DESIGN ENGINEERING

PRINCIPLES OF SOFTWARE DESIGN

: Software design is a phase in S/E in which a blue print is developed to serve as a base for constructing the S/w system.

a) Choose the right Programming Paradigm -

Procedural Paradigm object oriented Prototype Paradigm

b) S/w design should be uniform & integrated

c) S/w design should be flexible - Must be able to adopt changes easily

d) S/w reuse

e) Designing for Testability

SOFTWARE DESIGN CONCEPTS

1. **ABSTRACTION** - refers to Powerful design tool, which allows S/w designers to consider components at an abstract level. while neglecting the implementation details of the components

Abstraction can be used in two ways - ① Process ② Entity

Process: mechanism of hiding irrelevant details & representing essential features

Entity: refers to a model or view of an item

Abstraction mechanisms

a) **Functional abstraction** - Can be generalized as collection of subprograms referred to as "Groups"

b) **Data Abstraction** - Specifying data that describes a data object

Attributes of data objects is ignored

c) **Control Abstraction** - states desired effect, without stating the exact mechanism of control

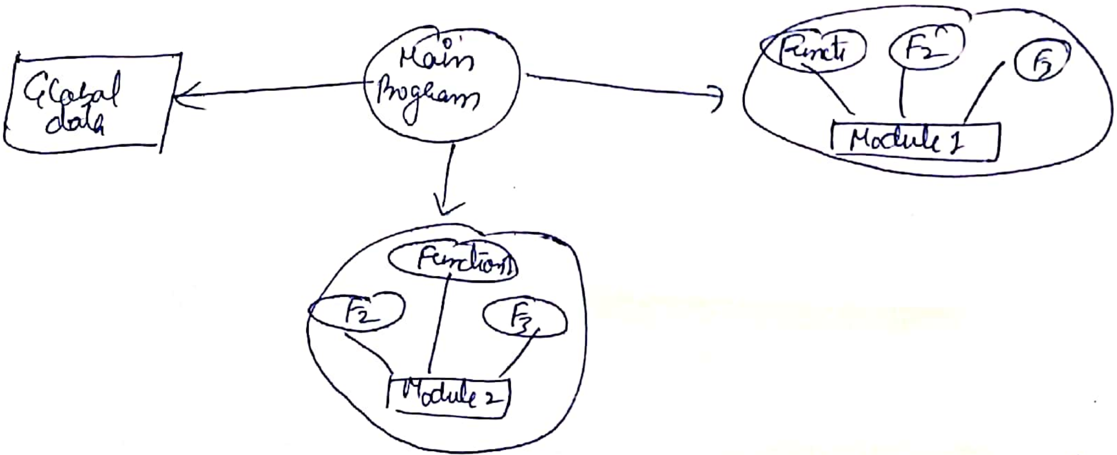
2) ARCHITECTURE - refers to the structure of the System which is composed of various components like System, the attributes of those Compon
& relationship amongst them.

- The S/w architecture does the follawing
  - Provides an insight to all the interested Stakeholder's
  - Highlights early design decisions
  - Creates intellectual models of how the System is organised into components & how these components interact with each other.

③ PATTERN - Provides a description of the Solution to a recurring design Problem of some specific domain in such a way that the Solution can be used again & again

④ Modularity - Modularity is achieved by dividing the S/w into uniquely named & addressable components also known as Modules

A complex system (Large Program) is ~~Partition~~ Partitioned into a set of descrete modules in such a way that each module can be developed independent of other modules. Later can be integrated Together to meet S/w requirements



⑤ Information hiding - refers to way of hiding unnecessary details

Modules should be specified & designed in such a way that data structure & Processing details of one module are not accessible to other modules.

They Pass only that much infor^n to each other which is required to accomplish the S/w functions.

- Information hiding is of immense use when modifications are required during the testing & maintenance phase.

Advantages associated with Information hiding

1. Leads to low Coupling
2. Decreases the Probability of adverse effects
3. Results in higher Quality S/w
4. Emphasizes communication through controlled Interfaces.

6. Stepwise Refinement - is a top-down design strategy used for decomposing a system from a high level of abstraction into a more detailed level.

Let us consider an example of Stepwise refinement
Every computer Program comprises Input, Process & output

a) Input
Get user's name (string) through a Prompt

b) Process - Computation

c) output - result

(7) Refactoring - is a reorganization technique that simplifies the design (Internal code structure) of a component without changing its function or external behaviour.

- It removes
  - Redundancy
  - unused design elements
  - Inefficient or unnecessary algorithm
  - Inappropriate data structures

(8) **Functional independence** – is achieved by developing modules with "single-minded" function & an "aversion" to excessive interaction with other modules.

