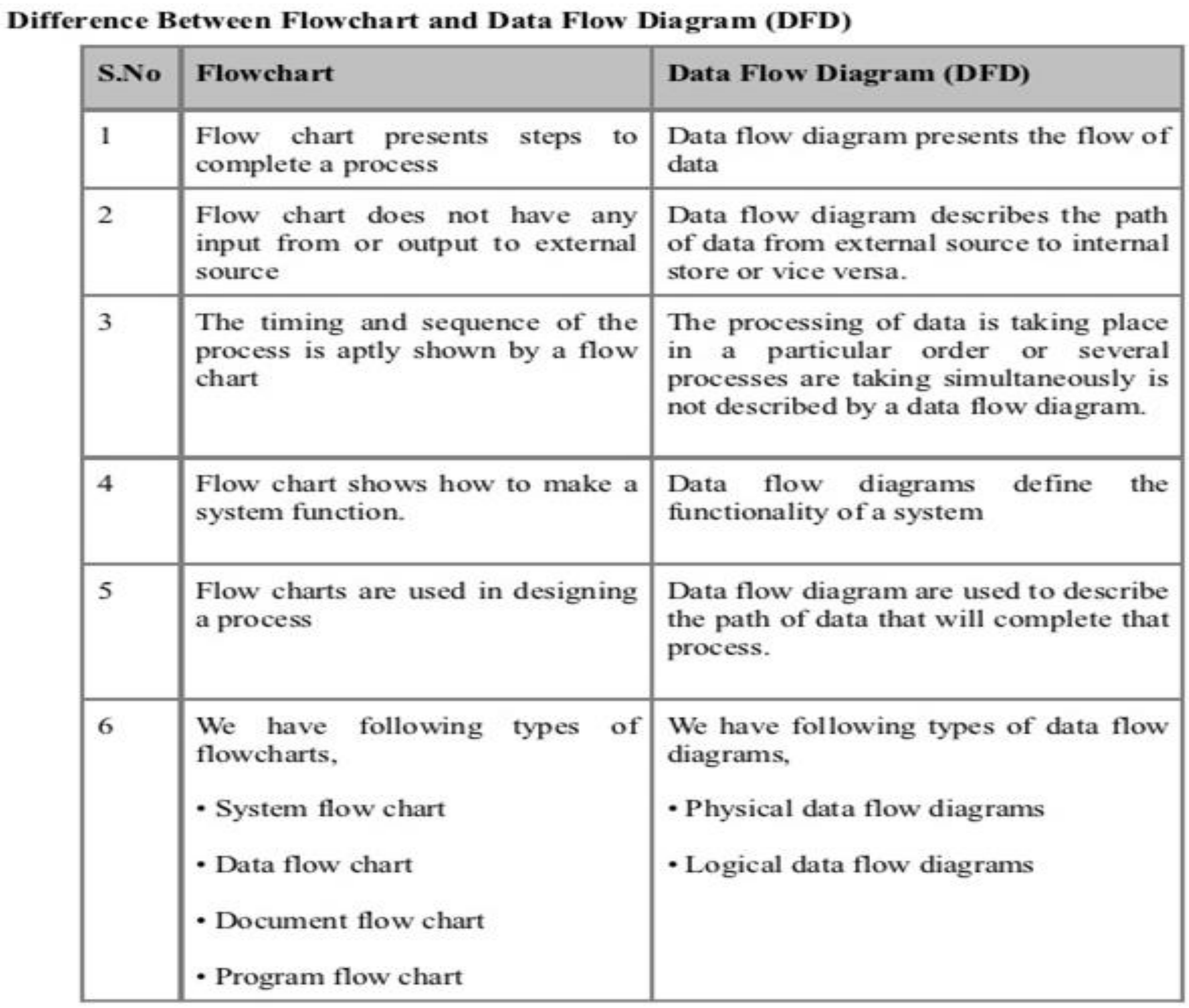
- **Process:**

A process shows a transformation or manipulation of data flows within the system; hence the data flow leaving a process is always labeled differently from the one entering it. Processes represent work being performed within the system. The symbol used is a circle which contains 2 descriptive elements: a unique identifier and a descriptive name

- **Data Store:**

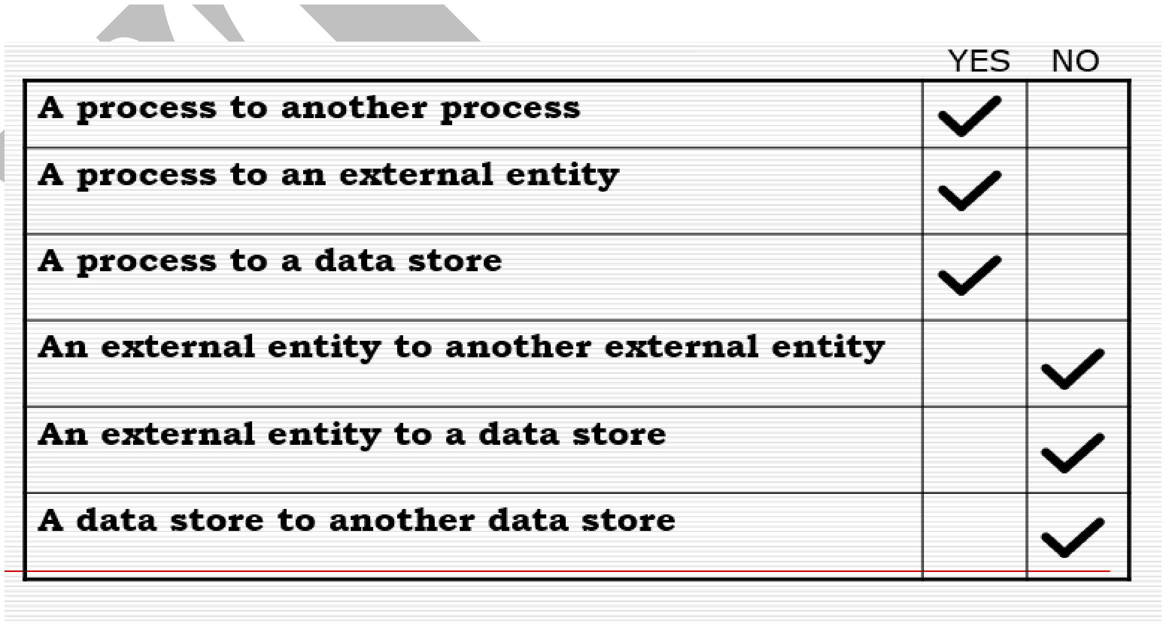
A data store can be thought of as data at rest. It is typically a database but it could be a filing cabinet, a notebook or a datafile. It could also be a permanent or a temporary store. It is represented by an open ended narrow rectangle.

Symbol	Yourdan	Gane & Sarson
Process		
External Entity		
Data Flow		
Data Store		



Difference Between Flowchart and Data Flow Diagram (DFD)

S.No	Flowchart	Data Flow Diagram (DFD)
1	Flow chart presents steps to complete a process	Data flow diagram presents the flow of data
2	Flow chart does not have any input from or output to external source	Data flow diagram describes the path of data from external source to internal store or vice versa.
3	The timing and sequence of the process is aptly shown by a flow chart	The processing of data is taking place in a particular order or several processes are taking simultaneously is not described by a data flow diagram.
4	Flow chart shows how to make a system function.	Data flow diagrams define the functionality of a system
5	Flow charts are used in designing a process	Data flow diagram are used to describe the path of data that will complete that process.
6	We have following types of flowcharts, • System flow chart • Data flow chart • Document flow chart • Program flow chart	We have following types of data flow diagrams, • Physical data flow diagrams • Logical data flow diagrams



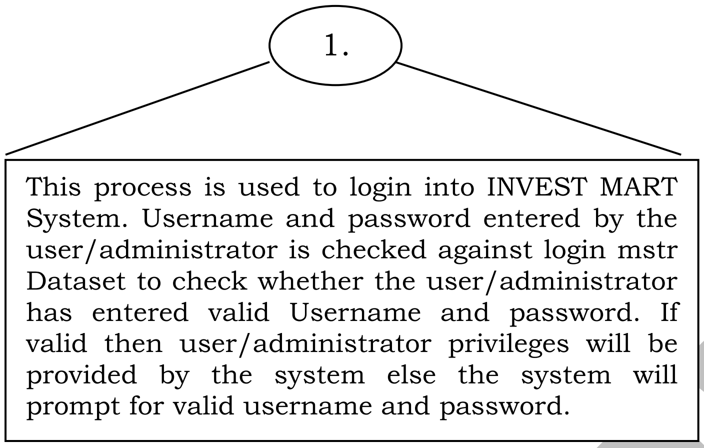
	YES	NO
A process to another process	✓	
A process to an external entity	✓	
A process to a data store	✓	
An external entity to another external entity		✓
An external entity to a data store		✓
A data store to another data store		✓

Data Dictionary:
The analysis model encompasses representations of data objects, function, and control. In each representation data objects and/or control items play a role. Therefore, it is necessary to provide an organized approach for representing the characteristics of each data object and control item. This is accomplished with the data dictionary. The data dictionary has been proposed as a quasi-formal grammar for describing the content of objects defined during structured analysis. This important modeling notation has been defined in the following manner: The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations. Today, the data dictionary is always implemented as part of a CASE "structured analysis and design tool." Although the format of dictionaries varies from tool to tool, most contain the following information:
• Name—the primary name of the data or control item, the data store or an external entity.
• Alias—other names used for the first entry.
• Where-used/how-used—a listing of the processes that use the data or control item and how it is used (e.g., input to the process, output from the process, as a store, as an external entity. Content description - a notation for representing content.
• Supplementary information/Description—other information about data types, preset values (if known), restrictions or limitations, and so forth.
Eg.