Functional independence is accessed using two criteria

COHESION

- is the indication of the relationship within module

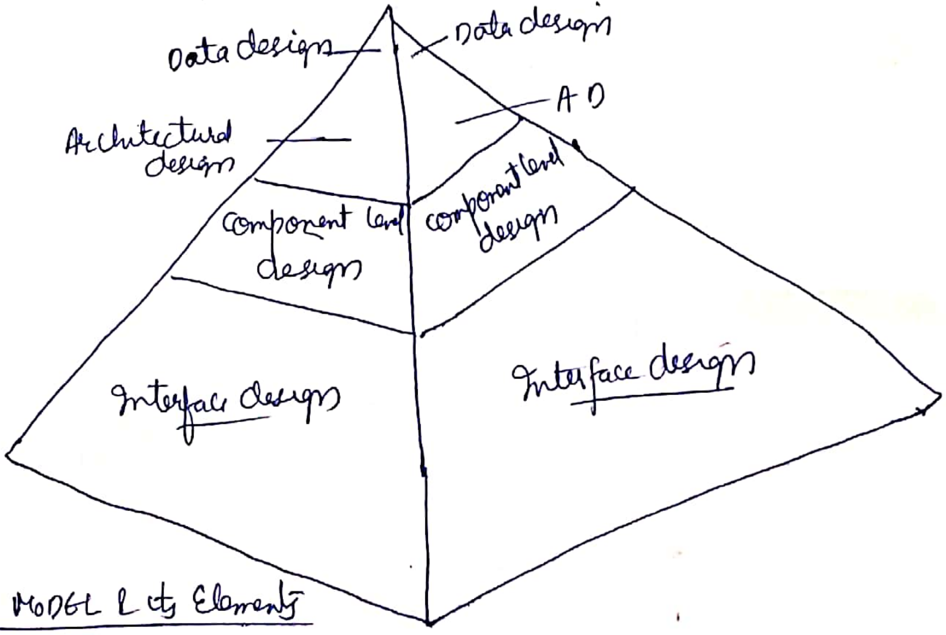
COUPLING

- degree of interdependence b/w S/w modules, measure of how closely connected two modules are

DEVELOPING A DESIGN MODEL

- To develop a complete specification of design, four design models are needed.

1. **Data design** – This specifies data structures for implementing the S/w by converting data objects & their relationships identified during analysis phase
2. **Architectural design** – This specifies relationship b/w the structural elements of the S/w, design patterns, architectural styles & factors affecting the way in which architecture can be implemented
3. **Component level design** – Provides detailed description of how structural elements of S/w will actually be implemented.
4. **Interface design** – depicts how the S/w communicates with the system that interoperates with it & with end users



DESIGN MODEL & its Elements

DESIGNING CLASS-BASED COMPONENTS

COMPONENT LEVEL DESIGN PRINCIPLES:

BASIC DESIGN PRINCIPLES

a) • **OPEN - CLOSED Principle** - A module or component should be open for extension but closed for modification

- The designer should specify the component in a way that allows it to be extended without the need to make internal code or design modifications to the existing Parts of the Component

b) **LISKOV SUBSTITUTION PRINCIPLE**

- Subclasses should be substitutable for their base classes.
- A component that uses a base class should continue to function properly if a subclass of the base class is passed to the component instead.
- This Principle says that inheritance should be well designed & well behaved

c) **Dependency Inversion Principle**

- Depend on abstractions (i-e interfaces), do not depend on concretions
- The more a component depends on other concrete components, the more difficult it will be to extend

d) **Interface Segregation Principle**

- Many client-specific interfaces are better than one general purpose interface.
- Imp For a Server class, specialized interfaces should be created to serve major categories of clients
- Only those operations that are relevant to a particular category of clients should be specified in the interface.

e) Release reuse Equivalency Principle

- Group The reusable classes into Packages That Can be managed, upgraded & controlled as newer versions are Created

f) Common closure Principle –

- Classes that change together belong together
- Classes should be Packaged cohesively; they should address the Same functional or behavioural area on the assumption that if one Class experiences a change then They all will experience a change

g) Common Reuse Principle –

- Classes that aren't reused together should not be grouped together

COMPONENT LEVEL DESIGN GUIDELINES

1. COMPONENTS – Naming conventions should be used
   - It is the basic building block for Computer S/w
   - It is higher level abstraction

2. Interfaces – Provides important information about communication & callaboration.

3 Dependencies & Inheritance –
   - It is good idea to model dependencies from left to right
   - And ~~for~~ Inheritance from bottom (derived class) to Top (base classes)

CHAPTER : USER INTERFACE ANALYSIS & DESIGN

USER INTERFACE - is the front end application view to which user interacts in order to use the S/w. The S/w becomes more popular if its user interface is

a) Attractive b) Simple to use c) Clear to understand
d) Responsive in short time