Name	Login
Alias	Login mstr
Where/How to use	Data store, used to store the login details of the administrator of the our application
Description	Username + Password

Once a data object or control item name and its aliases are entered into the data dictionary, consistency in naming can be enforced. That is, if an analysis team member decides to name a newly derived data item xyz, but xyz is already in the dictionary, the CASE tool supporting the dictionary posts a warning to indicate duplicate names. This improves the consistency of the analysis model and helps to reduce errors.

Process Specification:
A process specification describes the purpose of processes depicted in the DFD, the inputs to the process, and the output. It describes what the process should do and not how it should do it. However, if the process relates to some decision making then the decision rules used in the process may be described. The decision rules are better depicted by documentation tools such as Decision Table and Decision Tree.
Eg:



This process is used to login into INVEST MART System. Username and password entered by the user/administrator is checked against login mstr Dataset to check whether the user/administrator has entered valid Username and password. If valid then user/administrator privileges will be provided by the system else the system will prompt for valid username and password.

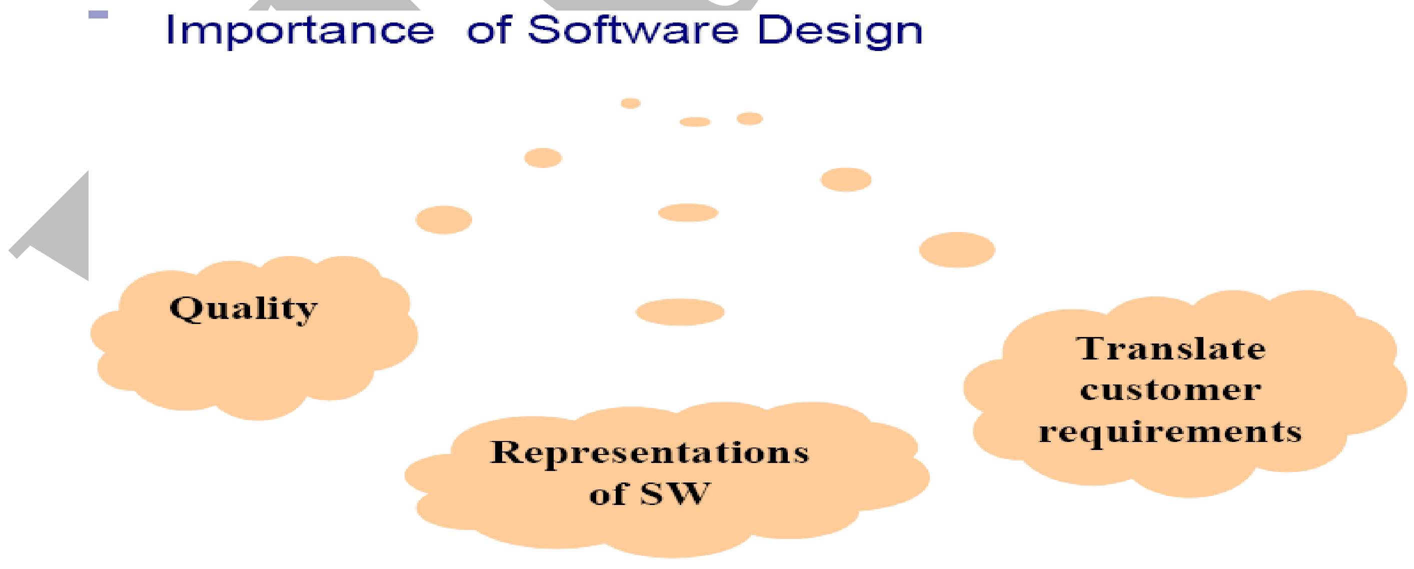
4.3 DESIGN MODEL:

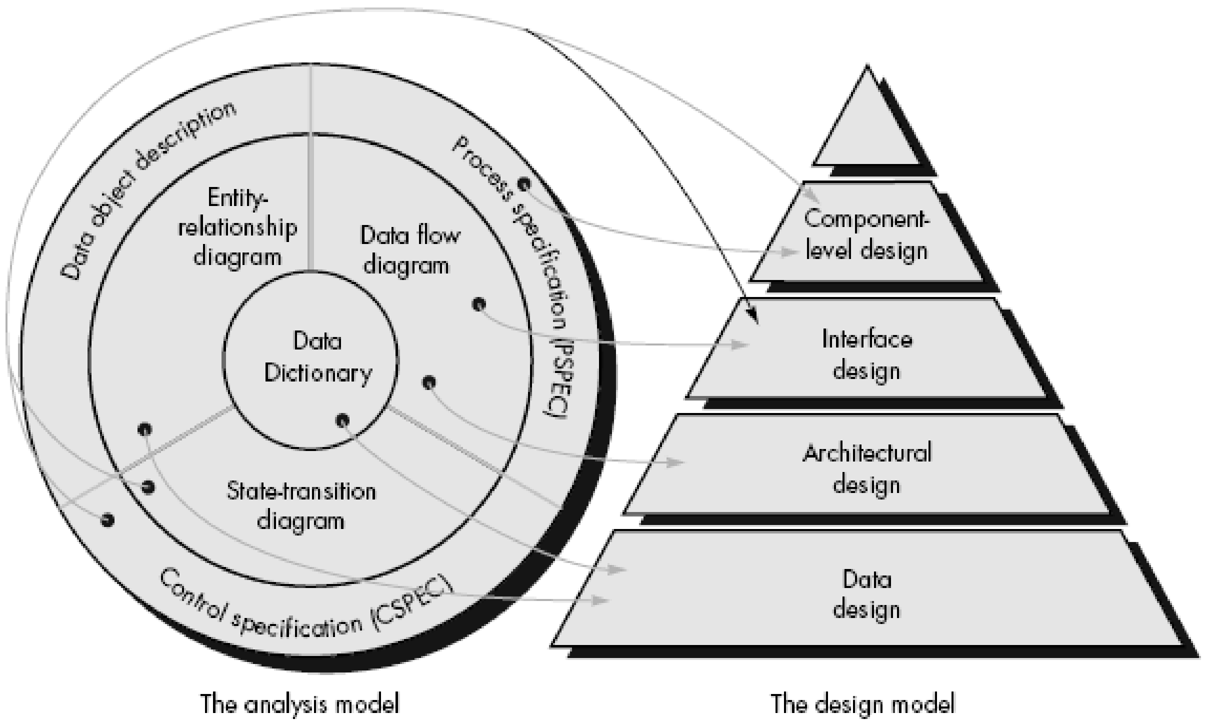
Design is a meaningful engineering representation of something i.e. to be built. It can be traced to the customer requirement and at the same asset for quality against a set of predefines criteria. In the software engineering context design focuses on four measure area of concern.

1. Data Design.
2. Architecture Design.
3. Interface Design.
4. Components Design.

The concepts and principles of design should apply to these all four areas.

Design activity transform information in a manner which is ultimately results in validated computer software. Design task produces data design, an architectural design, an interface design and a component design.


Importance of Software Design
Quality
Representations of SW
Translate customer requirements



1. Data Design: -
   The data design transforms the information domain model which is created during analysis into the data structure that will be required to implement the software. The data objects and relationships defined in the entity relationship diagram and the detail data content which is shown in the data dictionary proving the bases for the data design activity.

2. Architecture Design: -
   The architectural design defined the relationship between major structural elements of the software, the design patterns that can be used to achieve the requirement that have been defined for the system and the constraints that affect the way in which architectural design patterns can be applied. This design represents the frame work of a computer based system can be derived from the system specification.