Two types of user Interfaces

① CLI ② GUI

The Golden Rules - stated by Theo Mandel that must be followed during the design of the Interface

a) Place the user in Control

- Define the interaction modes in such a way that does not force the user into unnecessary or undesired actions.
- Provide for flexible interaction like mechanism to use keyboard, commands, use mouse, or touch screen
- Allow user interaction to be interruptable & undo able
- Design for direct interaction with user's that appear on screen

b) Reduce the user's memory load

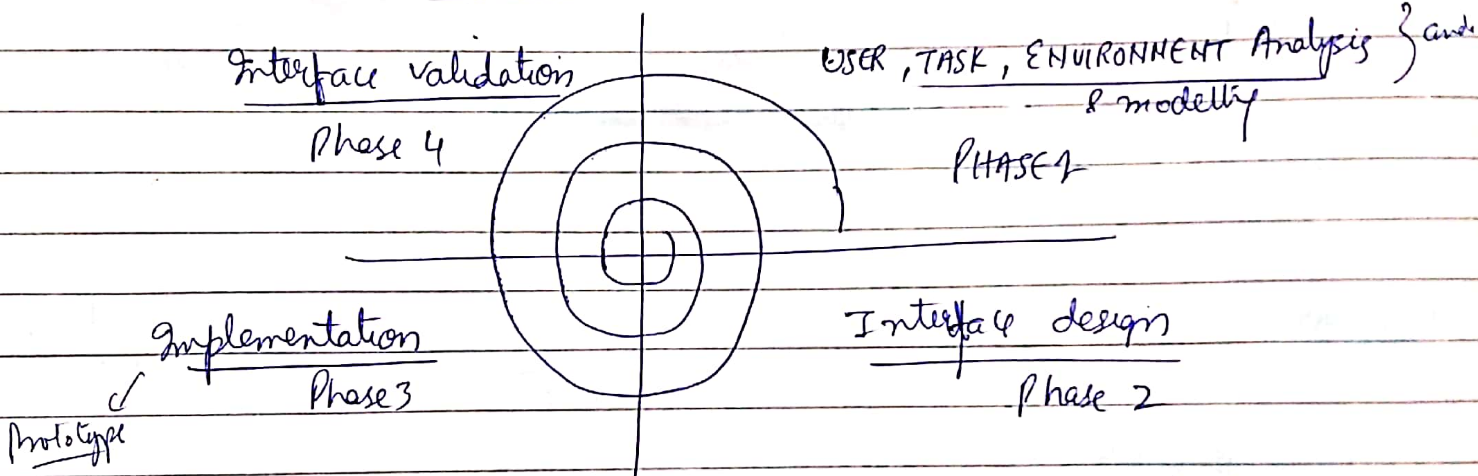
- Reduce demand on short term memory - The interface should be designed in such a way to reduce the remembering of previously done actions, given inputs & results (like Browser)
- Establish meaningful defaults - Always initial set of defaults should be provided to the average user.
- Define short cuts that are intutive : Mnemonics should be used by user. It means keyboard shortcuts

to do some action on the screen.

(3) Make the Interface consistent –

- Maintain consistency across a family of applications
  A set of applications should all implement the same design rules, so that consistency is maintained for all interactions

USER Interface design Process



- The analysis & design process of a user Interface is Iterative & can be represented by a spiral model.
- Consists of 4 framework activities

1. user, Task, Environment analysis & Modelling
   - Initially, the focus is based on the Profile of users who will interact with the system
   - Once all requirements are gathered, the task that the user performs to establish the goals of the system are identified
   - The analysis of user environment focuses on the physical work environment like
     - ~~[illegible]~~ where will the Interface be located physically
     - Does the Interface h/w accomodates space, light or noise constraints

   Example – Biometric attendance System

(2) Interface design - Goal is to define the set of Interface objects & actions (3)
This Phase serves as the foundation for the Implementation phase

(3) Interface construction & Implementation

The Implementation actually begins with creation of Prototype (model). As Iterative design Process continues, a user interface toolkit that allows the creation of windows, menus, device interaction can be used for completing the construction of an interface

(4) Interface validation - This Phase focuses on testing the Interface.

Interface Analysis & Interface design steps

Interface analysis - - In case of user Interface design, understanding the problem means understanding

a) The People (end users) who will interact with the system through interface

b) The task that end users must perform to do their work

c) The environment in which these tasks will be conducted

Interface design steps

- once interface analysis has been completed, all tasks required by end user have been identified in detail & Interface design actually commences.

- Interface design is an Iterative Process

Design steps

1. using information developed during Interface analysis, define interface objects & actions

2. Define events (user actions) that will cause the state of the user Interface to change

3. Depict each interface state as it will actually look to an end user

4. Indicate how the user interprets the state of the system from Inforⁿ Provided Through The Interface