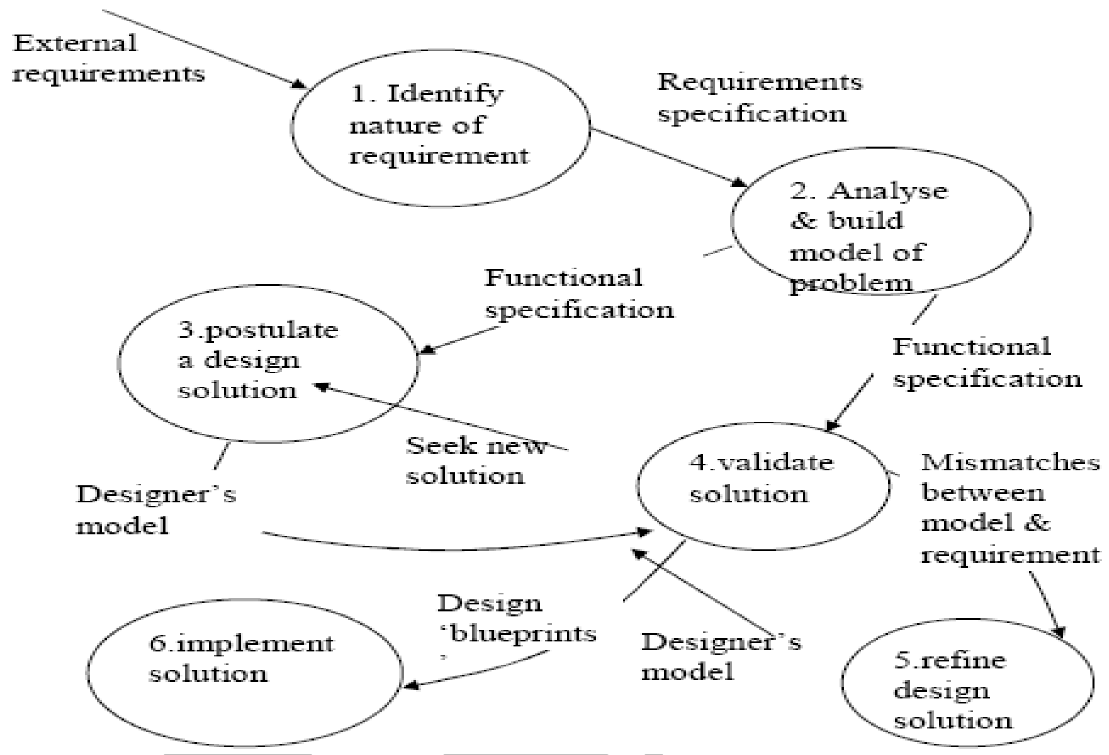
3. Interface Design: -
   The interface design describes how the software communicates within itself with systems that interoperate with it and with people who use it. An interface implies a flow of information. E.g. data or control and a specification type of behavior therefore data and control flow diagram provides much of the information required for interface design.

4. Components Design: -
   The component level design transforms structural elements of the software architectural into a procedural description of software component.

Information obtains from the process specification and control specifications save as the basis for component design.

Design Process: -

- **SW design is an iterative process**
- **Requirements are translated "blueprint" for constructing the SW**



4.4 DESIGN PRINCIPLES:

Software design is both a process and a model. The design process is sequence of steps that enable the designer to describe all aspects of the software to be built. It is important that the design process is not simply sequential process but it is a sense of what makes quality software. The design model is the equivalent of an architectural plant for a house. It begins by representing the total things to be built. Davis has suggested following principles for software.

1. **The design process should not suffer from "tunnel vision"** means a good designer should consider alternative approaches, judging each based on the problem, the resources available to do the job and should consider all design concepts.
2. **The design should be traceable to the analysis model** means a single element of the design model often traces to multiple requirements so it is necessary to have a means for tracking how requirements have been satisfied by the design model.
3. **The design should not reinvent the wheel** means systems are constructed using a set of design patterns. These patterns should always been chosen as an alternative to reinvention because time is short and resources

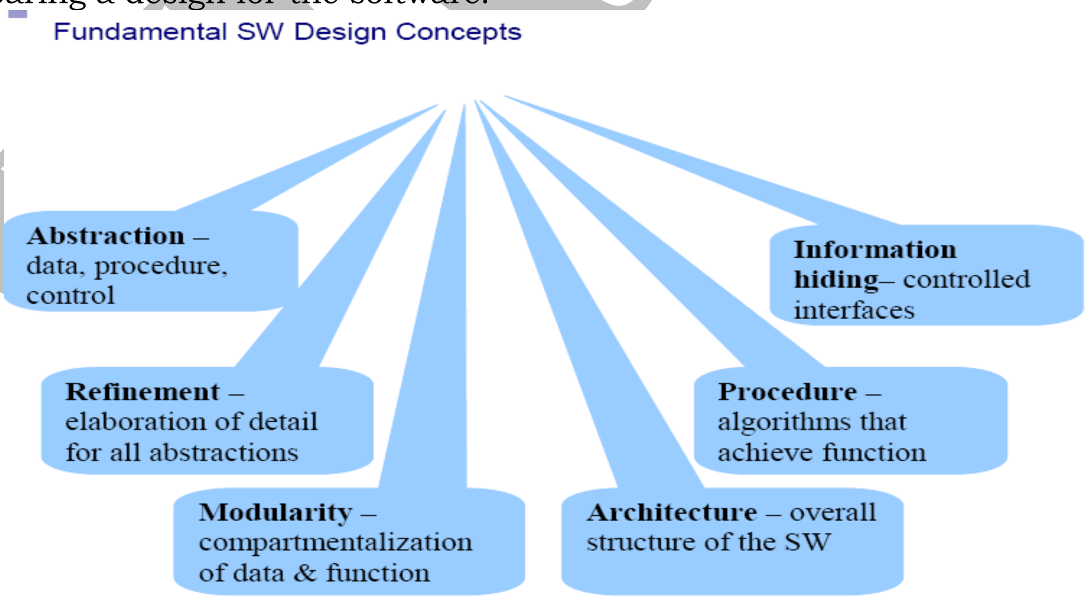
are limited so design time should be invested in representing truly new ideas and integrating those patterns that already exists.

4. **The design should minimize the intellectual distance between the software and the problem as it exists in the real word.**
5. **The design should exhibit uniformly and integration**. Design is uniform means it appears that one person develop the entire think. Rules of style and formats should be defined for a design till before design work begins. Design is integrated if care is taken in defining interface between design components.
6. **The design should be structured to accommodate change** means if we are using prototyping according to above concept.
7. **The design is not a coding and coding is not a design** means even when detail procedural design are created for program components the level of abstraction of the design model is higher than source code.
8. **The design should be received and access for quality as it is being created.** A variety of design concepts and design measures are available to access the designer is accessing quality.

When these design principles are properly applied the software engineer creates a design that exhibit both external and internal quality factors. External quality factors are those properties of the software that can be observed by users. E.g. speed, reliability, concentrate etc. an inter quality factors are of important to software engineer they lead to a high quality design from the technical point of view to achieve internal quality factors. The designer must understand basic design concepts.

DESIGN CONCEPTS: -

Following are some design concepts which designer should keep in mind while preparing a design for the software.


Fundamental SW Design Concepts
Abstraction – data, procedure, control
Information hiding– controlled interfaces
Refinement – elaboration of detail for all abstractions
Procedure – algorithms that achieve function
Modularity – compartmentalization of data & function
Architecture – overall structure of the SW

1. **Abstraction:** *-*

Concentrate on the essential features and ignore details that are not relevant Means hiding the complexity.

1) Procedural Abstraction: -
It is name sequence of instructions that has a specific and limited function. E.g. procedural abstraction would be the word open for a door for open implies a long sequence of procedural steps like walk to the door, reached out and knocked and pull door and step away from moving door etc.
Means in-sort hiding the Procedure details here hiding the procedure of how to open door.

2) Data Abstraction*:* -
Data abstraction is a name collection of data that describes a data object in the context of the procedural abstraction open we can define a data abstraction called door like any data object. The data abstraction for door would be a set of attributes that describe the door. E.g. Door type, swing direction, opening mechanism, weight dimension etc. it follows that the procedural abstraction would make use of information contain in the attributes of the data abstraction door.
Means in-sort hiding the data details here hide the door details.

3) Control abstraction:
Implies a program control mechanisms without specify internal details.
Means in-sort hiding the Control Details.

2. Refinements: -

The process of program refinement (modification or enhancement) is a partitioning process i.e. used during requirement analysis. Refinement is actually a process of elaboration (expansion). we begin with a statement of function or description of information i.e. defines a high level of abstraction. The statement describes function or information conceptually but provides no information about the internal working of the functions or the internal structure of the information. Refinement causes the system designer to elaborate on the original statement providing more and more detail as each successive refinement occurs.

Abstraction and refinement are complimentary concepts. Abstractions enable a designer to specify procedure and data refinement helps the designer a detail at a low level.

3. Modularity: -

Software is divided into separately named and addressable components which are called modules. Those are integrated to satisfy problem requirements. Modularity is a single attribute of software that allows a program to be intellectually manageable. Monolithic software i.e. a large program composed of a single module cannot be easily readable the number of control paths, spend of referential number of variables and overall complexity would make understanding close of impossible.

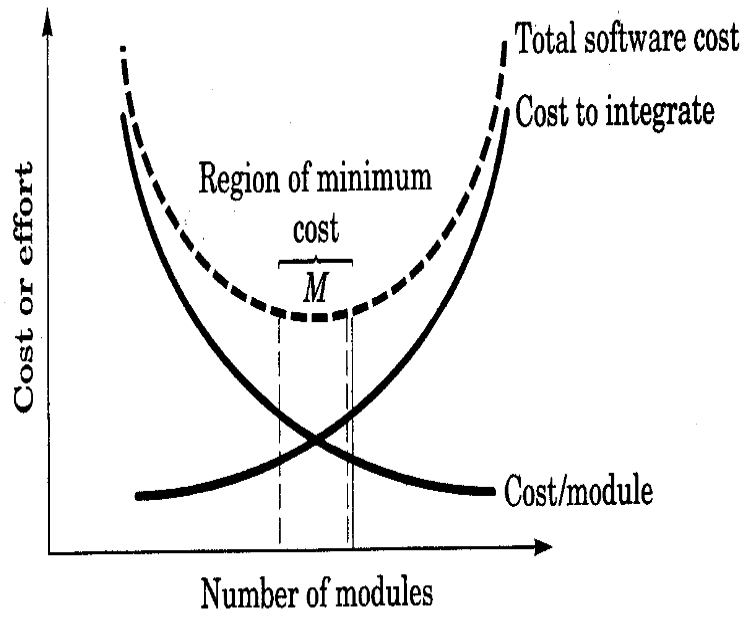


FIGURE 13.2. Modularity and software cost

it is easier to solve a complex problem when you break it into manageable PCs(Modules). When we divide software in to modules then development effort also decreased. From above graph we can show that as the number of modules grows the effort or cost associated with integrating the module also grows. These characteristics leads to a total cost or effort which shown in above figure. This is a M of modules that would result in minimum development cost and here another important question arises when modularity is consider how do we define an appropriate module of a given size? The answer lies in the method use to define modules within a system. These are arteries define that enable us to evaluate a design method with respect to define an effective module system.

1) Modular Decomposability: -
Provides a systematic approach for decomposing the problem into sub-problems.
If a design method provides a systematic mechanism for decomposable problem into the sub problem, then we can reduce the complexity of overall problem by achieving effective modularity software.
2) Modular Composability: -
Modular compos ability means if a design method enable existing or reusable design components to be assembled into a new system then the modular solution does not reinventing wheel.

3) Modular Understandability: -

If a module can be understood as a standalone unit without reference to other module then it will be easier to built and easier to change.

4) Modular Continuity: -
If small changes to the system requirements results in changes to individual modules rather-then system wise changes the impact of change all other side effects should be minimized.

5) Modular Protection: -
If any unexpected condition occurs within a module and its effects are contain within that module only, the impact of other side effects should be minimized.

4. Software Architecture: -
Architecture is the hierarchical structure of program components or modules. There are some set of properties that should be specified as an architectural design.

1) Structural Properties: -
This aspect of the architectural design represents the components of a system.

2) Extra Functional Properties: -
The architectural design description should address how the design architecture achieves requirements for performance, capacity, security and other system characteristics.
The architectural design can be represented using one or more different models.

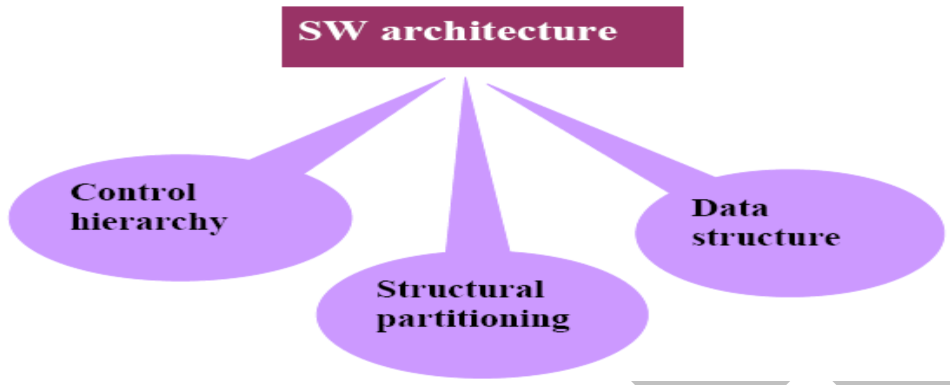
i. Structural Model: -
Which is represent architectural as an organized collection of program components.

ii. Framework Model: -
Increase the level of design abstraction by attempting to identify repeatable architectural design frame works.

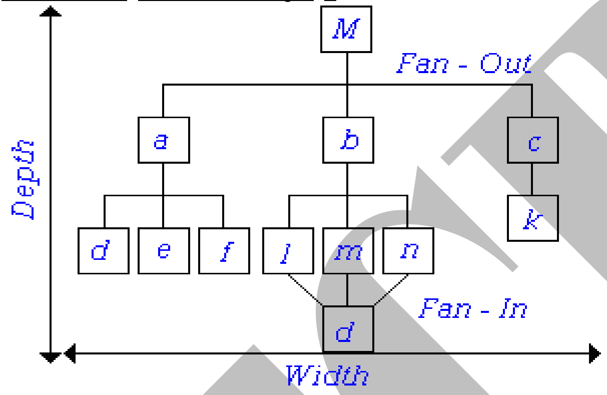
iii. Dynamic Model: -
Address the behavioral aspects of the program architecture it indicates how the structure or system configuration may change as a function of external events.

iv. Process Model: -
Focus on the design of the business or technical process that system must accommodate.

v. Functional Model: –
It can be used to represent the functional hierarchy of the system.



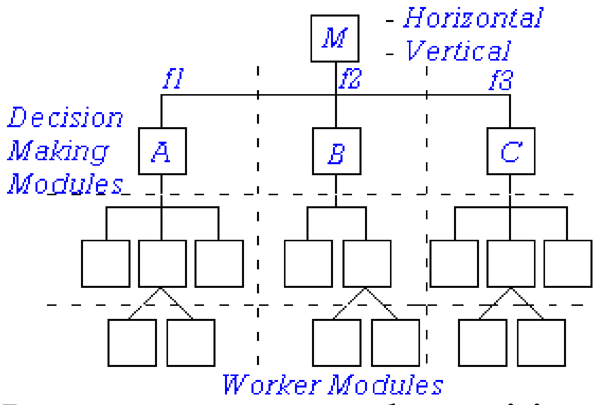
Control Hierarchy: -



The 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 software, occurrence, and order of decision or representation of operations.

In above **depth and width** provide an indication of the number of levels of control and overall span of control. **Fan – out** is a major of the number of modules that are directly controlled by another module. **Fan – in** indicate how many modules directly control a given module? The control relationship among modules is expressed in the following way. **A module that controls another module is said to be super ordinate** to it and a **module controlled by another is said to subordinate** to the controller. In above figure module **M is super-ordinate to modules a, b and c** and module **k is subordinate to module c.**

Structural Partitioning: -



Program structure can be partitioned both horizontally and vertically.

Horizontal partitioning: defines separate principles branches of modular hierarchy for each major program function. Control modules represents each coordination between remaining modules and execution of the function, partitioning the horizontally provides following benefits.

1) Software i.e. easier to test.
2) Software i.e. easier to maintain.
3) Software i.e. easier to extend.
4) Propagation of fewer side effects or less side effects.

Vertical partitioning: is also called factoring we suggest that control and work should be distinguished top down in program structure. Top level modules should perform control functional do less actual processing word, modules that reside low in structure are called the worker modules performing all input computation and output task.

Data Structure: -

Data structure is a representation of logical relationship among individual elements of data. Data structure is an important program structure to the representation of software architecture.

Data structure shows the organization of methods to access a scalar item is the simplest form of all data structure. A scalar item represents a single element of information which is addressed by an identifier and i.e. accessed by specific a single address in memory. When scalar items are organized as a list of continuous group then a sequential vector is formed, when the sequential vector is extended into two or three or an arbitrary number of dimension then a n dimension space is created and the most common n dimension space is two dimensional matrix and n dimension space is also called an array and a link list is a data structure that organized the memory elements into a noncontiguous scalar item or vector.

5. **Software Procedure: -**

Focus on the processing details of each module. Procedure must provide a exact specification of processing, including sequence of events, exact decision points, repetitive operation and even data organization and structure, there is relationship between structure and procedure.

6. **Information Hiding: -**
   The principle of information hiding suggest that modules should be specify and design so that **procedure and data contain within a module is in accessible to other modules that have no need for such information** hiding implies that effective modularity can be achieved by a set of independent module.

Design Documentation: -

Design document will cover all design aspects as a document.

1. Design Specification: -
   It addresses different aspects of the design module and it is completed as the designer refines this representation of the software. First the overall scope of the effort is describe here most of the information presented is derived from the system specification and the SRS.

2. Data Design: -
   It describes the data structure, any external file structure; internal data structure and a cross reference that connects data objects to a specific file are defined.

3. Architectural Design: -
   It indicates how the program architecture has been derived from the analysis model.

4. Interfaces: -
   The external and internal program interface is represented and detail design of the human machine interface describe.

5. Components: -
   It is also known as separately addressable elements of software such as subroutines, functions or procedures.

6. Cross Reference: -
   The design specification contains a requirement cross reference. The purpose of the cross reference normally it is represented as a simple matrix.
   1) To established that all requirements are satisfied by the software design.
   2) To indicate that which components are critical to the implementation of specific requirement.

7. Design Constraints: -

Design constraints such as physical memory limitation or the necessity for a specialized external interface may dictate special requirements for packaging or assembling software.

8. Final Section: -
The final section contain supplementary data, algorithm description, alternative procedures, tabular data and other relevant information are presented as supplementary nodes.